Chapter #3: Tilt with the Memsic Accelerometer

Acceleration is a measure of how quickly speed changes. Just as a speedometer is a meter that measures speed, an accelerometer is a meter that measures acceleration. You can use an accelerometer's ability to sense acceleration to measure a variety of things that are very useful to electronic and robotic projects and designs. Here are some examples.

- Acceleration
- Tilt and tilt angle
- Incline
- Rotation
- Vibration
- Collision
- Gravity

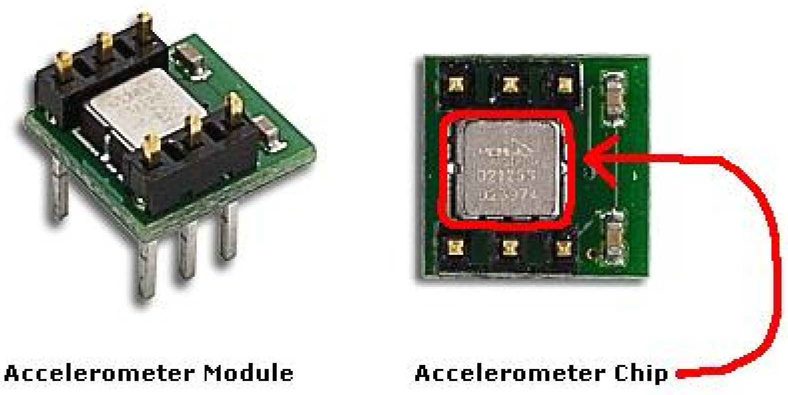
Accelerometers are already used in a wide variety of machines, specialized equipment and personal electronics. Here are just a few examples:

- Self balancing robots
- Tilt-mode game controllers
- Model airplane auto pilot
- Car alarm systems
- Crash detection/airbag deployment
- Human motion monitoring
- Leveling tool

Once upon a time, accelerometers were large, clunky and expensive instruments that did not lend themselves to electronic and robotic projects. This all changed thanks to the advent of MEMS, micro-electro-mechanical-systems. MEMS technology is responsible for an ever increasing number of formerly mechanical devices designed right onto silicon chips.

The accelerometer you will be working with in the forthcoming activities is the Parallax Memsic 2125 Dual Axis Accelerometer module shown in Figure 1. This module measures less than $^1/_2$" X $^1/_2$" X $^1/_2$", and the accelerometer chip itself is less than $^1/_4$" X $^1/_4$" X $^1/_8$".

Figure 3-1 Accelerometer Module and MX2125 Chip



People naturally sense acceleration on three axes, forward/backward, left/right and up/down. Just think about the last time you were in the passenger seat of a car on a hilly and curvy road. Forward/backward acceleration is the sensation of speeding up and slowing down. Left/right acceleration involved making turns, and up down acceleration is what you felt going over hills.

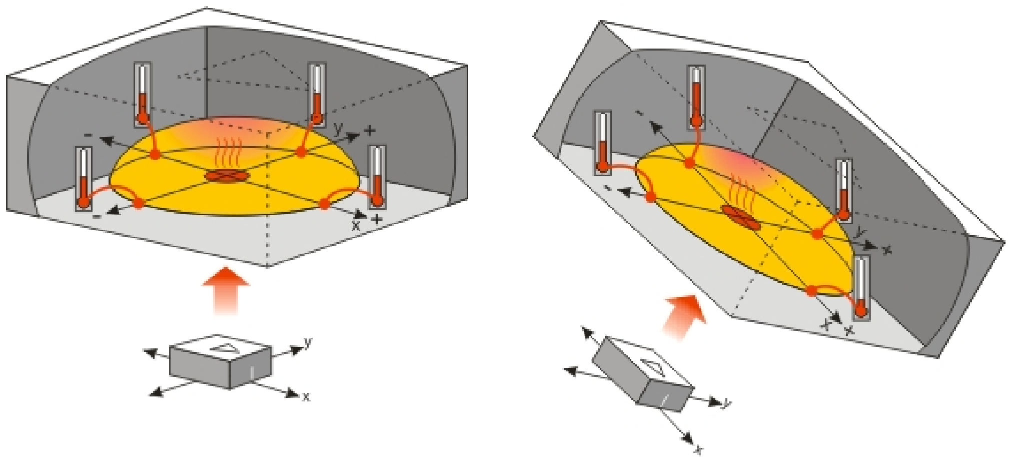
Up/down acceleration is also the way we sense gravity. When on the ground, people tend to sense gravity as their own weight. In free-fall, they sense gravity as weightlessness. In physics terms, gravity is a form of acceleration. When an object is on the ground, gravity is sometimes called static acceleration. When an object is rolling down hill or falling, gravity becomes dynamic acceleration.

Instead of the three axes people sense, the MX2125 accelerometer senses acceleration on two axes. The acceleration it senses depends on how it's positioned. By holding it one way, it can sense forward/backward and left/right. If you hold it a different way, it can sense up/down and forward/backward. Two axes of acceleration is enough for many of the applications listed earlier. While you can always mount and monitor a second accelerometer to capture that third axis, three-axis accelerometers are also common.

THE MX2125 ACCELEROMETER – HOW IT WORKS

The MX2125's design is amazingly simple. It has a chamber of gas with a heating element in the center and four temperature sensors around its edge. Just as hot air rises and cooler air sinks, the same applies to hot and cool gasses. If you hold the accelerometer still, all it senses is gravity, and tilting it gives us an example of how it senses static acceleration. When you hold the accelerometer level, the hot gas pocket rises to the top-center of the accelerometer's chamber, and all the temperature sensors measure the same temperature. Depending on how you tilt the accelerometer, the hot gas will collect closer to one or maybe two of the temperature sensors.

Figure 3-2 Accelerometer Heated Gas Pocket



Both static acceleration (gravity and tilt) and dynamic acceleration (like taking a ride in a car) are detected by the temperature sensors. For example, if you take the accelerometer for a car ride, the hotter and cooler gasses slosh around in the chamber in a manner similar to a container that is partially filled with water.

In most situations, making sense out of these measurements is a simple task thanks to the electronics inside the MX2125. The MX2125 converts the temperature measurements into signals (pulse durations) that are easy for the BASIC Stamp microcontroller to measure and decipher.

ACTIVITY #1: CONNECTING AND TILT-TESTING THE MX2125

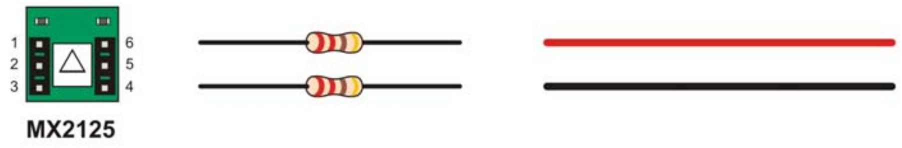
In this activity, you will connect the accelerometer module to the BASIC Stamp, run a test program, and verify that it can be used to sense tilt.

Accelerometer Parts

The parts you will need for this activity are listed here, and Figure 3 shows their drawings.

Parallax Part Number	**Quantity**	**Description**
800-00016	(2)	3-inch Jumper wires
150-02210	(2)	Resistor – 220 Ω
28017	(1)	Memsic MX2125 Dual-Axis Accelerometer

Figure 3-3 Accelerometer Part Drawings

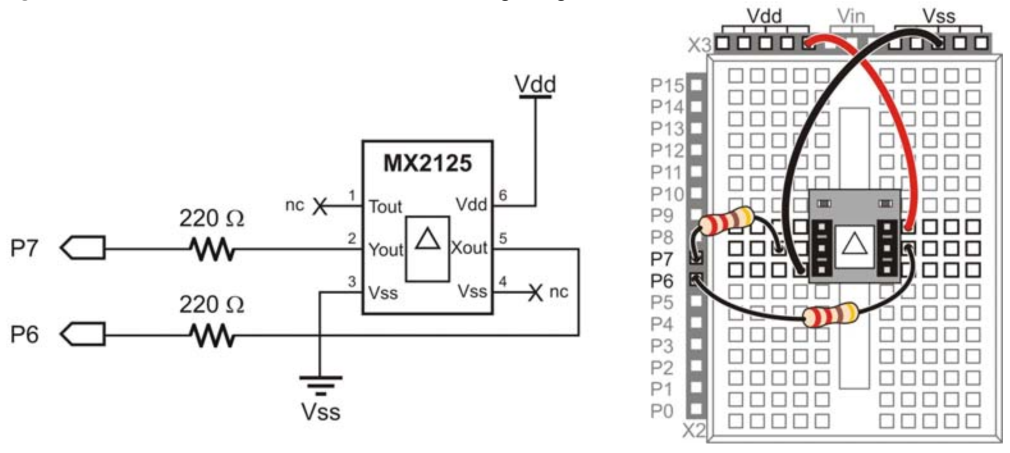


Accelerometer Electrical and Signal Connections

Figure 4 shows how to connect the accelerometer module to the Board of Education's power supply along with the BASIC Stamp I/O pin connections you will need to make to run the example program.

√ Connect the accelerometer module using the schematic and wiring diagram as your guides.

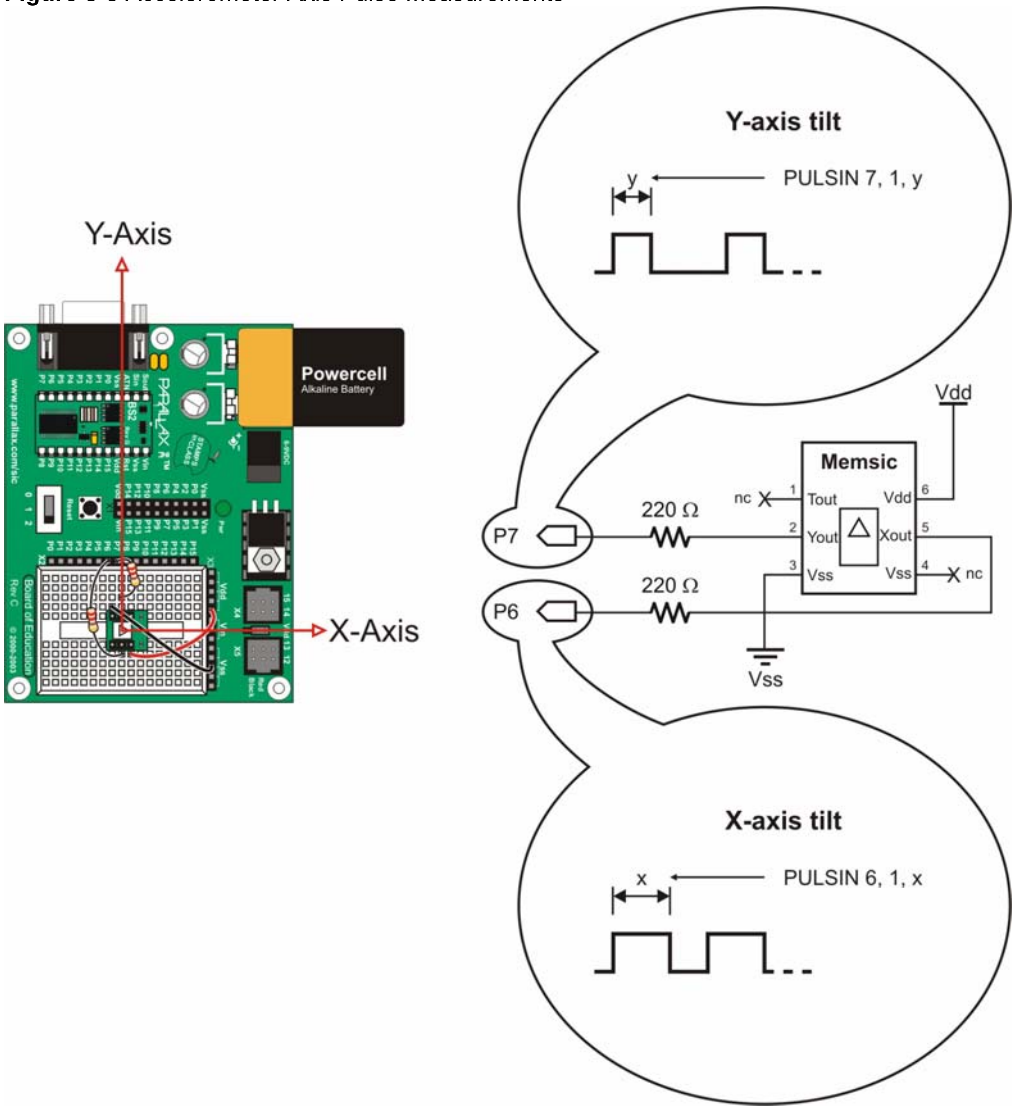
Figure 3-4 Accelerometer Schematic and Wiring Diagram



Listening to the Accelerometer's Signals with the BASIC Stamp

The two axes the MX2125 uses to sense gravity and acceleration are labeled x and y in Figure 5. It will help if you set your board flat on the table in front of you as shown in the figure. That way, the x and y axes point the same directions they do on most xy plots. For room temperature testing, you can get a pretty good indication of tilt by just measuring the high times of the pulses sent by the MX2125's Xout and Yout pins with the **`PULSIN`** command. Depending on how far you tilt the board and in which direction, the **`PULSIN`** time measurements should range from 1875 to 3125. When the board is level, the **`PULSIN`** command should store values in the neighborhood of 2500.

Figure 3-5 Accelerometer Axis Pulse Measurements



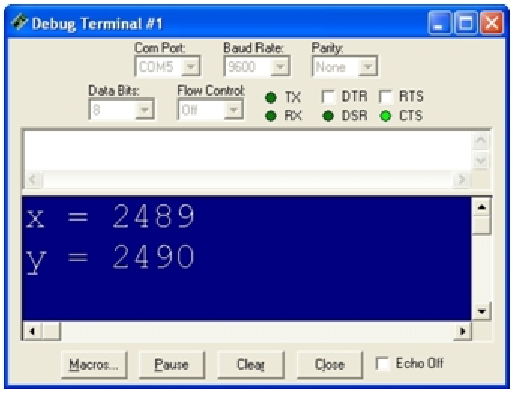
√ Make sure your board is sitting flat on the table, oriented with its x and y axes as shown in the Figure 5.

√ Enter and run SimpleTilt.bs2.

```
' SimpleTilt.bs2
' Measure room temperature tilt.

'{$STAMP BS2}
'{$PBASIC 2.5}

x               VAR     Word
y               VAR     Word

DEBUG CLS

DO

  PULSIN 6, 1, x
  PULSIN 7, 1, y

  DEBUG HOME, DEC4 ? X, DEC4 ? Y

  PAUSE 100

LOOP
```

√ Check to make sure the Debug Terminal reports that the x and y variables are both storing values around of 2500.

Figure 3-6 Debug Terminal Output



```
x = 2489
y = 2490
```

√ Grab the edge of the board with the Y-Axis label and gradually lift it toward you. The y value should increase as you increase the tilt.

- √ Keep tilting the board toward you until it's straight up and down. The Debug Terminal should report that the y variable stores a value near 3125.
- √ Lay the board flat again.
- √ Next, instead of tilting the board toward you, gradually tilt it away from you. The y-axis value should drop below 2500 and gradually decrease to 1875 as you tilt the board until it's straight up and down.
- √ Lay the board flat again.
- √ Repeat this test with the x-axis. As you tilt the board up with your right hand, the x value should increase and reach a value near 3125 when the board is vertical. As you tilt the board upward with your left hand, the x value should approach 1875.
- √ Finally, hold your board in front of you, straight up and down like a steering wheel.
- √ As you slowly rotate your board, the x and y values should change. These values will be used in another activity to determine the rotation angle in degrees.

ACTIVITY #2: MOBILE MEASUREMENTS

This activity demonstrates displaying the Memsic Accelerometer's measurements on the Parallax Serial LCD. Provided you're using a battery, you can disconnect from your computer and take the setup to remote locations of your choosing.

Connecting Both Modules to the BASIC Stamp

Both the Memsic Accelerometer and the Serial LCD can fit on your board at the same time, so there will be no need for extension cables unless you chose to mount the Parallax Serial LCD near the board of education or in a project box.

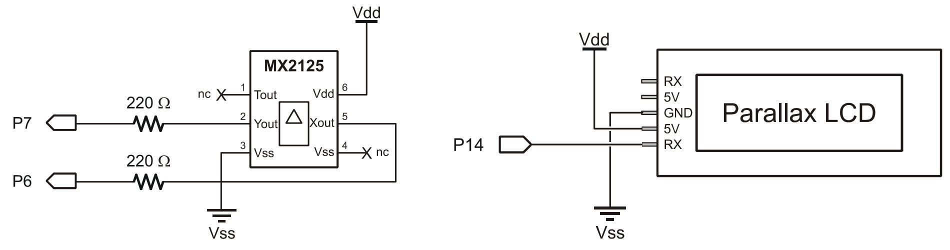
Parts List

(1) Memsic 2125 Accelerometer
(1) Parallax Serial LCD (2×16)
(5) Jumper Wires
(2) 220 Ω resistors (red red brown)

Building the Accelerometer and LCD Circuits

The schematics shown in Figure_Next are identical to the ones that have been used for the Memsic accelerometer and Parallax Serial LCD in previous activities.

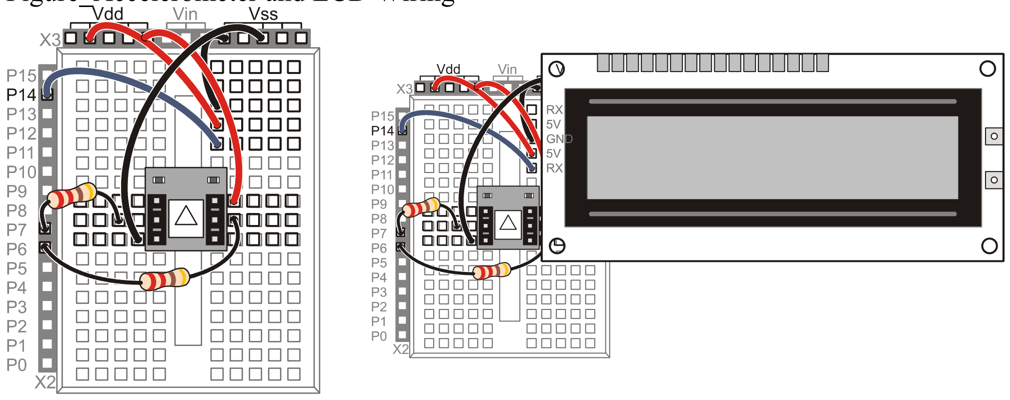
Figure_ - Ping))) and Parallax Serial LCD Schematics



The wiring diagrams for the Memsic Accelerometer and Parallax Serial LCD in Figure_Next are a combination of the two earlier wiring diagrams for the individual modules.

√ Build the wiring diagram on the left first, then insert the Parallax Serial LCD as shown on the right.

Figure_Accelerometer and LCD Wiring



LCD Tilt Display

Modifying any of the accelerometer example programs from this chapter to make them display measurements on the LCD is typically a matter of adding the LCD initialization routine and then replacing DEBUG commands with SEROUT commands that will display the information on the LCD.

Always remember to add this initialization, either before the main routine, or in small programs, before the first DO keyword. That will keep the initialization from being repeated over and over again in the DO...LOOP with the rest of the program. Make sure to keep it out of the main DO...LOOP because it could cause the display to flicker.

```
' Initialize LCD
PAUSE 200
SEROUT 14, 84, [22, 12]
PAUSE 5
```

Next, the DEBUG commands need to be changed to SEROUT commands. Here is the DEBUG command from SimpleTilt.bs2.

```
DEBUG HOME, DEC4 ? X, DEC4 ? Y
```

The HOME control should be replaced with 128, which is the LCD's home character. The ? directive displays the variable name, and then a carriage return (CR) character afterwards. Remember from Chapter #1 that CR is the one control character that happens to be the same for both the Debug Terminal and the Parallax Serial LCD? Because of this, we can leave the ? directive in the SEROUT commands to the LCD. Here is a SEROUT command that does the equivalent display on the Parallax Serial LCD.

```
SEROUT 14, 84, [128, DEC4 ? X, DEC4 ? Y]
```

Example Program: SimpleTiltLcd.bs2

This program is a modified version of SimpleTilt.bs2 from the previous activity. Instead of displaying its measurements in the Debug Terminal, it displays them in the Parallax Serial LCD.

√ Connect a battery to your board.
√ Enter, save and run SimpleTiltLcd.bs2.
√ Disconnect the serial cable, and take your board with you to wherever you want to test the Memsic Accelerometer's measurements.

```
' SimpleTiltLcd.bs2
' Measure room temperature tilt and display it with the
' Parallax Serial LCD.

'{$STAMP BS2}
'{$PBASIC 2.5}

x              VAR     Word
y              VAR     Word

' DEBUG CLS
' Initialize LCD
PAUSE 200
SEROUT 14, 84, [22, 12]
PAUSE 5

DO

  PULSIN 6, 1, x
  PULSIN 7, 1, y

'  DEBUG HOME, DEC4 ? X, DEC4 ? Y
  SEROUT 14, 84, [128, DEC4 ? X, DEC4 ? Y]

  PAUSE 100

LOOP
```

Your Turn - Customizing the Display

The carriage return (CR) that's built into the ? operator makes it more difficult to display information after the x or y values. You can rewrite the DEBUG and SEROUT to perform the same operations like this.

```
DEBUG HOME, "x = ", DEC4 x, CR, "y = ", DEC4 y
```

This SEROUT command displays the same information on the Parallax Serial LCD. Notice how the 128 places the cursor on Line-0, character-0. Instead of a CR control character, 148 places the LCD's cursor on Line-1, Character-0.

```
SEROUT 14, 84, [128, "x = ", DEC4 x, 148, "y = ", DEC4 y]
```

With this modified SEROUT command, it's easier to display characters after each value. For example, here is a SEROUT command that multiplies each measurement by 2 and displays us afterwards.

```
SEROUT 14, 84, [128, "x = ", DEC4 (2 * x), " us",
                148, "y = ", DEC4 (2 I y), " us"]
```

While "us" isn't really the same as "µs" because we are using u instead of the Greek character mu, most people take its meaning. You can also make a custom character for mu. This will involve adding a SEROUT command to the beginning of the program that defines a custom character. Then, you will have to display that custom character where "u" is currently displayed.

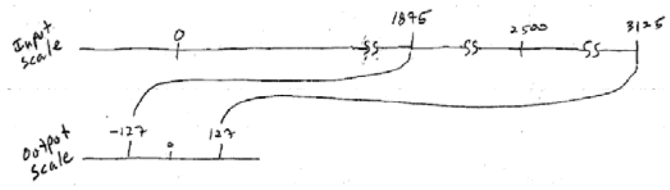
ACTIVITY #3: SCALING DOWN AND OFFSETTING INPUT VALUES

When working with the MX2125 and BASIC Stamp 2, tilt measurements range between 1875 and 3125. This range may have to be scaled and offset any number of ways. For example, Activity #4 scales this to a range of −100 to 100. Activity #5 scales it to −127 and 127.

Introducing an offset into a range of values is easy, and typically involves an addition or subtraction operation. Scaling can be a little trickier, especially with a processor like the BASIC Stamp, which does all its calculations with integer math. This activity introduces the simplest and most accurate way to scale a larger range of values into a smaller range with a PBASIC program. The technique introduced here helps prevent errors from creeping into your sensor measurements with each successive PBASIC calculation, and it will be used and re-used in many of this book's activities.

Scale and Offset Example

In this first example, we'll take an input value that could be anywhere between 1875 and 3125, and scale and offset it to a corresponding output value that falls in a range from −127 to 127. Figure_Next shows how this should work. The position of the value in the output scale should be proportional to the position of the value in the input scale. For example, if the input value is 2500, which is half way between 1875 and 3125, we should expect the output value to be 0, which is half way between −127 and 127.

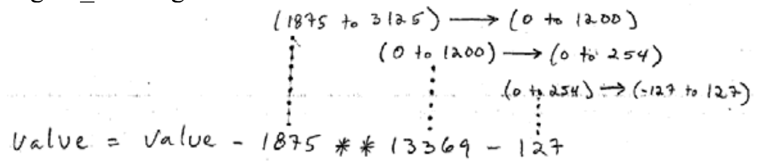


To apply scale and offset in PBASIC, remember these three steps:

1) Apply offset to align the input scale to zero.
2) Apply the scale.
3) Apply any additional offset that is needed for our output scale.

Figure_Next shows how to apply these steps with a single PBASIC command that performs scaling and offset. Keep in mind that PBASIC calculations work from left to right unless they are overridden with parentheses. So the first thing this calculation does is subtract 1875 from the input value. The new range is now 0 to 1200 instead of 1875 to 3215. Next, ** 13369 scales value down to 0 to 254. After the range has been scaled, 127 is subtracted from it resulting in −127 to 127.

Figure_ Scaling the Value Variable


(1875 to 3125) ⟶ (0 to 1200)
(0 to 1200) ⟶ (0 to 254)
(0 to 254) ⟶ (-127 to 127)
Value = Value - 1875 ** 13369 - 127

Choosing the Right ** Constant for Scaling

The value 13369 used with the ** constant to scale (0 to 1250) to (0 to 254) was determined by substituting the number of elements in the input and output scales into this equation. The number of output scale elements is 255, including 0, and the number of input scale elements is 1251, also including 0. Use this equation whenever you need to fit a larger scale into a smaller one with the ** operator.

$$\text{ScaleConstant} = \text{Int}\left[65536\left(\frac{\text{output scale elements}}{\text{input scale elements} - 1}\right)\right]$$

$$\text{Scale Constant} = \text{Int}\left[65535\left(\frac{255}{1251 - 1}\right)\right]$$

$$= \text{Int}\left[13369.344\right]$$

$$= 13369$$

Always round your ScaleConstant result down, even if the result is already an integer! Otherwise, the largest value in your input scale might be one value outside the output scale's range.

Clamping the Input Range

The best way to make sure the output values do not exceed the output range is to make sure the input values do not go outside the input range. For example, if you do not want the output of this command to go outside −127 to 127, the most convenient approach is to make sure that the input values do not go below 1875 or above 3125. Here is a modified version of value = value - 1875 ** 13369 - 127 that prevents the problem.

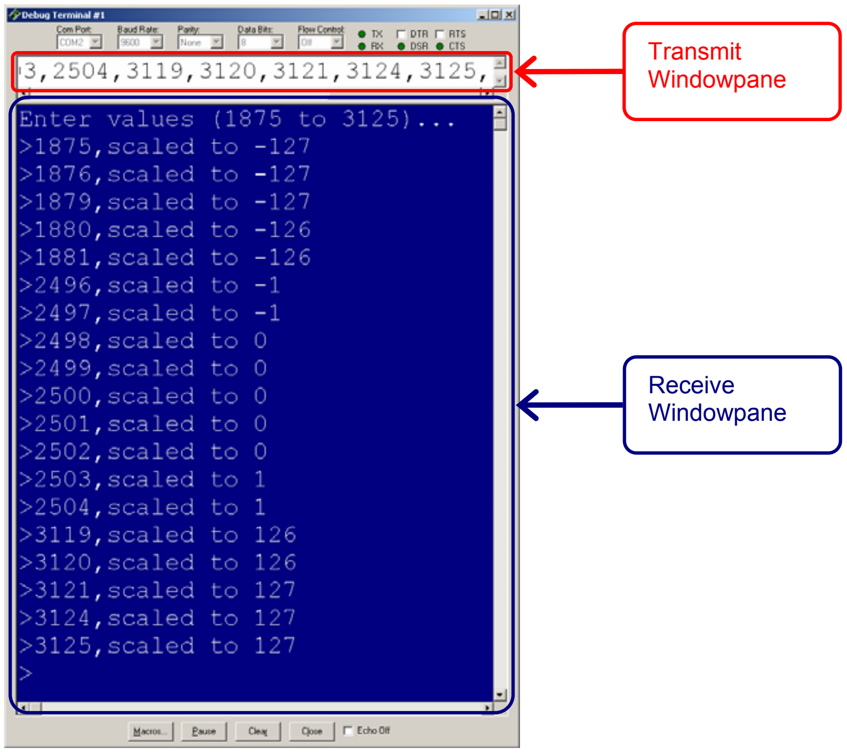
```
        value = (value MIN 1875 MAX 3125) - 1875 ** 13369 - 127
```

Before subtracting 1875 from the value variable, this command uses MIN 1875 MAX 3125 to make sure value stores something in this range. If the value variable is storing a number in this range, the MIN and MAX operators leave it alone. However, if it's storing something less than 1875, MIN 1875 will change value to 1875. Likewise, if it's storing something above 3125, MAX 3125 changes it to 3125.

Example Program: TestScaleOffset.bs2

Figure 3-7 shows what the Debug Terminal looks like as the next example program is tested. When you enter input values (separated by commas) into the Debug Terminal's transmit windowpane, the program displays the scaled and offset equivalent in the Debug Terminal's receive windowpane.

Figure 3-7 Scaling Test



```
3,2504,3119,3120,3121,3124,3125,
```

```
Enter values (1875 to 3125)...
>1875,scaled to -127
>1876,scaled to -127
>1879,scaled to -127
>1880,scaled to -126
>1881,scaled to -126
>2496,scaled to -1
>2497,scaled to -1
>2498,scaled to 0
>2499,scaled to 0
>2500,scaled to 0
>2501,scaled to 0
>2502,scaled to 0
>2503,scaled to 1
>2504,scaled to 1
>3119,scaled to 126
>3120,scaled to 126
>3121,scaled to 127
>3124,scaled to 127
>3125,scaled to 127
>
```

√ Enter, save, and run TestScaleOffset.bs2

√ Enter this sequence into the Debug Terminal's transmit windowpane: 1875,1876,1879,1880,1881,2496,2497,2498,2499,2500,2501,2502,2503,2504,3119,3120,3121,3124,3125.

√ Test various other values that range from 1875 to 3125, and verify with a calculator that the output value 's position in the output range is proportional to the input values position in the input range.

```
' TestScaleOffset.bs2
' Test scaling from an input range of 1875 to 3125 to an output
' range of -127 to + 127.

'{$STAMP BS2}
'{$PBASIC 2.5}
```

```
value          VAR     Word

DEBUG CLS, "Enter values (1875 to 3125)...", CR

DO

  DEBUG ">"

  DEBUGIN DEC value

  value = (value MIN 1875 MAX 3125) - 1875 ** 13369 - 127

  DEBUG "scaled to ", SDEC value, CR

LOOP
```

Your Turn - A Closer Look at the ScaleConstant and ** Operator

For small input and output ranges, we can examine them with a calculator, pencil and paper. Let's take 0 to 10 as our input scale, and 0 to 2 as our output scale. The first step is to figure out what the constant for the ** operation should be using the ** scale constant equation.

$$\text{ScaleConstant} = \text{Int}\left[65536\left(\frac{\text{output scale elements}}{\text{input scale elements} - 1}\right)\right]$$ <<<scale constant>>>

There are three elements in the output scale, 0, 1, and 2. There are 11 elements in the input scale, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. Remembering to round down to the nearest integer, the result is 19660 which is the constant to use with the ** operator.

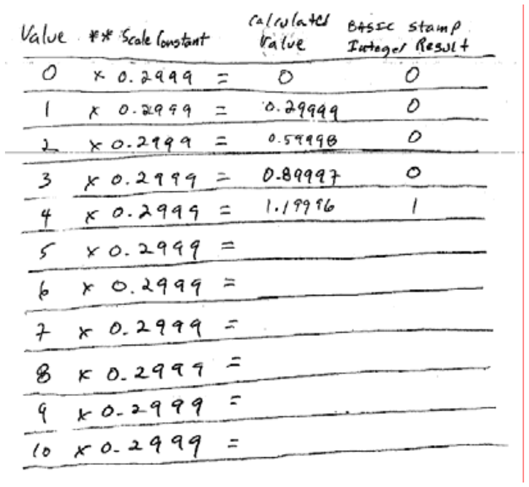
$$\begin{aligned}\text{ScaleConstant} &= \text{Int}\left[65536\left(\frac{3}{11-1}\right)\right]\\ &= \text{Int}\left[19660.8\right]\\ &= 19660\end{aligned}$$

The term value = value ** 19660 multiplies the value variable by

19660 ÷ 65536 ≈ 0.29999 ---> value = value × 0.29999

Table_Next shows some examples of the BASIC Stamp's calculations for each of the values in the input range for value = value ** 19660. Keep in mind that it's about the same as multiplying value by 0.29999 with a calculator. Since the BASIC Stamp is an integer math processor, it truncates any result to an integer value, effectively rounding down. Notice how the first four input values result in outputs of zero. Then, when the input value is 4, the result is 1.19996, which gets rounded to 1. As you perform the rest of the calculations in the table, notice how the output scale of 2 receives four input elements. If −1 was not used in the denominator, it would only receive one input element. <<<This needs to be double-checked>>>

√ Finish the calculations in Table_Next for input values from 5 to 10.



Value	** Scale Constant		Calculated Value	BASIC Stamp Integer Result
0	x 0.2999	=	0	0
1	x 0.2999	=	0.29999	0
2	x 0.2999	=	0.59998	0
3	x 0.2999	=	0.89997	0
4	x 0.2999	=	1.19996	1
5	x 0.2999	=		
6	x 0.2999	=		
7	x 0.2999	=		
8	x 0.2999	=		
9	x 0.2999	=		
10	x 0.2999	=		

√ Save TestScaleOffste.bs2 as TestScaleOffsetYourTurn.bs2.
√ Modify the program so that you can test Table_Previous with the BASIC Stamp and Debug Terminal.
√ Compare the Debug Terminal results to your table.

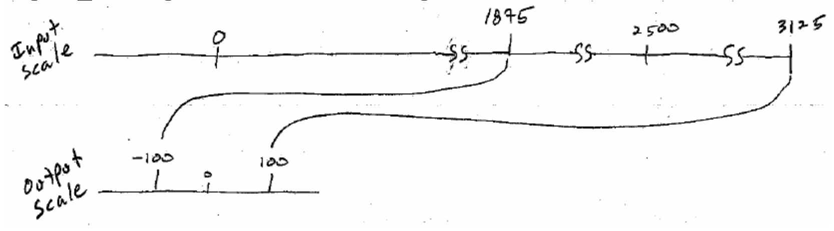
ACTIVITY #4: SCALING TO 1/100 G

The standard measure of gravity on the earth's surface is abbreviated g. This activity demonstrates how to use the techniques introduced in the previous activity to display the number of hundredths of a g acting on the accelerometer's x and y axes.

From PULSIN to 1/100 g

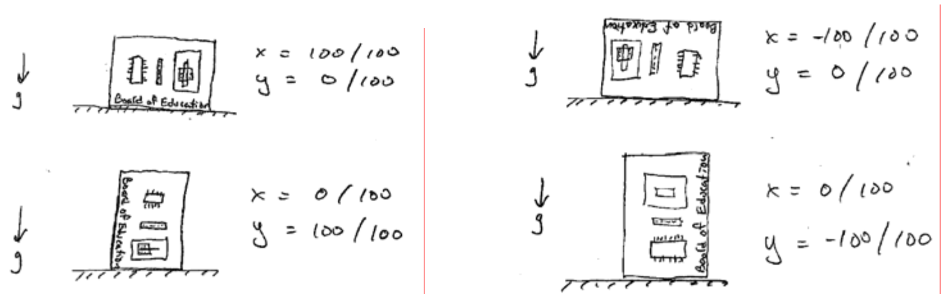
The goal here is to modify the example program from Activity #1 so that it displays the x and y axis measurements in terms of 1/100 g instead of 2 µs units. It's another scaling and offset problem, but this time, we want to fit the 1875 to 3125 input scale into an output scale of −100 to 100 as shown in Figure_Next.

Figure_Scaling and Offset for 1/100 g.



Your Turn - Developing the Program

The goal here is to use the scaling techniques from Activity #3 to modify the program from Activity #1 so that it displays the x and y-axis measurements in terms of 1/100 g. Figure_Next shows the approximate readings you should expect after your modifications.

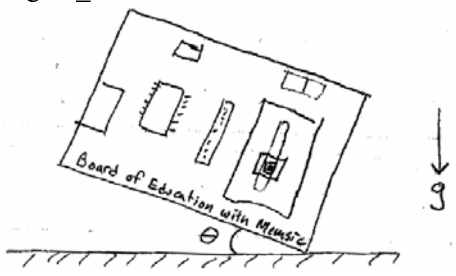


- √ Open SimpleTilt.bs2 from Activity #1 and save it as CentigravityTilt.bs2
- √ Follow the steps for scaling from Activity #3 and determine the ** scale constants.
- √ Add lines of code to the program that scale the x and y values down to g/100.
- √ Modify the display so that it shows in the Debug Terminal.
- √ Test according to Figure_Previous and trouble-shoot if necessary.

ACTIVITY #5: MEASURING 360° VERTICAL ROTATION

The MX2125 has a built-in feature that allows you to use both the x and y axis tilt measurements to calculate the accelerometer's angle of rotation in the vertical plane, as shown in Figure_Next. There are lots of applications where vertical tilt is useful, including virtual steering wheels for video games and counting bicycle wheel revolutions. This activity demonstrates how to calculate tilt on the vertical plane with the PBASIC ATN operator.

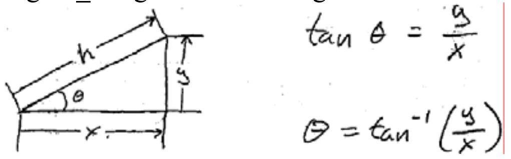
Figure_BOE Tilt on the Vertical Plane



Calculating Arctangent with PBASIC

The tangent of an angle theta (θ) in a right triangle is the ratio of the opposite side of a right triangle (y) divided by the adjacent side (x). If you know the values of x and y, you can use the inverse tangent or arctangent to figure out the angle θ. The most common notations for arctangent are $\tan^{-1}$ and arctan.

Figure_Tangent and Arctangent

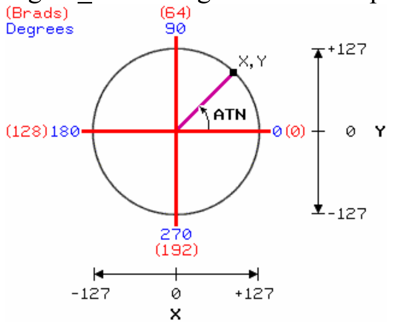


The arctangent function can be used to determine the accelerometer's rotation angle with its x and y measurements. PBASIC has an operator called ATN you can use for calculating $\tan^{-1}(y/x)$. To calculate the arctangent of y/x and store it in a variable named angle, use the command angle = x ATN y.

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \longrightarrow \text{angle} = x \text{ atn } y$$

Figure_Next is from the BASIC Stamp Editor's Help file, and it shows ATN works. Both the x and y variables have to be scaled to values between −127 and 127. The result of the ATN operator is the angle in binary radians, which is abbreviated brads. With brads, a circle is split up into 256 segments in the same way that degrees splits a circle into 360 segments.

Figure_ATN Diagram From Help File



Converting from Brads to Degrees with */

In the previous activity, we used the ** operator to scale values down from a larger range to a smaller range. Converting from brads to degrees involves scaling a smaller scale of 0 to 255 to a larger scale of 0 to 359. The PBASIC */ operator is designed for this job.

When you use a command like value = ScaleConstant */ value, the ScaleConstant term is the number of 256ths you want to multiply the value variable by. For example, let's say you want to multiply value by 2.5. Multiply 2.5 by 256 and the result is 640. Now, if value starts as 10, the result of value = 640 */ value will be 25.

Want: Value = 2.5 x value

ScaleConstant = 2.5 x 256
= 640

Value = 640 */ value 'Multiply by 2.5

Remember

The ** operator multiplies by a number of 65536ths.

The */ operator multiplies by a number of 256ths.

The rules of integer math for scaling from one scale to another still apply, even though we are converting from a smaller scale to a larger one. The only thing that will change is the scale constant, which is a numerator of 256 for */, instead of 65536 for **.

$$\text{*/ scale constant} = \text{Int}\left[256\left(\frac{\text{output scale elements}}{\text{input scale elements} - 1}\right)\right]$$

The input scale is 0 to 255, which has 256 elements, and the output is 0 to 359, which has 360 elements. The result after substituting these values into the */ scale constant equation is 361.

$$\begin{aligned}\text{*/ scale constant} &= \text{Int}\left[256 \times \left(\frac{360}{256-1}\right)\right]\\ &= \text{Int}\left[256 \times \left(\frac{360}{255}\right)\right]\\ &\approx \text{Int}\left[361.412\right]\\ &= 361\end{aligned}$$

This demonstrates that, if the angle variable stores a measure of brads, and you want to store a measure of degrees instead, use this command.

```
        angle = 361 */ angle
```

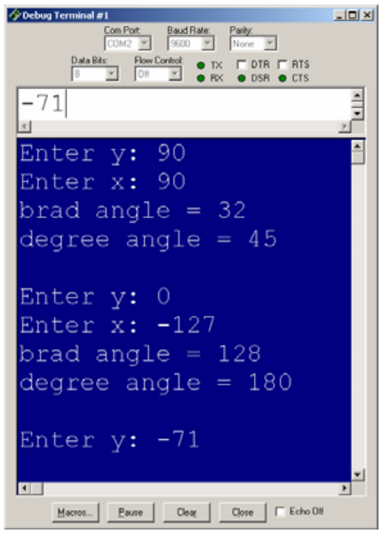
Most documents recommend angle = 360 */ angle. However, using a */ scale constant of 361 is slightly more accurate over the input/output ranges. Try comparing the BASIC Stamp's results to this equation for a calculator or spreadsheet.

$\text{angle}_{\text{degrees}} = (360/256) \times \text{angle}_{\text{brads}}$

Round the result of $\text{angle}_{\text{degrees}}$ to the nearest integer. If the result has a fractional component of 0.5 or higher, round up. Otherwise, round down. Then compare it to the 256 possible Debug Terminal outputs with 360 */ angle, then repeat with 361 */ angle. A spreadsheet is useful for this comparison. If you try it, you'll se that the rate of integer value matches is much higher with 361 */ angle.

Example Program: TestAtn.bs2

This example program calculates angles based on the y and x values you enter into the Debug Terminal's transmit windowpane.



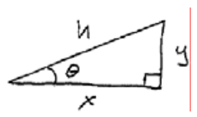
```
-71
```

```
Enter y: 90
Enter x: 90
brad angle = 32
degree angle = 45

Enter y: 0
Enter x: -127
brad angle = 128
degree angle = 180

Enter y: -71
```

To calculate x and y values to enter into the Debug Terminal, use these equations.

$$y = h \sin \theta$$

$$x = h \cos \theta$$



For example, let's say that h = 127 and θ = 45°, then the x and y values to be entered into the Debug Terminal are both 90. If h = 100 and θ = 315°, the y value to enter into the Debug Terminal will be −71, and the x value will be 71. If h = 100 and θ = 180°, y will be 0 and x will be −127.

$h = 127$ and $\theta = 45°$

$h = 100$ and $\theta = 315°$

$y = 100 \sin 315°$
$= -71$

$x = 100 \cos 315°$
$= 71$

$h = 127$ and $\theta = 180$

$y = 127 \sin 180$
$= 0$

$x = 127 \cos 180$
$= -127$

$$
\begin{aligned}
y &= 127 \sin 45^\circ \\
&= 89.8 \\
&\approx 90
\end{aligned}
$$

$$
\begin{aligned}
x &= 127 \cos 45^\circ \\
&= 89.8 \\
&\approx 90
\end{aligned}
$$

√ Enter, save, and run TestAtn.bs2
√ Click the Debug Terminal's transmit windowpane. When prompted for the x value, type 90 and press the carriage return. When prompted for y, type 90 and the carriage return again.
√ Verify that the result is 32 brads = 45°.
√ Repeat for the other x and y values just discussed.
√ Use your calculator to determine the x and y values that correspond with various h and angle values. Compare your calculated results to the Debug Terminal's results.

Some values will be lower than you predict. For example, when h = 100 and θ = 30°, y = 50 and x = 87. The Debug Terminal will display 21 for the brad angle, which is correct, but 29 for the degree angle is not correct. It should be 30. This happens occasionally when scaling from a smaller range to a larger range. The 21 brads measurement corresponds to 29° and 22 brads corresponds to 31°.

```
' TestAtn.bs2
' Test BASIC Stamp arctangent calculations.

'{$STAMP BS2}
'{$PBASIC 2.5}

angle          VAR     Word
x              VAR     Word
y              VAR     Word

DO

  DEBUG "Enter y: "
  DEBUGIN SDEC y
  DEBUG "Enter x: "
  DEBUGIN SDEC x

  angle = x ATN y
  DEBUG "brad ", SDEC ? angle

  angle = angle */ 361

  DEBUG "degree ", SDEC ? angle, CR

LOOP
```

Your Turn - Testing Brad to Degree Conversion

As mentioned earlier, the ideal integer result comes from calculating $\text{angle}_{\text{degrees}}$ = (360/256) × $\text{angle}_{\text{brads}}$, and then rounding up if the value to the right of the decimal point is 5 to 9 or down if it is 1 to 4. You can generate a list of all 256 brad to degree conversions with this program.

```
' BradsToDegrees.bs2
' Display brad to degree conversions for */ 360 and */ 361.

'{$STAMP BS2}
'{$PBASIC 2.5}

angle          VAR     Word
brads          VAR     Word

DEBUG CLS, "brads  */ 360   */ 361", CR

FOR brads = 0 TO 255

  DEBUG DEC3 brads

  angle = brads */ 360
  DEBUG "      ", DEC3 angle

  angle = brads */ 361
  DEBUG "      ", DEC3 angle, CR

NEXT

END
```

√ Enter, save and run BradsToDegrees.bs2.
√ Use a spreadsheet or calculator to generate the a list with this forumal.

$\text{angle}_{\text{degrees}}$ = (360/256) × $\text{angle}_{\text{brads}}$ formula.

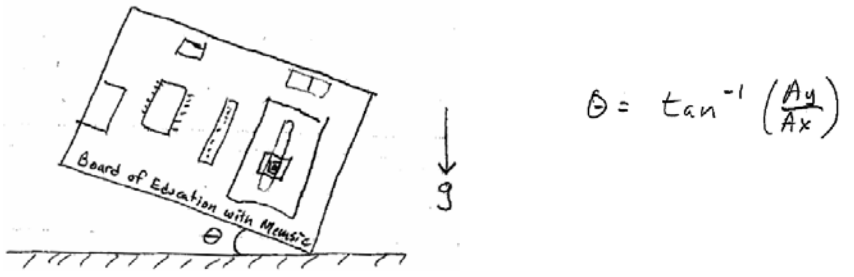
Remember to round up if the value to the right of the decimal point is 5 to 9 or down if it's 1 to 4.

√ Compare your results to the Debug Terminal display. How many exact matches occurred for */ 360? How many occurred for */ 361?

Measuring Tilt Angle On the Vertical Plane

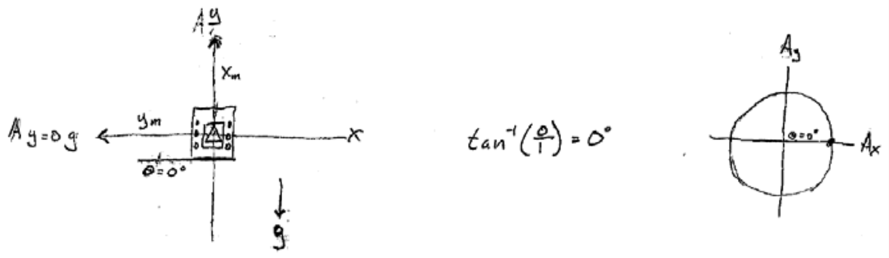
The angle of your board's clockwise rotation in the vertical plane (θ) shown in Figure_Next is the arctangent of the gravity's effect on the MX2125's y-axis (Ay) divided by its effect on its x-axis (Ax).

Figure_Equation for Clockwise Vertical Rotation


Board of Education with Memsic
θ
g
$\theta = \tan^{-1}\left(\frac{A_y}{A_x}\right)$

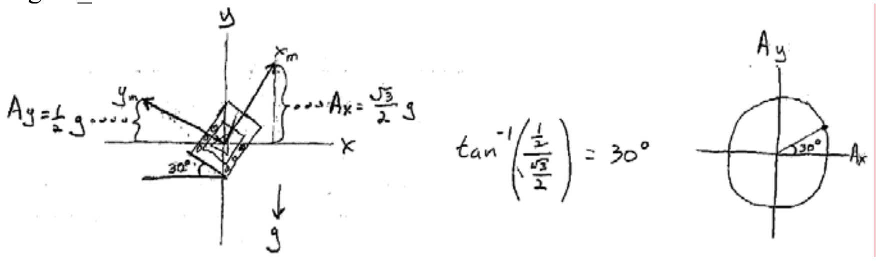
Here are a few examples of what the accelerometer detects and how it relates to the arctangent of the ratio of Ay to Ax. Figure_Next shows what the accelerometer senses at the 0° mark. If θ is 0°, then Ay senses 0-gravity (g), and Ax senses 1 g, the arctangent of 0/1 is 0°.

Figure_Accelerometer Rotated 0°



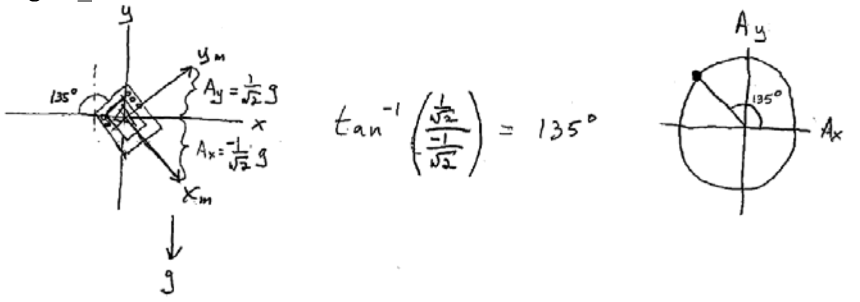
When the accelerometer is rotated 30° clockwise, shown in Figure_Next, the component of gravity acting on the accelerometer's x-axis is approximately √3/2 g. The component of gravity acting on the y-axis is 1/2 g, and the arctangent of √3/2 ÷ 1/2 is 30°.

Figure_Accelerometer Rotated 30 Clockwise°


y
x_m
y_m
$A_y = \frac{1}{2} g$
$A_x = \frac{\sqrt{3}}{2} g$
x
30°
g
$\tan^{-1}\left(\frac{\frac{1}{2}}{\frac{\sqrt{3}}{2}}\right) = 30°$
A_y
30°
A_x

When the accelerometer is rotated to 135° clockwise, the component of gravity acting on the accelerometer's x-axis is Ax = -1/√2, and the component acting on its y-axis is 1/√2. The arctangent of 1/√2 ÷ (−1/√2) is 135°.

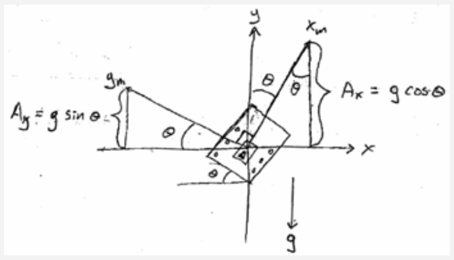
Figure_Accelerometer Rotated 135° Clockwise


y
y_m
135°
$A_y = \frac{1}{\sqrt{2}} g$
x
$A_x = \frac{-1}{\sqrt{2}} g$
x_m
g
$\tan^{-1}\left(\frac{\frac{1}{\sqrt{2}}}{\frac{-1}{\sqrt{2}}}\right) = 135°$
A_y
135°
A_x

The General Case

The angle of rotation (θ) is the inverse tangent or arctangent of the acceleration acting on the y-axis (Ay) divided by the acceleration acting on the x-axis (Ax). Figure_Next shows the MX2125 tilted at an angle θ, which rotates both sensing axes by θ. By applying a couple of geometry identities, θ is also inside the two triangles that show the components of gravity acting on each of the accelerometers sensing axes (x_m and y_m). The component of gravity acting on xm is Ax = g cosθ, and the component acting on ym is Ay = g sinθ. After applying the trig identities shown on the right, it demonstrates that the angle of rotation θ is in fact the arctangent of Ay/Ax.

Figure_Gravity Acting on the Accelerometer Axes



$$\frac{A_y}{A_x} = \frac{g\sin\theta}{g\cos\theta} = \tan\theta$$

$$\tan\theta = \frac{A_y}{A_x}$$

$$\tan^{-1}(\tan\theta) = \tan^{-1}\left(\frac{A_y}{A_x}\right)$$

$$\theta = \tan^{-1}\left(\frac{A_y}{A_x}\right)$$

Example Program: VertWheelRotation.bs2

This program displays your board's angle of rotation as shown in (Figure_BOE Tilt on the Vertical Plane) at the beginning of this activity.

- √ Enter, save, and run VertWheelRotation.bs2.
- √ Hold the board in front of you like a steering wheel. As you rotate the board clockwise, watch the angle measurement grow. Verify that the display angle ranges from 0 to 359.

```
' VertWheelRotation.bs2
' Mount accelerometer on a vertical wheel and measure
' the rotation angle.

'{$STAMP BS2}
'{$PBASIC 2.5}

angle          VAR     Word
x              VAR     Word
y              VAR     Word

DO
```

```
  PULSIN 6, 1, x
  PULSIN 7, 1, y

  x = (x MIN 1875 MAX 3125) - 1875 ** 13369 - 127
  y = (y MIN 1875 MAX 3125) - 1875 ** 13369 - 127

  angle = x ATN y
  angle = angle */ 361

  DEBUG HOME, CLREOL, SDEC ? x,
              CLREOL, SDEC ? y,
              "angle = ", CLREOL,
              DEC angle,
              176                           ' ASCII 176 is degree symbol

  PAUSE 100

LOOP
```

Your Turn - Debug Terminal Behavior

This DEBUG command below displays signed values of x and y followed by the angle and the degree symbol, which is the ASCII code 176. The reason CLREOL comes before each number is to prevent characters that don't disappear on the right of some measurements. For example, if one measurement is −105, and the next measurement is 076, it will display as 0755 if the CLREOL doesn't clear the previous value before displaying the new one. Although CLS can fix this problem too, the Debug Terminal flicker that results is not pleasant to examine for any length of time. CLREOL erases to the right of the cursor on a given line. While it still causes a little bit of flicker in each value, you'll likely agree that it's much easier to look at than the CLS version.

```
DEBUG HOME, CLREOL, SDEC ? x,
            CLREOL, SDEC ? y,
            "angle = ", DEC3 angle,
            176                        ' ASCII 176 is degree symbol
```

√ Save VertWheelRotation.bs2 as VertWheelDisplayTest.bs2.
√ Replace HOME with CLS in the debug command and run the program.
√ Change CLS back to home and run the program again. Do you see an improvement in the display?
√ Remove the CLREOL control characters and note the effect on the display as you rotate the board. Extra digits will appear at the end of non-negative values.

√ Put the CLREOL control characters back in and run the program again. Those pesky extra digits that don't disappear should be gone.

Your Turn - LCD Display

The Debug Command has three lines of display, and the degree symbol will need a custom character. Here is an initialization command for the LCD that does three things: (1) start the LCD, (2) display text that doesn't change, and (3) defines custom character 7 as the degree symbol °.

```
' Initialize LCD
PAUSE 200
SEROUT 14, 84, [22, 12]
PAUSE 5

SEROUT 14, 84, [ 130, "angle = ", DEC angle, 7,
                 150, "x=", SDEC x,
                 157, "y=", SDEC y ]

SEROUT 14, 84, [255,                  ' Define custom character 7
                %01000,               '   *
                %10100,               ' *   *
                %01000,               '   *
                %00000,               '
                %00000,               '
                %00000,               '
                %00000,               '
                %00000]               '
```

Custom Character Definitions

Remember, 248 defines custom character 0. 249 defines custom character 1. 250 defines custom character 2, and so on, up to 255, which defines custom character 7.

The DEBUG command that the SEROUT command has to replace uses three lines in the Debug Terminal. The SEROUT command below only uses two. To minimize LCD display flicker, only the digits are erased before the new digits are printed. The SEROUT places the cursor at 138 (line-0, character-10), then overprints the previous measurement with five spaces. Then, it places the cursor at 138 again and displays the new degree measurement with DEC angle. Then, it prints the degree sign with custom character 7. This is repeated for the x and y measurements, but there only need to be four spaces between quotes following cursor positions 152 and 159.

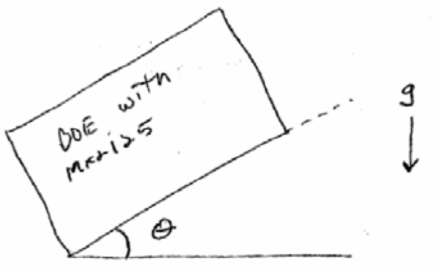
```
' LCD Display Routine
SEROUT 14, 84, [ 138, "     ", 138, DEC angle, 7,
                    152, "    ", 152, SDEC x,
                    159, "    ", 159, SDEC y ]
```

- √ Save VertWheelRotation.bs2 as VertWheelRotationLcd.bs2.
- √ Insert the initialize LCD routine between the variable declarations and the DO keyword.
- √ Replace the DEBUG command in the DO...LOOP with the LCD Display Routine.
- √ Change PAUSE 100 to PAUSE 350.
- √ Run the program and test your LCD rotation display.

Your Turn - Rotation in the Opposite Direction

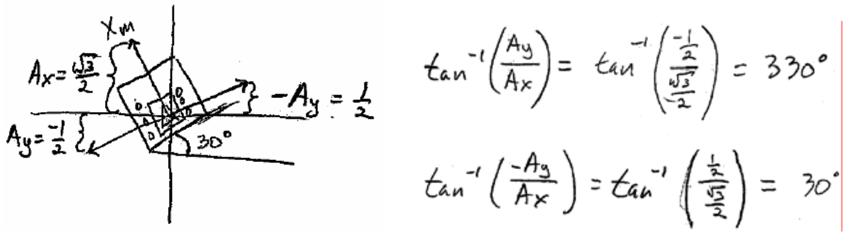
Diagrams that show the rotation angle increasing as the object rotates counterclockwise, like Figure_Next are quite a bit far more common.

Figure_Angle Measurement Increases with Counterclockwise Rotation



To reverse the angle of rotation the program displays, all you have to do is use −Ay istead of Ay. Take a look at Figure_Next. If you rotate the accelerometer counterclockwise, Ay is −1/2, and the arctangent turns out to be 330°. By taking the arctangent of −Ay/Ax, the result is 30°.

Figure_Reversing Direction of Rotation with −Ay


Xm
Ax = √3/2
−Ay = 1/2
Ay = −1/2
30°
tan⁻¹(Ay/Ax) = tan⁻¹((−1/2)/(√3/2)) = 330°
tan⁻¹(−Ay/Ax) = tan⁻¹((1/2)/(√3/2)) = 30°

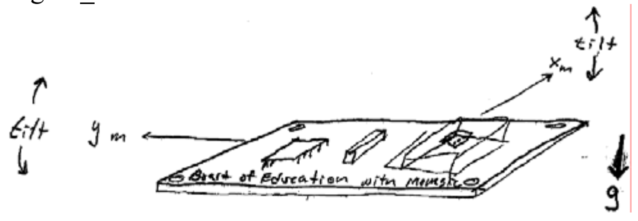
This change is easy to make in the program. Simply insert a negative sign before the y in angle = x ATN y.

- √ Save VertWheelRotation.bs2 as VertWheelRotationCounterclockwise.bs2
- √ Change angle = x ATN y to angle = x ATN −y.
- √ Run the program and verify that the rotation angle now increases as you rotate the board counterclockwise.

ACTIVITY #6: MEASURE TILT FROM THE HORIZONTAL

This activity measures how far the Board of Education is tilted from the horizontal. Figure_Next shows the Board of Education with the Memsic Accelerometer on the breadboard. The accelerometer's acceleration sensing axes (x_m and y_m) point toward the top and left of the Board of Education. This activity develops a program that displays the tilt angle for each axis. When the board is held level, the tilt angle is 0° for both the x_m an y_m axes. If you tilt the board so that y_m points up, the program will report a positive tilt angle for the y-axis. If you tilt it so that y_m points down, it will report a negative tilt angle. The same applies for x_m; point it up for a positive tilt angle or down for a negative tilt angle. If you tilt the board towards one of its corners, the program will report tilt for both the x_m and y_m axes.

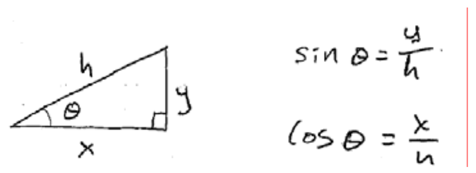
Figure_Accelerometer Tilt Axes on Board of Education



Sine and Cosine

Figure_Next shows the relationship between the sides of a right triangle and the sine, and cosine functions. The sine of an angle is the opposite side of the triangle (y) multiplied by the hypotenuse (h). If you know h and y, and want to know the angle (θ), use arcsine ($\sin^{-1}$). The cosine of the angle is the adjacent side (x) divided by h. If you want to know the angle given x and h, use arccosine ($\cos^{-1}$).

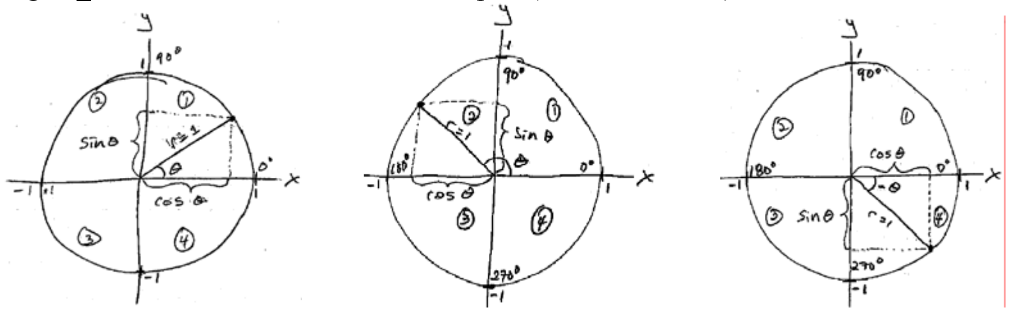
Figure_Sine, Cosine, Arcsine and Arccosine



Note from the equations for Figure_Previous that the x value can be at most the same as h when θ = 0°. Likewise, the y value can be at most 1 when θ = 90°. For angles between 0 and 90°, the ratio of x/h and y/h are both less than 1. It doesn't matter how large the triangle is, the ratio will always be between 1 and 0.

The unit circle is a common device for describing the sine and cosine functions. The triangle's hypotenuse becomes the radius of the circle. The unit circle is so named because the length of the hypotenuse is 1 (one unit). As the hypotenuse is rotated clockwise, the angle θ becomes larger, or smaller if it is rotated counterclockwise. The cosine is determined by drawing a vertical line from the point where the hypotenuse meets the circle down (or up if the hypotenuse is below) to the x-axis. Whatever the x value is, that's the cosine. The sine of the angle is determined by drawing a line from the end of the radius, horizontally to the y-axis.

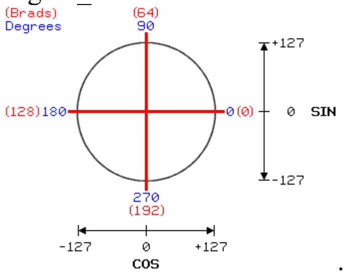
Figure_Unit Circle Sine and Cosine Examples (Label a, b, and c)



The range from 0 to 90° is the unit circle's quadrant 1. When θ is in quadrant 1, both the cosine and sine of the angle will be positive numbers. When θ is between 90 and 180° (quadrant 2), the cosine becomes negative but the sine is still positive. In quadrant 3, both sine and cosine are negative, and in quadrant 4, the sine is still negative but cosine is positive again. Notice in Figure_Previous (c) that a negative value of θ (between 0 and −90) can be in quadrant 4 just as a value between 270 and 360°. One other thing to keep in mind here is that the minimum value for both sine and cosine is −1, and the maximum value is 1. For example, when θ = 0°, cos θ = 1, and sin θ = 0. If θ = 90°, sin θ = 1 and cos θ = 0. At θ = 180°, cos θ = −1 and sin θ = 0.

Figure_Next shows the BASIC Stamp's version of a unit circle for its SIN and COS operators. Instead of results that range from −1 to 1, the results for SIN and COS range from −127 to 127. Angles for the SIN and COS operators are in terms of brads. So, instead of 45°, use 32 brads. Instead of 90°, use 64 brads, and so on. To convert from brads to degrees with a calculator, number of brads by 360/256. To convert from degrees to brads, use 256/360.

Figure_Unit Circle for the PBASIC SIN and COS Operators



i-box: The next example program converts from degrees to brads with ** 46733, which is derived using the procedure in Activity #3.

Example Program: SineCosine.bs2

This example program displays the BASIC Stamp's integer calculations for sine and cosine. You can divide these values by 127 to get an approximation of the actual sine or cosine values.

- √ Enter, save and run SineCosine.bs2
- √ Compare the results (divided by 127) to calculated values of sine and cosine.

```
' Smart Sensors and Applications - SineCosine.bs2
' Display BASIC Stamp sine and cosine values.

' {$STAMP BS2}
' {$PBASIC 2.5}

degrees VAR Word
brads VAR Word
sine  VAR Word
```

```
cosine VAR Word

DEBUG "Degrees  Brads  Cosine  Sine", CR

FOR degrees = 0 TO 359 STEP 15

  brads = degrees ** 46733
  sine = SIN brads
  cosine = COS brads
  DEBUG "  ",
        SDEC3 degrees, "     ",
        SDEC3 brads, "    ",
        SDEC3 cosine, "    ",
        SDEC3 sine, CR

NEXT

END
```

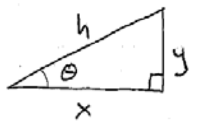
Your Turn - Program Modifications

- √ Try modifying the FOR...NEXT loop's STEP argument to get different values.
- √ Try modifying the program so that it prompts you for a degree value with the DEBUGIN command and then displays the result.

Arcsine and Arccosine Subroutines

While the sine is a ratio of y/h for a given angle, arcsine ($\sin^{-1}$) is the reverse. (See Figure_Next.) Given the ratio y/h, arcsine tells you the angle. Likewise, cosine is the ratio of x/h for a given angle, and arccosine ($\cos^{-1}$) is the angle for a given ratio of x/h.

Figure_Sine, Arcsine, Cosine, and Arccosine



$$\sin\theta = \frac{y}{h} \qquad \theta = \sin^{-1}\left(\frac{y}{h}\right)$$

$$\cos\theta = \frac{x}{h} \qquad \theta = \cos^{-1}\left(\frac{x}{h}\right)$$

While the BASIC Stamp does not have ASIN and ACOS operators, Tracy Allen, Author of Applied Sensors, published some very nice subroutines that perform these functions on his web site www.emesystems.com. The next example program uses modified versions of these subroutines.

Remember that the SIN and COS operators return values between −127 and 127. If you divide the result by 127, you'll get a value between −1 and 1 that is an approximation of the actual sine (y/h) or cosine (x/h) ratios. With the Arcsine and Arccosine subroutines, you can set a variable named side to a value between −127 and 127, and the subroutine will store the degree measurement results in the angle variable.

If you want the Arcsine and Arccosine subroutines to return brads instead of degrees, just comment the line of code that converts from brads to degrees in the Arccosine subroutine.

```
'  angle = angle */ 361    ' Convert brads to degrees
```

Example Program: TestArcsine.bs2

This next program sweeps sine values from −127 to 127, and its Arcsine subroutine converts these sine values back to degree angles. Keep in mind that this is the reverse of the calculations in the previous example program. The previous example program displayed sine values for given angles. This one displays angles for given sine values.

√ Enter, save, and run TestArcsine.bs2
√ Compare the results to the sine values calculated in the previous example program.

```
' -----[ Title ]--------------------------------------------------------------
' Smart Sensors and Applications - TestArcsine.bs2
' Test arcsine for sine values from -127 to 127.

' {$STAMP BS2}                               ' BASIC Stamp Directive
' {$PBASIC 2.5}                              ' PBASIC Directive

' -----[ Constants ]----------------------------------------------------------

Negative       CON     1                     ' Sign - .bit15 of Word variables
Positive       CON     0

' -----[ Variables ]----------------------------------------------------------

sine           VAR     Word                  ' sine in circle r = 127
side           VAR     Word                  ' trig subroutine variable
angle          VAR     Word                  ' result angle - degrees
sign           VAR     Bit                   ' Sign bit

' -----[ Initialization ]-----------------------------------------------------
```

```
DEBUG CLS                                    ' Clear Debug Terminal
sine = -128                                  ' Start y at -128

' -----[ Main Routine ]-----------------------------------------------------

DO UNTIL sine = 127                     ' Sweep from y = -127 to y = 127
  sine = sine + 1                       ' Increment by 1
  side = sine                           ' Set side to y
  DEBUG "sine = ", SDEC sine, "  "      ' Display sine value
  GOSUB Arcsine                         ' Calculate arcsine
  DEBUG SDEC ? angle                    ' Display the result angle
LOOP                                    ' Repeat DO...LOOP

END                                     ' End program

' -----[ Subroutine - Arcsine ]---------------------------------------------

' This subroutine calculates arcsine based on the y coordinate on a circle
' of radius 127.  Set the side variable equal to your y coordinate before
' calling this subroutine.

Arcsine:                                     ' Inverse sine subroutine
  GOSUB Arccosine                            ' Get inverse cosine
  angle = 90 - angle                         ' sin(angle) = cos(90 - angle)
  RETURN

' -----[ Subroutine - Arccosine ]-------------------------------------------

' This subroutine calculates arccosine based on the x coordinate on a circle
' of radius 127.  Set the side variable equal to your x coordinate before
' calling this subroutine.

Arccosine:                                   ' Inverse cosine subroutine
  sign = side.BIT15                          ' Save sign of side
  side = ABS(side)                           ' Evaluate positive side
  angle = 63 - (side / 2)                    ' Initial angle approximation
  DO                                         ' Successive approximation loop
    IF (COS angle <= side) THEN EXIT         ' Done when COS angle <= side
    angle = angle + 1                        ' Keep increasing angle
  LOOP
  angle = angle */ 361                       ' Convert brads to degrees
  IF sign = Negative THEN angle = 180 - angle' Adjust if sign is negative.
  RETURN
```

Your Turn - Testing the Arccosine Subroutine

Here are some modifications you can make to TestArcsine.bs2 to make it test the Arccosine subroutine instead.

- √ Save TestArcsine.bs2 as TestArccosine.bs2.
- √ Update the comments in the title section. Cosine values will be swept from 127 to −127.
- √ Change sine VAR Word to cosine VAR Word in the Variables section.
- √ Change sine = −128 to cosine = 128 in the Initialization section
- √ Modify the Main Routine so that it looks like this

```
DO UNTIL cosine = -127
  cosine = cosine - 1
  side = cosine
  DEBUG "cosine = ", SDEC cosine, "  "
  GOSUB Arccosine
  DEBUG SDEC ? angle
LOOP

END
```

- √ Run the modified test program. As cosine sweeps from 127 to -127, the angle should sweep from 0 to 180°.

Displaying Accelerometer Tilt Angle

Figure_Next shows the Board of Education with a Memsic Accelerometer. The figure also shows a close-up of the accelerometer module and its x_m and y_m acceleration sensing axes. These sensing axes detect components of the earth's acceleration due to gravity. As you tilt a given axis toward vertical, larger components of the earth's 1 g acts on the axis.

Figure_Tilting the Board of Education, Tilting the Memsic Accelerometer

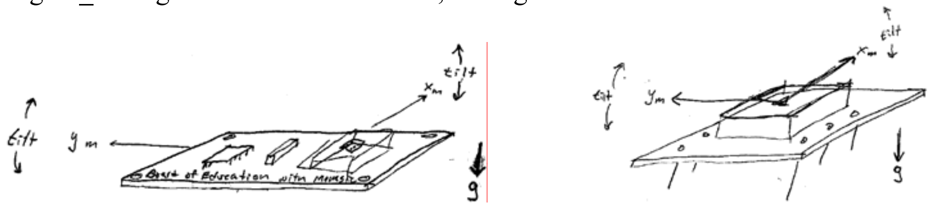
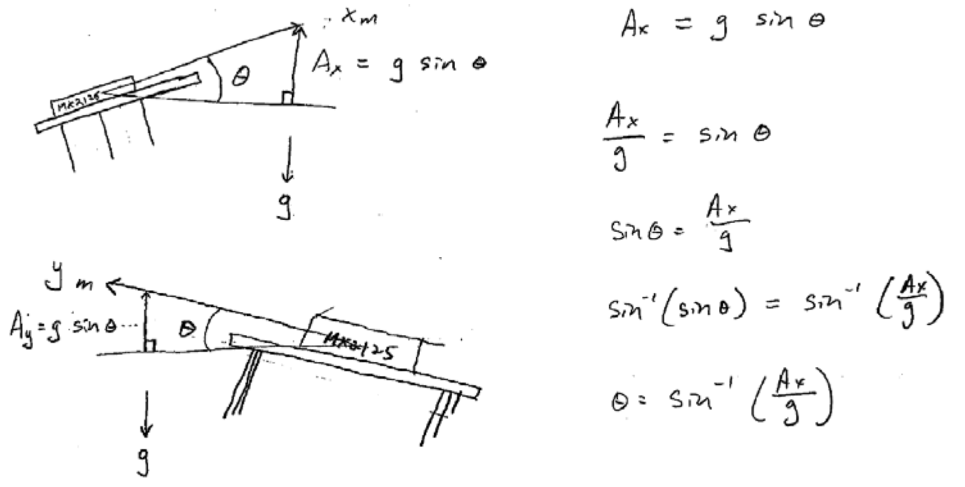


Figure 3-8 shows how arcsine can be used to determine the tilt angle. Looking at the Memsic Accelerometer Module from the side, the component of gravity acting on its x_m is the x-axis acceleration (A_x), which is $g \times \sin \theta$. Since $\sin \theta$ is equal to A_x / g, θ_x can be determined by taking the arcsine of A_x / g. In terms of an equation, that's $\theta_x = \sin^{-1}(A_x/g)$.

The same principle applies to the accelerometer's y_m axis, and the result is $\theta_y = \sin^{-1}(A_y/g)$.

Figure 3-8 Determining Tilt Angle with Arcsine



x_m
$A_x = g \sin\theta$
θ
MX2125
g
y_m
$A_y = g \sin\theta$
MX2125
$A_x = g \sin\theta$
$\frac{A_x}{g} = \sin\theta$
$\sin\theta = \frac{A_x}{g}$
$\sin^{-1}(\sin\theta) = \sin^{-1}\left(\frac{A_x}{g}\right)$
$\theta = \sin^{-1}\left(\frac{A_x}{g}\right)$

With the MX2125, a measurement of 1875 is −1 g, and a measurement of 3125 is 1 g. In Activity #3, we scaled this to a range of −127 to 127. Remember that −127 is the equivalent of −1 for the Arcsine subroutine, and 127 is the equivalent of 1. Anything between −127 and 127 is the equivalent of a fraction, and coming from the MX2125, it's actually sin θ. So, once the MX2125's measurement has been scaled to −127 to 127, all you have to do is use the Arcsine subroutine to determine the tilt angle (the value of θ).

The simplest way to write a tilt program is to start with the previous example program, TestArcsine.bs2. Then, incorporate the accelerometer measurement and scaling and offset commands from TestScaleOffset.bs2 and the accelerometer measurements from VertWheelRotation.bs2. This program's main routine boils down to two commands for measuring the x and y axes, two commands for scaling, and two small routines that call the Arcsine routine and display the result.

```
DO

  PULSIN 6, 1, x
  PULSIN 7, 1, y
```

```
    x = (x MIN 1875 MAX 3125) - 1875 ** 13369 - 127
    y = (y MIN 1875 MAX 3125) - 1875 ** 13369 - 127

    side = x
    GOSUB Arcsine
    DEBUG HOME, "x tilt angle = ", CLREOL, SDEC3 angle, CR

    side = y
    GOSUB Arcsine
    DEBUG "y tilt angle = ", CLREOL, SDEC3 angle

    PAUSE 100

  LOOP
```

Example Program: HorizontalTilt.bs2

This example program displays your board's tilt in terms of degrees from horizontal.

√ Enter, save and run HorizontalTilt.bs2.
√ Compare various tilt angles to the Debug Terminal's axis display.

```
' -----[ Title ]-----------------------------------------------------------
' Smart Sensors and Applications - HorizontalTilt.bs2
' Test arcsine for sine values from -127 to 127.

' {$STAMP BS2}                             ' BASIC Stamp Directive
' {$PBASIC 2.5}                            ' PBASIC Directive

' -----[ Constants ]-------------------------------------------------------

Negative       CON     1                   ' Sign - .bit15 of Word variables
Positive       CON     0

' -----[ Variables ]-------------------------------------------------------

x              VAR     Word                ' Memsic x-axis measurement
y              VAR     Word                ' Memsic y-axis measurement

side           VAR     Word                ' trig subroutine variable
angle          VAR     Word                ' result angle - degrees
sign           VAR     Bit                 ' Sign bit

' -----[ Initialization ]--------------------------------------------------

DEBUG CLS                                  ' Clear Debug Terminal

' -----[ Main Routine ]----------------------------------------------------
```

```
DO

  PULSIN 6, 1, x                        ' x-axis measurement
  PULSIN 7, 1, y                        ' y-axis measurement

  ' Scale and offset x and y-axis values to -127 to 127.
  x = (x MIN 1875 MAX 3125) - 1875 ** 13369 - 127
  y = (y MIN 1875 MAX 3125) - 1875 ** 13369 - 127

  ' Calculate and display Arcsine of x-axis measurement.
  side = x
  GOSUB Arcsine
  DEBUG HOME, "x tilt angle = ", CLREOL, SDEC3 angle, CR

  ' Calculate and display Arcsine of y-axis measurement.
  side = y
  GOSUB Arcsine
  DEBUG "y tilt angle = ", CLREOL, SDEC3 angle

  PAUSE 100                             ' Pause 1/10 second

LOOP                                    ' Repeat DO...LOOP

' -----[ Subroutine - Arcsine ]---------------------------------------------

' This subroutine calculates arcsine based on the y coordinate on a circle
' of radius 127.  Set the side variable equal to your y coordinate before
' calling this subroutine.

Arcsine:                                     ' Inverse sine subroutine
  GOSUB Arccosine                            ' Get inverse cosine
  angle = 90 - angle                         ' sin(angle) = cos(90 - angle)
  RETURN

' -----[ Subroutine - Arccosine ]-------------------------------------------

' This subroutine calculates arccosine based on the x coordinate on a circle
' of radius 127.  Set the side variable equal to your x coordinate before
' calling this subroutine.

Arccosine:                                   ' Inverse cosine subroutine
  sign = side.BIT15                          ' Save sign of side
  side = ABS(side)                           ' Evaluate positive side
  angle = 63 - (side / 2)                    ' Initial angle approximation
  DO                                         ' Successive approximation loop
    IF (COS angle <= side) THEN EXIT         ' Done when COS angle <= side
    angle = angle + 1                        ' Keep increasing angle
  LOOP
  angle = angle */ 361                       ' Convert brads to degrees
  IF sign = Negative THEN angle = 180 - angle' Adjust if sign is negative.
  RETURN
```

Your Turn - LCD Display

Modifying the example program to display the tilt measurements on the Parallax Serial LCD is still a matter of adding an initialization routine and porting DEBUG commands to SEROUT commands. As with the program from Activity #5, this program displays characters that don't change in the initialization routine to prevent display flicker.

√ Save HorizontalTilt.bs2 as HorizontalTiltLcd.bs2
√ Replace the DEBUG command in the Initialization routine with this.

```
' Initialize LCD
PAUSE 200
SEROUT 14, 84, [22, 12]
PAUSE 5

SEROUT 14, 84, [128, "x-tilt=",
                148, "y-tilt="]

SEROUT 14, 84, [255,                  ' Define custom character 7
                %01000,               '   *
                %10100,               ' *   *
                %01000,               '   *
                %00000,               '
                %00000,               '
                %00000,               '
                %00000,               '
                %00000]               '
```

√ Replace the first DEBUG command in the Main Routine's DO...LOOP with the SEROUT command below. Make sure there are four spaces between the quotation marks. The four spaces are needed to erase the maximum of four characters that the command might send to the LCD: a negative sign, two digits, and the custom character 7 degree symbol.

```
SEROUT 14, 84, [135, "    ", 135, SDEC angle, 7]
```

√ Replace the second DEBUG command in the Main Routine's DO...LOOP with this. Again, make sure to put four spaces between the quotation marks to erase the previous value.

```
SEROUT 14, 84, [155, "    ", 155, SDEC angle, 7]
```

√ Change PAUSE 100 to PAUSE 350.
√ Run the program and test the display.

Your Turn - Adjustments

If your display did not go all the way to 90° when you held your board with a particular axis vertical, you can customize your scaling and offset to get it to fit. This will involve determining your accelerometer's actual output scale. If it's really 1865 to 3100, repeat the steps in Activity #3 to make the scaling and offset corrections.

SUMMARY

This chapter focused on sensing the acceleration due to gravity with the Memsic 2125 Dual Axis Accelerometer. Sensing gravity makes it possible to measure both tilt and rotation. The Memsic Accelerometer transmits pulses that indicate the acceleration acting on its x and y axes. At room temperature, the pulses range from 3750 to 6250 µs, which can be used to measure a range of −1 to 1 g with either one of the accelerometer's two sensing axes. The PULSIN command is used to measure these pulses, and since it measures time in 2 µs units, the range programs have to examine is 1875 to 3125.

Accelerometer measurements can be displayed with the Parallax Serial LCD. If the program has already been tested with the Debug Terminal, displaying measurements with the serial LCD is typically a matter of adding an LCD initialization routine the beginning of the program and using SEROUT commands in place of DEBUG commands. Custom characters come in handy for displaying the degree symbol (°), and the Greek letter mu (µ).

The accelerometer can be used to measure rotation in the vertical plane. To do this, the BASIC Stamp must calculate the arctangent of the accelerometer's y-axis measurement, divided by its x-axis measurement. The x and y axis measurements have to be scaled and offset to fit in a range of −127 to 127, which is what the PBASIC ATN operator needs to return an angle, measured in binary radians. While degrees separate a circle into 360 segments, binary radians separate it into 256 segments. The PBASIC */ operator can be used to convert a given binary radian measurement to degrees.

The accelerometer can also be used to measure tilt angles. Since the component of gravity acting on each of the accelerometer's sensing axes is the sine of the tilt angle, the inverse sine or arcsine can be used on an axis' measurement to determine the tilt angle. An Arcsine subroutine can be used to calculate the angle (in degrees) given a value that ranges from −127 to 127. This range corresponds to sine values of −1 to + 1.

Since both the ATN operator and the Arcsine subroutine expect a value between −127 and 127, techniques for scaling and offsetting the accelerometer measurements were introduced. The range of measurements the BASIC Stamp collects from the accelerometer are on a scale of 1875 to 3125. The most efficient way to scale these values to a range of −127 to 127 involves subtracting 1875 zero-align the range, then using the ** operator to reduce the scale, then subtracting 127. This is the resulting line of code: **`value = (value MIN 1875 MAX 3125) - 1875 ** 13369 - 127`**. The value 13369 is determined by the ** scale constant equation in Activity #2.

Questions

1. What are seven quantities you can measure with an accelerometer?
2. What are seven applications an accelerometer can be used in?
3. What does MEMS stand for?
4. What kind of acceleration do you sense in an elevator?
5. What moves inside the MX2125 when you tilt it?
6. What is it about the MX2125 that relates acceleration to temperature?
7. Can gravity be considered a form of acceleration?
8. What kind of electronic signals does the MX2125 send?
9. What do you have to do to a program that displays measurements in the Debug Terminal to make it display measurements in a serial LCD instead?
10. What's the equation for the ** scale constant?
11. How can you restrict a variable to a range of values?
12. How do the signals the MX2125 send relate to units of gravity? What signal corresponds to −1 g, 0 g, and 1 g?
13. How can you orient your board to apply 1 g to the accelerometer's x-axis?
14. How can you orient your board to apply −1 g to the accelerometer's y-axis?
15. How can you orient your board to apply 0 g to both axes.
16. What does the arctangent calculation tell you?
17. What's the difference between a binary radian and a degree?
18. What's the equation for the */ scale constant?
19. What range of values do the SIN and COS operators accept? What do these values represent?
20. What range of values will the SIN and COS operators return? What do these values represent?
21. How can you convert from brads to degrees?
22. How can you convert from degrees to brads?
23. What range of values does the ATN operator accept? What do these values represent?
24. What range of values does the ATN operator return? What do these values represent?
25. Why can you use ATN to calculate your board's angle of rotation?
26. What's the radius of a unit circle?
27. What range of values is the Arccosine subroutine designed to accept? What do these values represent?

28. What range of values is the Arccosine subroutine designed to return? What do these values represent?
29. What range of values is the Arcsine subroutine designed to accept? What do these values represent?
30. What range of values is the Arcsine subroutine designed to return? What do these values represent?
31. Why is it necessary to use the Arcsine subroutine to determine tilt angle?

Exercises

1. Write a command that receives the acceleration measurement from the accelerometer's y-axis output pin is connected to P10.
2. Write a command that receives the acceleration measurement from the accelerometer's x-axis output pin is connected to P9.
3. Write a command that converts the x-axis measurement to microseconds.
4. Write a command that converts the x-axis measurement to milliseconds.
5. Write a line of PBASIC code that scales a range from 0 to 100 to a range of 20 to 32.
6. Write a line of PBASIC code that scales a range from 40,000 to 50,000 to a range of −17 to 17
7. Write a command that prevents a variables value from going below 100 or above 1000.
8. Write a command that prevents a variable's value from going below −100 or above 100. Hint: The MIN and MAX operators are not designed for negative numbers. Sift the range to 1 to 201 before limiting the values, then shift back.
9. Write a command that receives a signed decimal values from the Debug Window's transmit windowpane and stores it in a variable named temp.
10. Write a command that multiplies a variable named temp up by 10.13.
11. Calculate how many brads are in 45°.
12. Calculate how many brads are in 355°.
13. Calculate how many degrees are in 20 brads.
14. Calculate how many degrees are in 250 brads.
15. If the result of the SIN operator is 63, what must have been the value it received?
16. If the result of the COS operator is -127, what must have been the value it received?
17. If the result the ATN operator returned is 128, what can you say about the two arguments it received?
18. If the result of the ATN operator returned is 160, what can you say about the two arguments it received?

Projects

1. Design a device that counts the number of vertical circles it has rotated. Try putting it inside a tire and rolling it down a hill.
2. Design a device that you can hang from a pendulum and it counts the number of swings.
3. Mount your board on a standard servo's horn, and program it to stay level as you rotate it.
4. Mount your board on a standard servo so that it rotates the board in the vertical plane. Use the Ping))) detector to report the angle of the closed object. Cause the servo to sweep every 10 seconds and then point at the closed object.
5. To make a self calibrating unit, you can cause the LCD to point in four different directions (up, down, left, right). Each time the LCD points in a given direction, the user should rotate the unit until the arrow on the LCD is pointing straight up. Then, the user should press a button, let go and hold it there for a second. The unit should beep when it is done testing the accelerometer's measurement. The program should then develop it's own scale factors so that it displays an accurate scale as you tilt it.