Wireless Musical Expression Pedal
"Magic Wah"
Freescale Wireless Challenge
Project # FZ1592

Bill Bishop
Sixth Sensor
8500 Baileycroft Drive
Raleigh, NC 27615
sixthsensor@hotmail.com

Abstract

The Magic Wah is a wireless expression pedal for musical instruments. The system is based on the popular wah-wah pedal that has been around since the early 1960's. It produces a distinctive tone that many musicians rely use to add expression to their playing. The traditional wah pedal requires that the musician manually step on the wah pedal to make the "wah" sound. The wireless wah pedal frees the musician from the need to stand directly at the pedal and offers many other advantages over the traditional wah pedal.

Traditional wah pedals are simple analog circuits. The wah effect is accomplished by sweeping the frequencies of a bandpass filter as the user steps on the pedal. The act of stepping on the pedal turns a potentiometer that changes the bandpass filter frequencies making the "wah" sound. The Magic Wah uses the same type of analog circuitry to achieve the "wah" sound, but rather than step on a pedal to control the sound, a sensor, worn on the foot, controls the "wah" effect. The analog potentiometer is replaced with a digital potentiometer that is adjusted by a microcontroller.

The Magic Wah system consists of two physical devices that communicate with each other point to point using the 802.15.4 PHY layer. The transmitter is worn on the foot of the musician, and the receiver is plugged into the musical instrument signal path. The transmitter constantly monitors the foot angle of the musician and transmits the data to the receiver. The receiver translates the foot angle into a digital potentiometer setting that changes the frequency of the analog bandpass filter, thus creating a "wah" effect as the foot is moved up and down.

The Magic Wah was developed using a pair of Freescale 13192 SARD (sensor applications reference design) boards. The boards include MCS908GT microcontroller, MC13192 2.4GHz RF transceiver and two accelerometers: MMA6261 (X, Y axis) and MMA1260D (Z axis). The analog "wah" circuit is controlled by the software using a microcontroller output port that adjusts the value of the digital potentiometer (Dallas Semiconductor DS1804 100Kohm).

The MMA6261 and MMA1260D accelerometers are used to measure overall movement, detect specific gestures, and to measure tilt. The system is turned off and on by simple foot gestures. To turn the wah pedal on, the user simply plants their heel down with toe pointing up. To turn off, the foot is moved to the side laterally. When no movement is detected for an extended period of time, the software will transition into a low power mode to extend battery life.

Magic Wah With Guitar and Amp

Demonstration Sound File: wahExample.mp3

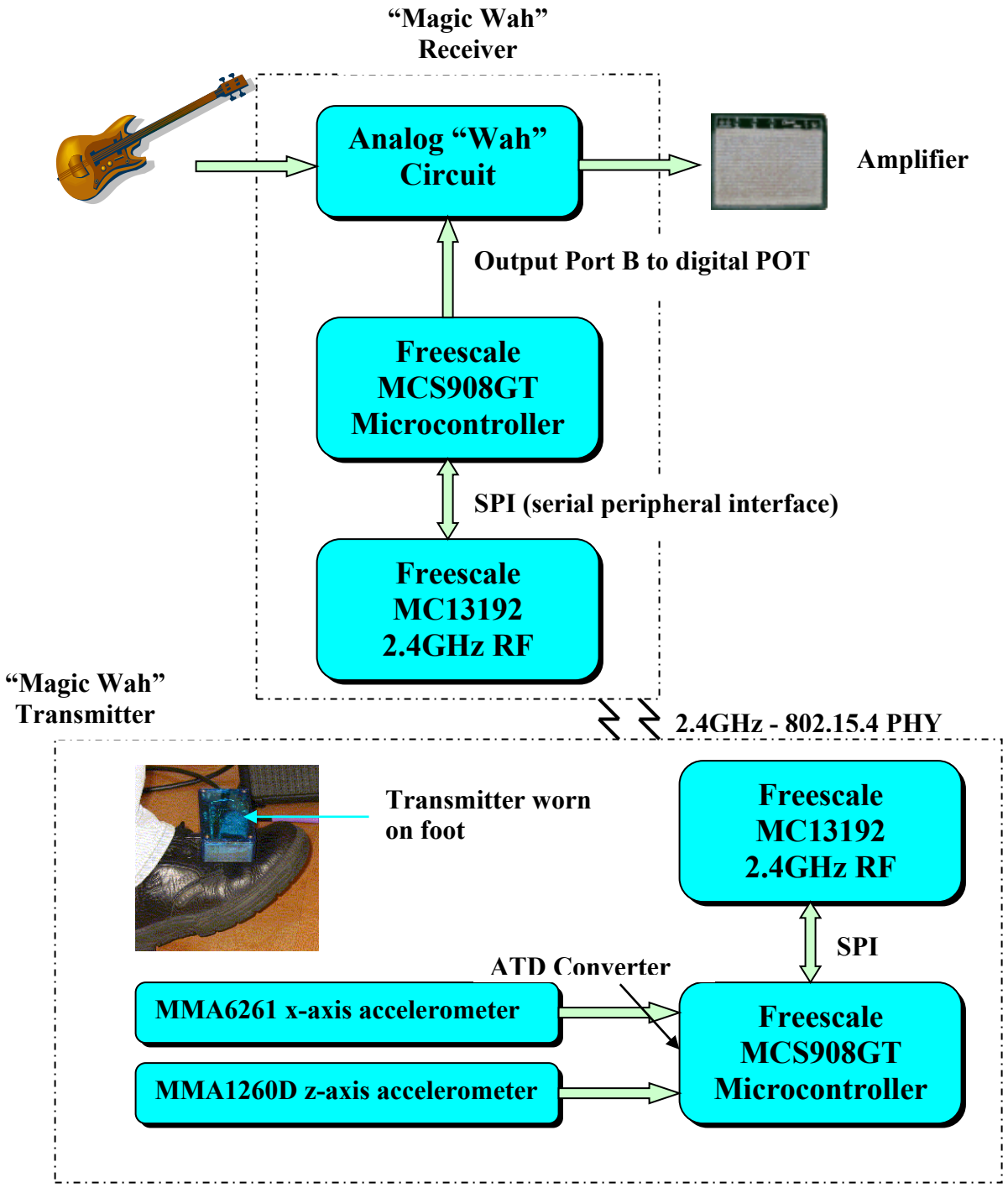
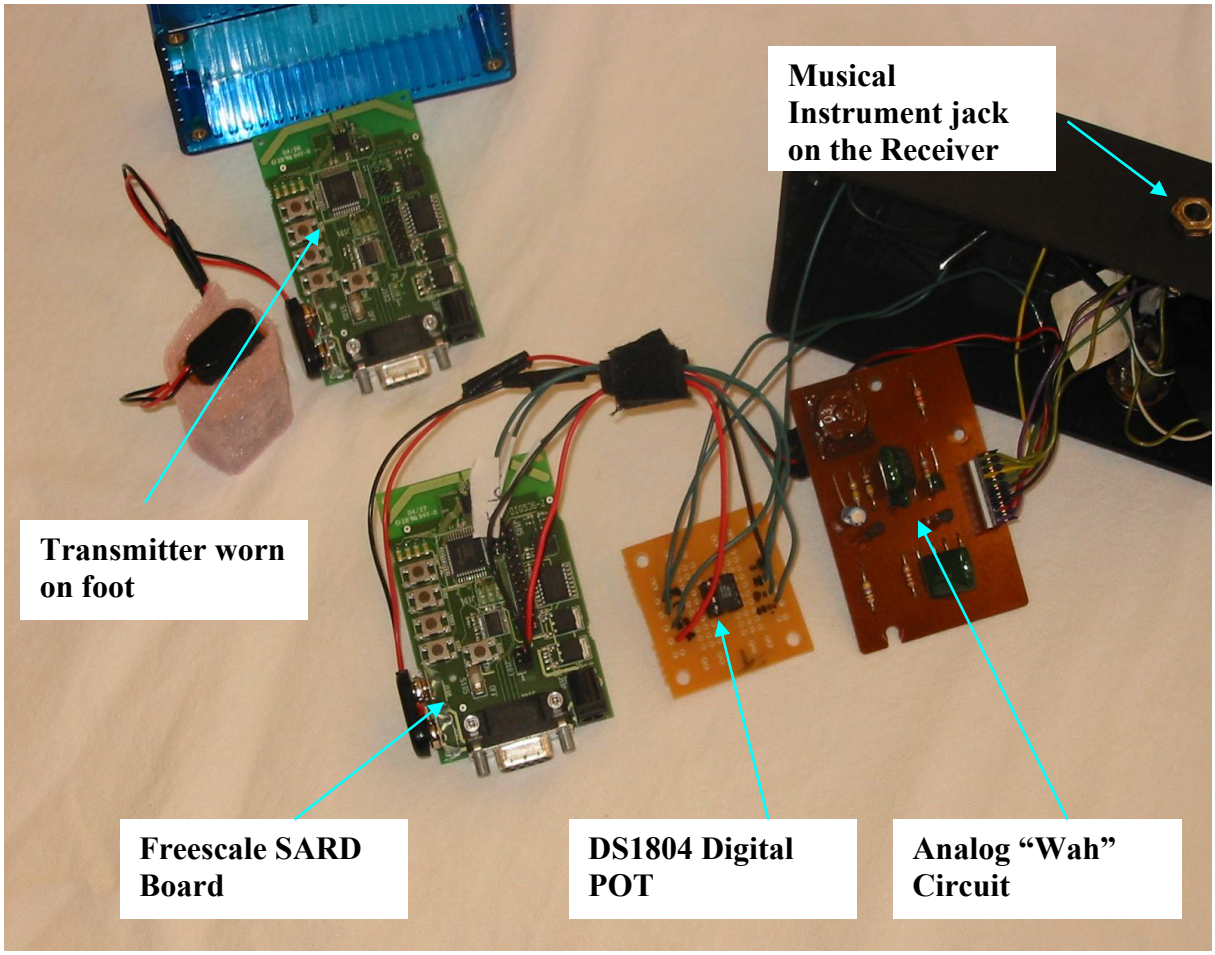
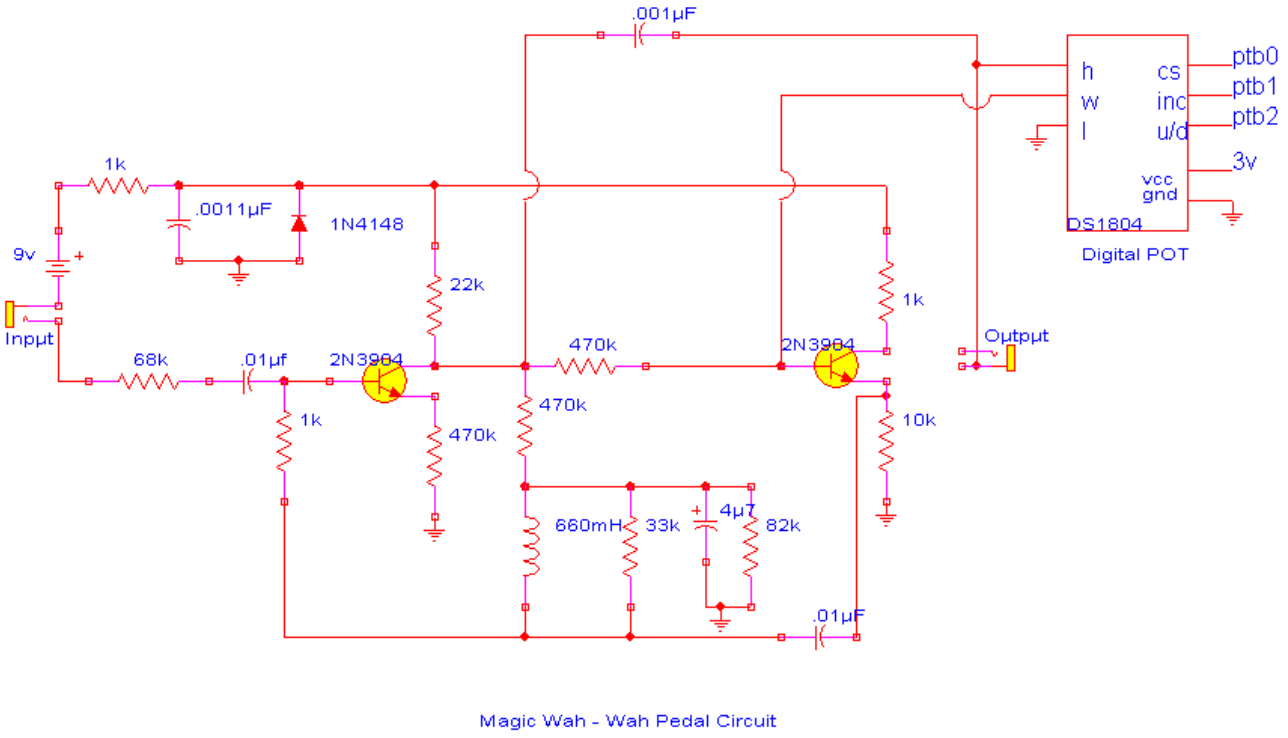


Figure 1 – Magic Wah Block Diagram

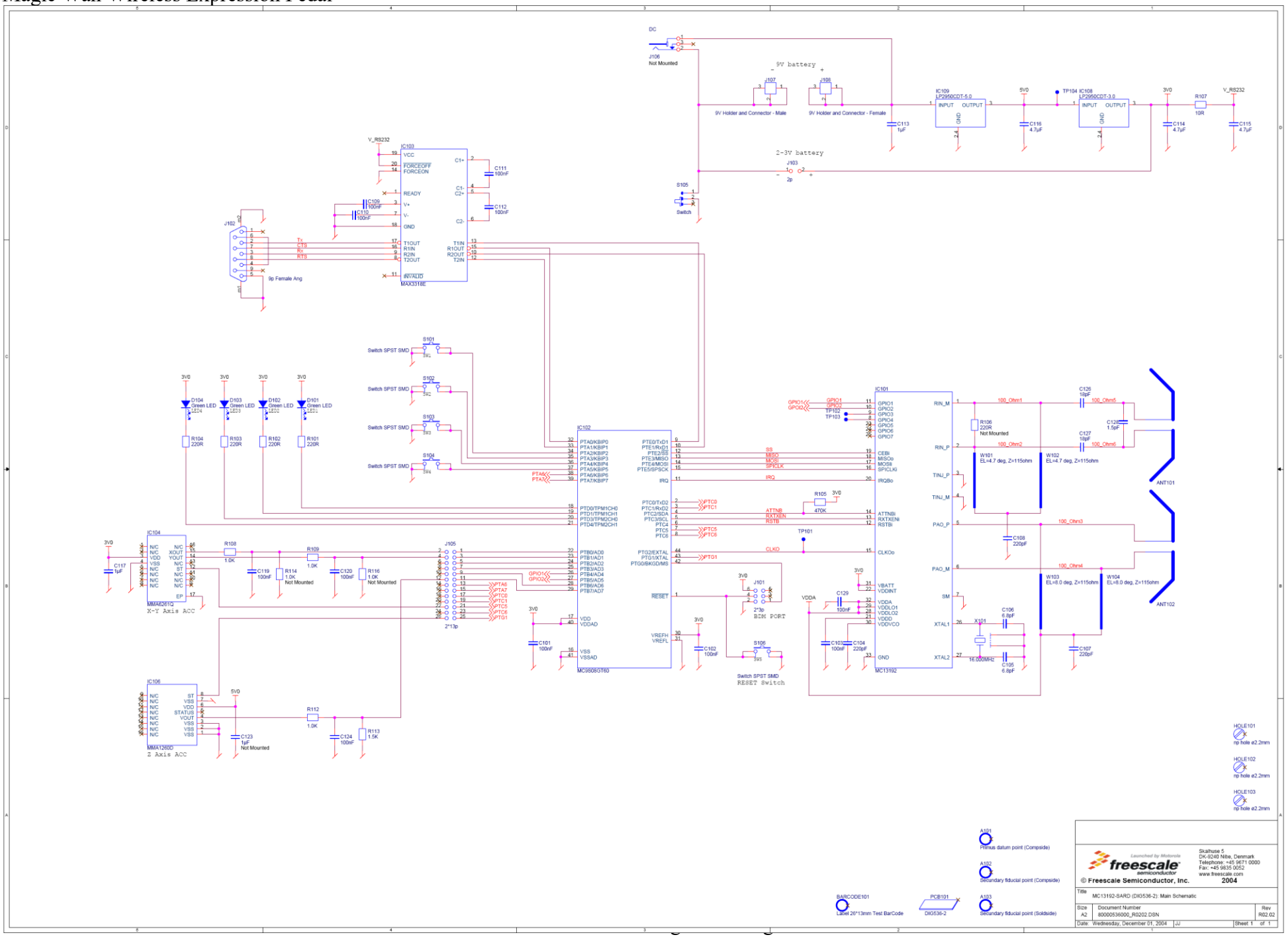
Magic Wah Components



Analog Wah Circuit Schematic and SARD Board Schematic



Magic Wah - Wah Pedal Circuit


9V battery
2-3V battery
9V Holder and Connector - Male
9V Holder and Connector - Female
Not Mounted
Switch
9p Female Ang
MAX3318E
Switch SPST SMD
Green LED
220R
MMA6261Q
X-Y Axis ACC
MMA1260D
Z Axis ACC
MCU908GT60
MC13192
2*13p
2*3p
BDM PORT
RESET Switch
16.000MHz
100_Ohm1
100_Ohm2
100_Ohm3
100_Ohm4
100_Ohm5
100_Ohm6
W101 EL=4.7 deg, Z=115ohm
W102 EL=4.7 deg, Z=115ohm
W103 EL=8.0 deg, Z=115ohm
W104 EL=8.0 deg, Z=115ohm
ANT101
ANT102
HOLE101
HOLE102
HOLE103
np hole ø2.2mm
BARCODE101
Label 26*13mm Test BarCode
PCB101
DIG536-2
A101
Primus datum point (Composide)
A102
Secundary fiducial point (Composide)
A103
Secundary fiducial point (Soldside)
© Freescale Semiconductor, Inc. 2004
Skaihuse 5
DK-9240 Nibe, Denmark
Telephone: +45 9671 0000
Fax: +45 9835 0052
www.freescale.com
MC13192-SARD (DIG536-2): Main Schematic
Document Number
80000536000_R0202.DSN
Rev R02.02
Wednesday, December 01, 2004
Sheet 1 of 1

Magic Wah Advantages

- **Cost**. The electronics in a traditional wah pedal are very simple. However, they are costly because the case is made from steel to support the weight of the user. The Magic Wah is encased in an ABS plastic case for durability and low cost.
- **Freedom**. The Magic Wah allows the musician to roam the stage with greater freedom.
- **Weight/size**. Traditional pedals are big, bulky devices.
- **Greater range of effect**. The range of motion in a traditional pedal is limited because of the bulk of the pedal. Several inches of foot motion is wasted because of the base of the pedal. The full range of foot motion is utilized with Magic Wah. This results in greater control over the wah effect.

Accelerometer Theory

The X,Y and Z axis accelerometers are used for multiple purposes in this application. A movement software interface was created to abstract the hardware interface and algorithms used to sense movement. The acclerometer.c file contains the movement sensing layer, and was designed with the goal of eventually being made into a library system that could be reused across applications. The following section describes the algorithms developed and challenges faced while developing this application:

Tilt Sensing. One method to determine the angle of tilt on an axis is to take the arcsine of acceleration in Gs (X and Y axis) or arcos of acceleration in Gs (Z axis). The following diagram demonstrates this principle.

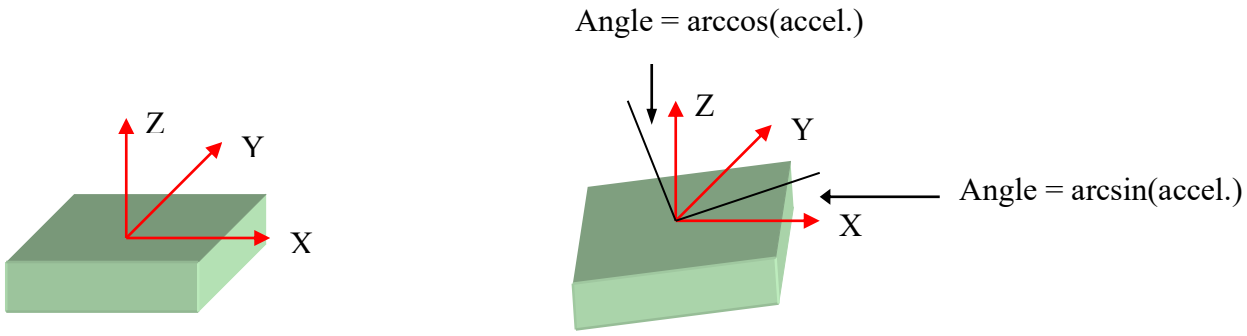


Figure 2 – Measuring Single Axis Tilt Using ASin/ACos

This method works if the board is not in motion, however it is difficult to differentiate between motion and tilt. Acceleration can be caused by tilting the board or by simply moving the board on axis (no tilt). The same acceleration reading can result from either action. However, tilt can be accurately calculated if two axes are combined. Regardless of the speed at which the board is moving, the ratio of the two axes will accurately determine the angle of the board. In this case the board is oriented such that it moves through the Y and Z axes. The board is essentially the hypotenuse and the angle can be measured by using the arctangent function of the two accelerations. One other note is that conversion to gs is not necessary since the ratio of Y to Z raw values yields the same results.

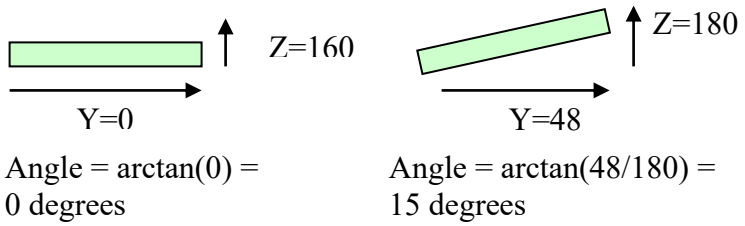


Figure 3 – Measuring Tilt with Two Axes

Movement Sensing. Sensing overall movement is accomplished by using the 3D version of Pythagorean's Theorem: total vector = sqrt ($x^2 + y^2 + z^2$). The movement detection software samples the 3 axes for a period of two seconds. Each time a sample is taken, the total vector is calculated and stored using Pythagorean's Theorem. At the end of the interval the standard deviation of the total vectors is calculated. If the standard deviation is greater than a preset value then movement has occurred. Using the standard deviation eliminates noise in the samples. This method also protects against constant movement detection due to riding in a vehicle.

Gesture and Jolt Sensing. To turn on the Magic Wah effect, the user plants their heel down with toe pointing up. This causes the accelerometers to register a jolt. When the jolt is sensed, the angle is measured. If the appropriate angle is measured then the Magic Wah effect is turned on. Jolt is detected by comparing the current accelerometer sample to the previous sample. If the difference is greater than a predefined value, then a jolt has occurred.

Accelerometer Orientation. The orientation of the transmitter board worn on the foot is critical because it determines the sensitivity of the system. The accelerometers are more sensitive to tilt when the sensitive axis is oriented perpendicular to the force of gravity. Therefore, the board will be positioned on the foot such that the Z axis is not in its 1g or –1g position (i.e. not flat). For demonstration purposes the board will be attached to the arched part of the foot and not the flat end of the foot. The mounting position of the accelerometers on the SARD board dictates how it will be mounted to the foot for demonstration purposes. For production models the sensor will be packaged such that it the axis of sensitivity is close to perpendicular its 1g or –1g position and mounted at the edge of the board closest to the toe.

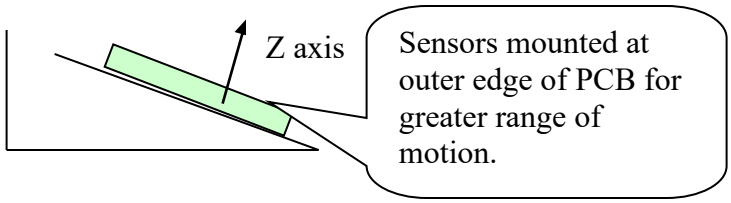


Figure 4. Sensor Orientation and Mounting

Sample Noise. The bandwidth of the MMA6261 X,Y axis accelerometer is 300hz, and 50hz for the MMA1260D Z axis accelerometer. The Z axis is not as susceptible to high frequency vibrations as are the X and Y axis. High frequency vibrations are particularly problematic when the Magic Wah changes direction. Regardless of the sample rate, some high frequency noise was sampled. Software filtering was used in lieu of hardware filtering to remove the noise. After much trial and error the sample rate of 250hz averaging every 8th sample (net 31.25hz) yielded the best results, removing almost all of the high frequency noise. The file linked here contains a hyperterm session that captures the beta test of the Magic Wah. From the file you can see the calculated foot angle and corresponding potentiometer setting that was set: *Hyperterm Session*

RF Communications Software

The two components of the Magic Wah communicate point to point using proprietary 802.15.4 PHY packet data. The SMAC (Simple media Access Controller) software supplied by Freescale has been modified and used to send and receive physical packets over the 2.4GHz RF link, providing layer 1 and layer 2 protocol support.

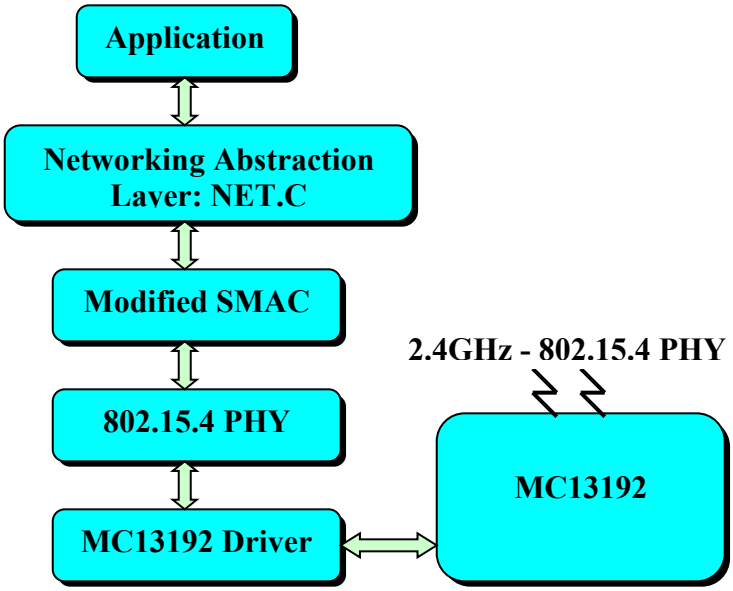
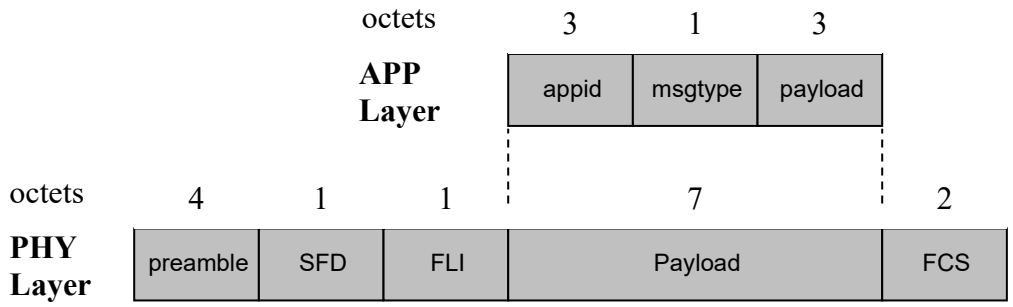


Figure 5. Magic Wah Communications Protocol Stack

Network Abstraction Layer

The network abstraction layer attempts to abstract the implementation details of sending and receiving data packets on the network. Currently the application supports point to point protocol. However, it is always beneficial for future development to abstract implementation details from the application. The application registers with the network layer, which will invoke a callback procedure when properly formed packets destined for the node arrive. The application does not care what medium to which the packets are being sent. The application data is kept to a minimum to reduce power requirements. The format of the packet is shown below. Because this is a real-time system, acknowledgments are not required for the wah pedal movements, therefore the link becomes connectionless once the wah pedal is turned on. The physical medium transmits at 250kbit/s = 32usec/byte. Therefore, the data transmission time for a data packet is 480usec.

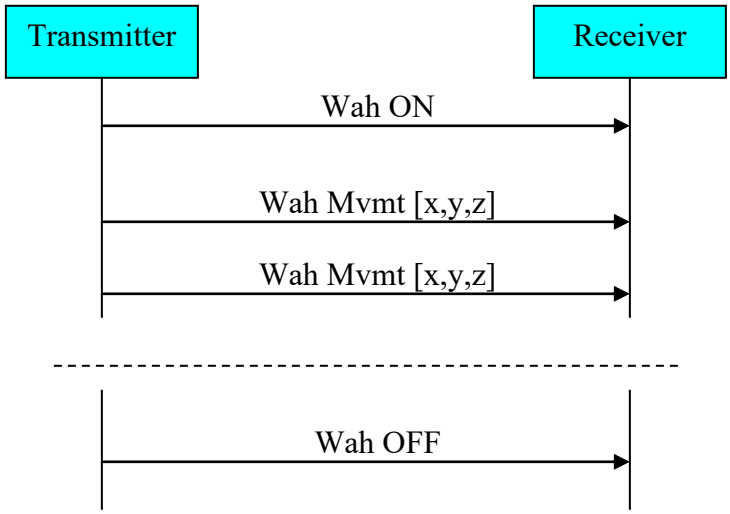


Field	Octets	Description
appid	3	Used to identify the application. The string "WAH" is used to identify this application. The appid is used to filter out any other 802.15.4 packets that may be received on the specific channel. If the appid does not match "WAH", then the packet is tossed.
msgtype	1	The type of message being transmitted. Valid message types are: 0 – heartbeat 1 – wah pedal transitions to ON 2 – wah pedal transitions to OFF 3 – wah pedal movement: payload=x,y,z raw values. 4 - acknowledgment
payload	3	Any message specific data, up to 3 bytes. Currently only wah movement requires a payload, which contains x,y,z raw data values.

Table 1. Magic Wah Communications Protocol

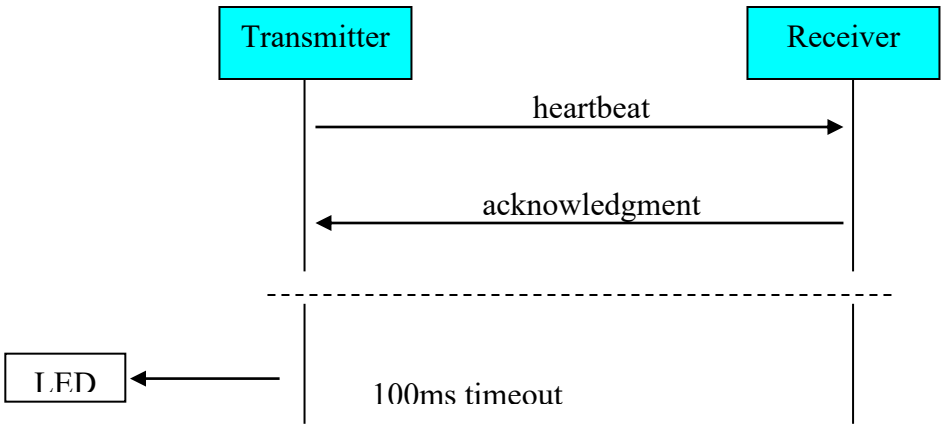
Wah Pedal Movements

When the wah pedal is switched on the transmitter begins sending X,Y, and Z coordinates. There is no need for acknowledgments since this is a real time system. Acknowledgments are used only during ready mode to ensure that the two units are communicating.



Heartbeat/Out of Range Detection

When the transmitter detects movement it enters Ready mode. In Ready mode the transmitter sends heartbeat messages every five seconds and waits for an acknowledgement from the receiver. If the acknowledgement is not received in timeout window an LED is turned on to inform the user that the two devices are not communicating properly.



Channel Selection

The user can select an RF channel by pressing the keyboard button 1. Both components must be communicating on the same channel. The user can monitor the status of the ready LED to determine if the devices are communicating. A future enhancement of the software will include RF channel auto detection.

Analog Hardware Overview

The analog circuitry used to shape the instrument signal is based on a simple design. The circuit is a bandpass filter in which the resonant peak can be adjusted up and down in frequency by moving the

pedal, making the distinctive "waaaah" tone, or the inverse, "aaaooow". The analog circuit design for the Magic Wah was taken from the Dunlop “Cry Baby” wah pedal - one of the most popular wah pedal models. The only difference between a traditional wah pedal circuit and the Magic Wah analog circuit is the digital potentiometer that is controlled by the receiver board based on the tilt angle of the transmitter. This results in no tonal difference between the traditional wah pedal and the Magic Wah.

Low-Power Scheme

The system clock on the SARD boards is a 16MHz crystal that drives the MC13192 transceiver. The MC13192 then supplies an external clock signal that can be used as an external clock source for the MCU. The MCU can also run in a self-clocked mode for reduced clock requirements. However, for simplicity and versatility, the MC13192 external clock is used as the system clock for this application. The clock scheme is central to the design of the software and impacts the implementation of low power scheme.

The MCU and transceiver both support multiple low power modes. The application was designed to take advantage of the low power modes of both devices. By far the biggest consumer of battery power is the RF transceiver. Therefore, it is important that the RF transceiver is not turned on until necessary. However, the dozing transceiver must continue to supply the clock source to the MCU. Therefore, the “Doze -Acoma” mode is used, in which the MC13192 continues to supply an external clock even though the chip is in low power mode. In addition the MCU will spend much of its life in Stop mode. Stop3 mode is used for its simplicity and flexibility.

Transmitter power strategy

The duty cycle of the MCU in the transmitter is relatively high because of the intensive math involved to determine if movement has occurred. The mathematical algorithms were written using fixed point math for speed, but it still requires many MCU cycles to determine the standard deviation of the square root of the sum of the squares for a two second period! However, switching on the RF transceiver is much more battery consuming than crunching numbers on the MCU. The following table summarizes the power consumption of the transmitter device. Because Wah pedals are used sparingly by musicians (usually only for small sections) the device will be in Idle or Ready state for most of the time. If the device is switched off or even left in idle state, it will provide long battery life. However, if the device is left in the Run state for an extended period, the battery life will be reduced significantly.

Software State	MCU duty cycle	power consumption	MC13192 duty cycle	power consumption
Idle State (Movement Detection)	.94 stop3 (.766ua) .06 run (81ua)	.00000072004ma/s .00000486ma/s	Doze mode. Duty cycle=0.	.000040 ma/s
Ready State (Gesture detection)	1.00 run (81ua)	.000081ma/s	29.899ms doze/30sec 480us tx/30sec 100ms rx/30sec	.0399 ma/s .00048 ma/s .1233 ma/s
Run State (Tilt measurements)	.25 stop3 (.766ua) .75 run (81ua)	.0000001915 ma/s .000061ma/s	85ms idle/sec 15ms tx/sec	.0425 ma/s .450 ma/s

Assumptions: Ready state worst case assumed, ack timeout
RTI enabled in stop mode
LVI disabled in stop mode

	ADC power consumption not factored
Totals:	Idle: .00004558004 ma/s
	Ready: .1636453 ma/s
	Run: .4925611915 ma/s

Table 2. Magic Wah Transmitter Device Power Consumption

Receiver power strategy
In this prototype version of the project, the receiver can be powered by the 9v power supply or an internal 9v battery for convenience. However, if the user wishes to use battery supply then the battery life will be reduced significantly if the device is not manually switched off. In the productized version, the ¼'' musical instrument plug will automatically turn the device on and off. The MC13192 is constantly powered in receive mode therefore the 9v power supply will be the suggested mode of operation.

Software Overview

The software was developed using the Metrowerks Codewarrior for HCS08 Special Edition. The BDM port was used for in circuit debugging and flash programming. The software is written in C language and some assembly language.

For speed optimization all mathematical operations are accomplished with fixed point math, there are no floating point variables used.

Finite State Machine Operating System

The statemach.c module contains the implementation of the state machine. A simple finite state machine operating system was developed to make the application more robust, maintainable, and also as a platform that can be reused for other applications. Of course there is always the tradeoff between complexity and system resources. In this case the FSM was developed with 1.5K bytes of code, so it is very limited by design, but allows for a more manageable software design. The system is event driven by nature, and each finite state is responsible for handling a set of events that are passed to it by the main application driver.

The responsibility of the main application driver is to initialize the overall system and implement a for loop calling the state machine each time through the loop. In order to implement a low-power cycle, each state is responsible for implementing a low power event (IDLE_LOOP_WAIT), which is called at the top of the loop. After a low power cycle, the main driver calls *handleLoopTimer*() which pegs the timer counters. If any events have occurred while in low power cycle they will be processed at the bottom of the loop when the state machine is called again.

Events are generally captured by timer expiries and from polling for hardware events. Below is a list of events used in this application.

event.h

```
// Events that drive the finite state machine for this application
typedef enum {
  NIL_EVENT,
```

```
  SYSTEM_INIT,
  MVMT_SAMPLE_READY, // a movement sample is ready
  TIMER_EXPIRED,     // timer has popped
  MVMT_OCCURED,      // movement has been detected
  ACK_RECEIVED,      // RF Ack received
  ACK_TIMEOUT,       // RF Ack timed out
  KB_PRESS,          // a KB press has occured
  KB_EVENT,          // debounce finished, process kb now
  IDLE_LOOP_WAIT,    // go into low power mode
  MAX_EVENTS
} t_EventId;
```

main.c (main application driver)

```
  // Initialize state machine
  event.eventId = SYSTEM_INIT;
  appState = stMachHandlers[IDLE_STATE](&event);

  for (;;) {
    // ******************************************************
    // Low power handling
    // To achieve lowest power setting we don't want to
    // continuously process, so we'll create a sleep event
    // each time through the loop.  The state machine can
    // determine how to handle low power mode because it
    // may be state specific.
    // ******************************************************

    // This event must be handled by each state and must return
    // an event to process - or NIL_EVENT if nothing to process
    event.eventId = IDLE_LOOP_WAIT;
    appState = stMachHandlers[appState](&event);
    // End Low Power handling

    // ******************************************************
    // Increment timers, don't process timeouts if movement
    // sample is ready, just increment timers.  We'll get back
    // to expired timers after checking for movement.
    // ******************************************************
    handleLoopTimer(&event, event.eventId != MVMT_SAMPLE_READY);

    // ******************************************************
    // Send the event to the state machine
    // ******************************************************
    appState = stMachHandlers[appState](&event);
  }
```

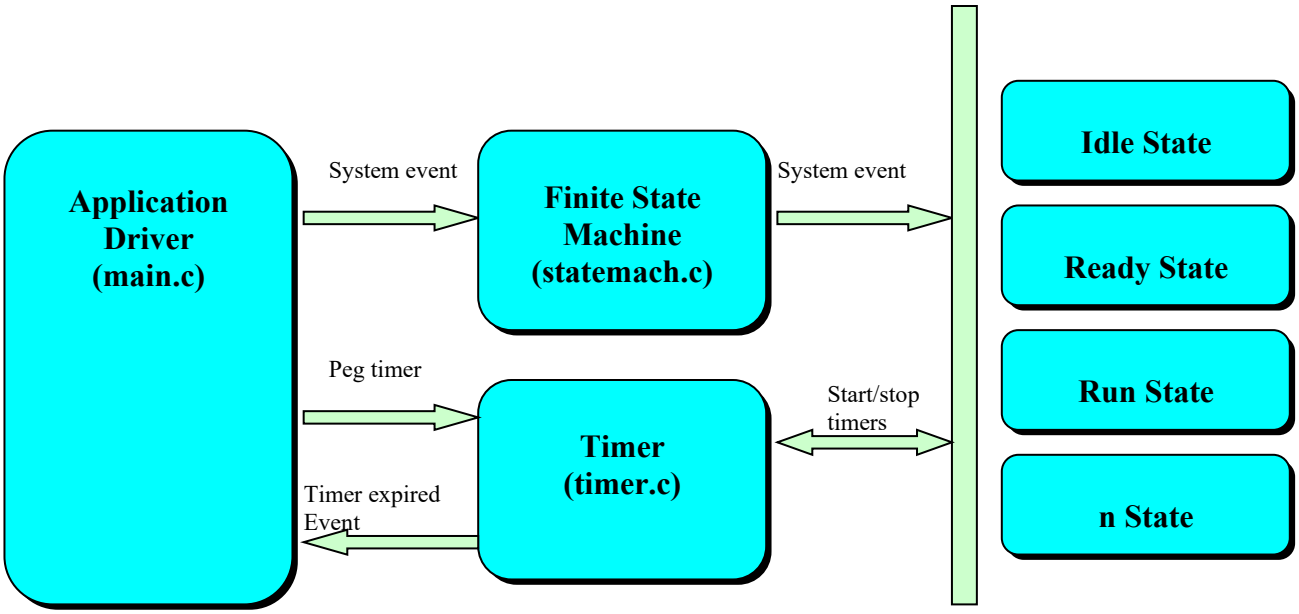


Figure 6. Finite State Machine Operating System

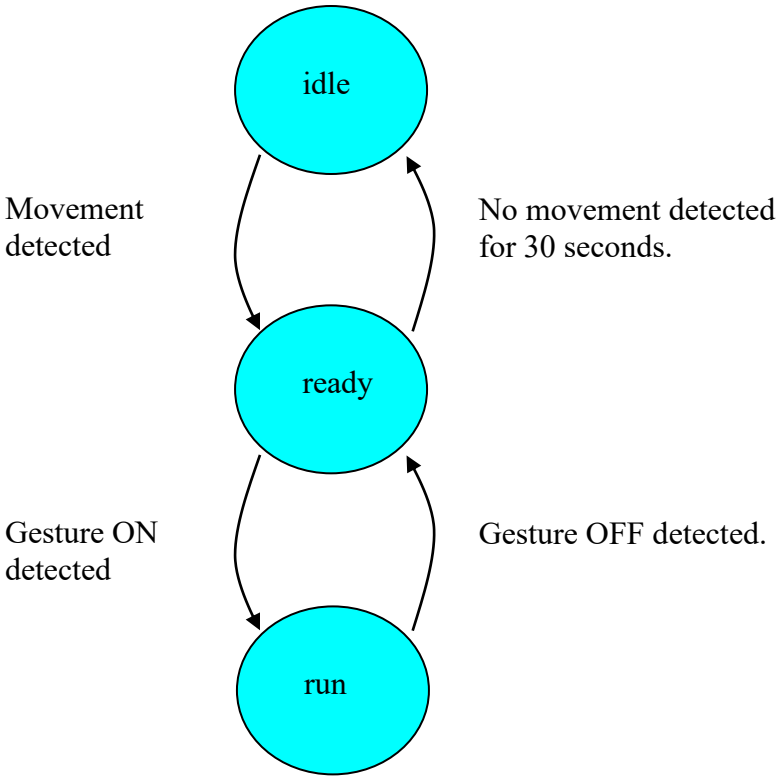


Figure 7. Transmitter Finite State Transitions

Hardware Absraction Layer

The HAL.C Module contains the Hardware Abstraction Layer. The HAL abstracts the implementation details of the MCU and RF device from the application. Constructs such as clock scheme, low power mode, delays, etc. are implemented in this module.

Accelerometer Algorithms

Movement Detection

As described above in the accelerometer theory section, movement detection requires using Pythagorean's theorem to determine the total vector of movement. The total vectors are stored in a 2 second buffer. At the end of each 2 second interval, the standard deviation is taken on the total vector of each sample in the 2 second buffer. If the standard deviation is greater than a predefined value, then movement has occurred. The predefined value was arrived at by trial and error wearing the transmitter device on the foot. All calculations are done using fixed point math for quick execution. There are no floating point variables used in this application.

```
// anything greater than this std deviation means movement occured
#define NO_MOVEMENT_DEVIATION 4

/*************************************************************************
 * calcVariance
 *
 * Description:  Determines standard deviation from the mean for active buffer.
 *               Standard Deviation is calculated as the square root of the
 *               variance.  The variance is the average of the differences
 *               from the mean of the series. A data series like 1, 2, 3, 6
 *               has a mean equal to (1+2+3+6)/4=3. The differences from the
 *               mean are: -2, -1, 0, +3. The variance is: {(-2)^2 + (-1)^2 +
 *               0 + (3^2)}/4=14/4=3.5. Finally, the standard deviation is
 *               equal to the square root of the variance: SQR(3.5)=1.87.
 *
 * Parms:        mean - mean of the series
 *
 * Returns:      variance from the mean for the series
 *************************************************************************/
unsigned long calcVariance(int mean)
{
  int i=0;
  int variance;

  static unsigned long vSquare=0, vSquareTotal=0, vSquareAvg=0;
  static int vInt=0;
  static unsigned long nSamples;
  static unsigned long stdDeviation;

  vSquareTotal=0;
  for (i=0; i<maxSampleIdx; i++) {
    variance =  pActivityTable->sample[i].integratedSample - mean;
    variance = abs(variance);
```

```
    vInt = (int)variance;
    vSquare = vInt * vInt;
    vSquareTotal+=vSquare;
  }

  nSamples = maxSampleIdx;
  vSquareAvg = vSquareTotal/nSamples;

  // now the square root gives us std deviation
  stdDeviation = isqrt (vSquareAvg);

  return stdDeviation;
}

/*************************************************************************
 * movementDetected
 *
 * Description: Returns true if movement is detected over the entire sample
 *              period.
 *
 * Parms:       none
 *
 * Returns:     True if mvmt detected, false if not
 *************************************************************************/
int movementDetected(void)
{
  int retcode = 0;
  static unsigned long mean, sum, numsamp;
  static unsigned long stdDeviation;

  // the sum has already been calculated on the fly, just divide by num samples
  numsamp = maxSampleIdx;
  sum = pActivityTable->runningSum;
  mean = sum / numsamp;

  // calculate the variance from the mean
  stdDeviation = calcVariance((int)mean);

  if (stdDeviation > NO_MOVEMENT_DEVIATION) {
    // movement detected
    retcode = 1;
  }

  return retcode;

}
```

Tilt Angle Determination

As described above the tilt angle can be calculated by taking the arctan of Y_{acc}/Z_{acc}. For optimization the arctangent is calculated using a fixed point arctangent table. The table is indexed by radians (y/z) to determine the angle in tenths of a degree. Because of memory constraints and entire tangent table (360 degrees) would require too much memory. However, the range of motion of the human foot is limited to only a few degrees, so an entire tangent table is not needed. The trig table demonstrated here supports a 25 degree range in tenths of a degree. Therefore the system is sensitive to movements to a tenth of a degree over a 25 degree range.

trigtables.h

```
//
// Arctangent fixed point table from 45.0 to 71.9
//
// Each entry is a degree to a tenth of a degree.
// The index to this table is radians * ARCTANGENT_PRECISION.
//
// For example: 1.000 = 45.0 degrees
//              1.001 = 45.0 degrees
//              1.003 = 45.1 degrees

short fixedArcTangentTable[] =
{ 450,  450,  450,  451,  451,  451,  451,  452,
  452,  452,  452,  453,  453,  453,  454,  454,
  ..
}
```

Receiver: main.c

```
// Returns atan of parm1/parm2.  Returns angle in
// fixed point degress to a precision of tenth of degree
int fixedArcTangent2(long tanRad1, long tanRad2)
{
  static long tanRadians;
  int tanAngle = 0;

  tanRad1 *= ARCTANGENT_PRECISION;
  //tanRad2 *= ARCTANGENT_PRECISION;

  if (tanRad2>0) {
    tanRadians = tanRad1/tanRad2;
    tanRadians -= ARCTANGENT_NORMALIZE;
    if (tanRadians >= 0 && tanRadians < ARCTANGENT_NUMENTRIES) {
      tanAngle = fixedArcTangentTable[tanRadians];
    }
  }

  return tanAngle;
}
```

Digital POT Control

The Dallas Semiconductor DS1804 100Kohm is a 100 position increment/decrement digital potentiometer. The device replaces the analog POT that controls the resonant frequency of the wah circuit. The chip is powered by the 3-volt power supply from the SARD board. Three pins are controlled by the MCU output port B in order to increment and decrement the position as described below:

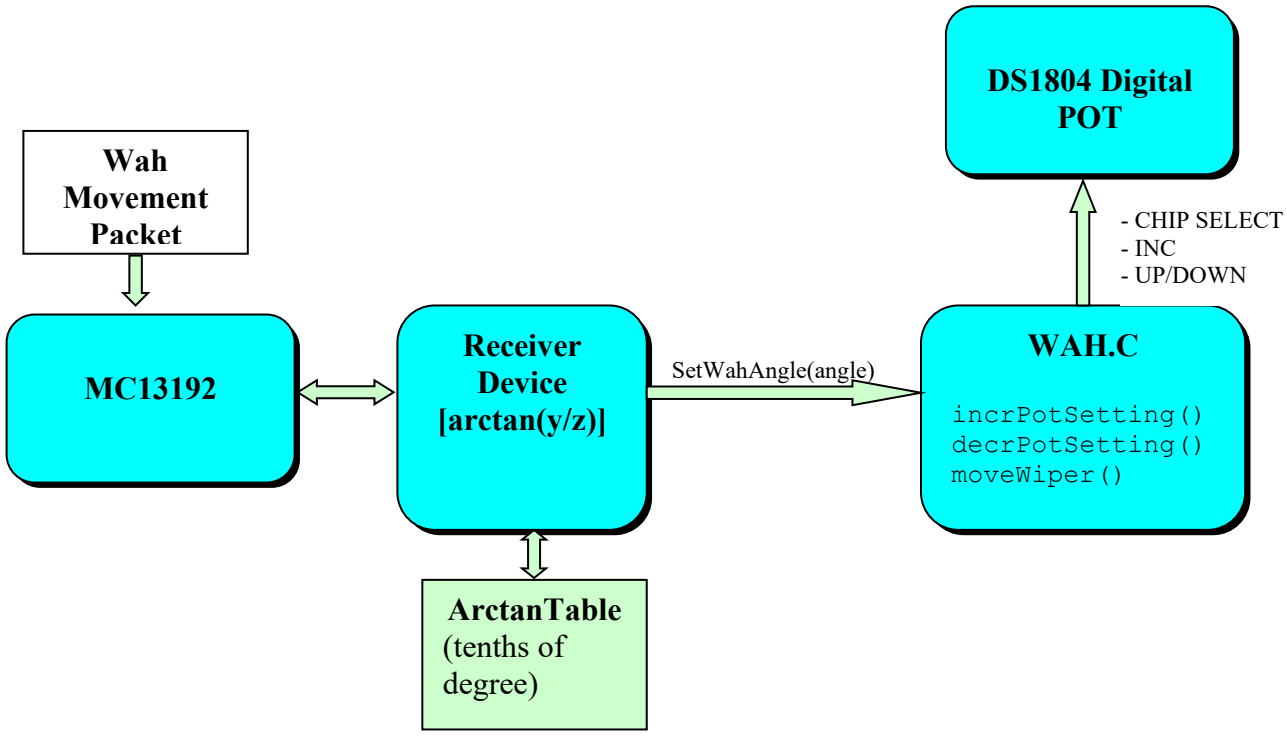


Figure 8. Setting Digital POT From Y/Z Accelerometer Sample

CS - Chip Select. The CS input is used to activate the control port of the DS1804. This input is active low. When in a high-state, activity on the INC and U/ D port pins will not affect or change wiper position.

INC - Wiper Movement Control. This input provides for wiper position changes when the CS pin is low. Wiper position changes of the W-terminal will occur one position per high-to-low transition of this input signal. Position changes will not occur if the CS pin is in a high state.

U/ D - Up/Down Control. This input sets the direction of wiper movement. When in a high state and CS is low, any high-to-low transition on INC will cause a one position movement of the wiper towards the H-terminal. When in a low state and CS is low, any high-to-low transitions on INC will cause the position of the wiper to move towards the L-terminal.

```
// DS1804 controlled by port D
#define WAH_PARALLEL_PORT     PTBDD

// Set our control pins to output
#define WAH_PORT_DIRECTION
      (PTBDD_PTBDD0_MASK|PTBDD_PTBDD1_MASK|PTBDD_PTBDD2_MASK|PTBDD_PTBDD3_MASK)

// Slew Port
#define WAH_PORT_SLEW         PTBSE

// Slew values for all bits
#define WAH_SLEW_VALUE
(PTBSE_PTBSE0_MASK|PTBSE_PTBSE1_MASK|PTBSE_PTBSE2_MASK|PTBSE_PTBSE3_MASK)
```

```
//#define WAH_SLEW_VALUE        0

// Port B0 is Chip Select
#define WAH_POT_CHIP_SELECT   PTBD_PTBD0

// Port B1 is INC/DEC
#define WAH_POT_INC           PTBD_PTBD1

// Port B2 is Direction (up or down)
#define WAH_POT_DIRECTION     PTBD_PTBD2

// Application interfaces
void setWahPedal(UINT8 stepValue);
void initWahPedal(int minAngle, int maxAngle);
void setWahPedalAngle (int topAngle);
```

The application calls initWahPedal to set the minimum and maximum angles that the system supports. When the wah pedal ON packet is received, the setWahTop() is called to set the top angle. As wah movement packets are received, the angle is calculated (using arctangent table) and setWahPedalAngle is called.

The POT driver then determines if the POT should be incremented or decremented and by how many steps. Once that has been calculated the POT is moved by calculated amount by calling one of these two functions that invoke the moveWiper function. Because the DS1804 timing values are much smaller than the clock speed of the 16Mhz MCU clock, the wiper move can be performed in a dead loop with no delays. The slew value is not a factor (enable or disabled would work), but has been enabled for fast transition times.

```
static void incrPotSetting(int nIncrement)
{
  // Drive direction to increment (high)
  WAH_POT_DIRECTION = TRUE;
  moveWiper(nIncrement);
}

static void decrPotSetting(int nDecrement)
{
  // Drive direction to decrement (low)
  WAH_POT_DIRECTION = FALSE;
  moveWiper(nDecrement);
}

static void moveWiper(int nMoves)
{
  int idx;

  // Drive INC high.  Each High to Low is an increment
  WAH_POT_INC = TRUE;

  // Drive CS low.  After this you must wait at least
  // 50ns before driving INC low.  Not a problem at
  // the relatively low speed of 16mhz.
  WAH_POT_CHIP_SELECT = FALSE;

  // Increment given number of steps
  for (idx=0; idx<nMoves; idx++) {
```

```
    // Transition increment from high to low
    WAH_POT_INC = FALSE;

    // Must be low for minimum 50ns

    // Transition from low to high, must be high
    // for 100ns, since we're going to top of loop
    // that will take a few clock cycles which should
    // be well over 100ns wait period

    if (idx+1 < nMoves) {
      // don't transition high on last loop.  If
      // we do then the EEPROM will be programmed
      // when we set CS high
      WAH_POT_INC = TRUE;
    }
  }

  // End wiper move
  WAH_POT_CHIP_SELECT = TRUE;
}
```

Conclusion

The prototype version of the Magic Wah is proof that the product is technically feasible. The next phase of the project would be to productize the software as discussed in previous sections and to identify manufacturing facilities for the printed circuit boards and determine packaging requirements. Another alternative is to sell the idea to existing wah pedal manufacturers so that the technology can be incorporated into existing wah pedal brands.

The POT controller software will need to be enhanced slightly so that it does not “warble”. Currently the POT adjusts too frequently and causes a slight warbling sound. This can be easily remedied by filtering out frequent, small adjustments to the POT. Also, a digital POT with 256 increments would provide better resolution and are readily available.

The Codewarrior version that shipped with the SARD boards limits the code size to 16k. This limiting factor puts many requirements on the software design that would not be there if the full 64KByte memory space were available. If the Magic Wah were to be productized into a commercial application, the full 64KByte memory space would be utilized and many enhancements would be made, such as: low battery detect, watch dog (system operating properly), RF channel auto select, and many other features.