Application Report
SLAA358–April 2007

Using GMR Sensors With the MSP430 Scan Interface

Keith Quiring *MSP430 Applications*

ABSTRACT

The scan interface (SIF) of the MSP430F42x family of microcontrollers provides an innovative, low-power method to measure rotation that can be used with a variety of sensor types. Many sensor configurations generate a quadrature signalling sequence, but some do not. The SIF can accommodate a variety of input signals – not just quadrature.

This application report describes implementation of a rotation detector that uses a pair of giant magneto-resistive (GMR) sensors, which can detect magnetic fields. The sensors are positioned such that they do not use a quadrature signalling sequence, and this provides an opportunity to discuss how the SIF can be configured for non-quadrature situations. Techniques for debugging SIF applications are discussed, and a tool is provided for designing non-quadrature processing state machines.

Contents

List of Figures

List of Tables

1 Introduction

The scan interface (SIF) peripheral module available on the MSP430FW42x series of devices is an innovative way to measure rotation of a mechanism. Its advantages lie in its power efficiency and flexibility. Because it operates while the CPU sleeps, the device draws very low average current, often less than 50 μA (depending on the sensor used, maximum rotation speed to be detected, etc.).

In terms of flexibility, it can operate with a variety of sensor types. Though designed with rotation in mind, this is not a requirement; linear motion also can be measured. In fact, the SIF can be used in a wide variety of applications in which a group of one to four sensors generates a cyclical output stream.

This application report discusses use of the SIF with giant magneto-resistive (GMR) sensors, a relatively new type of solid-state device for detecting magnetic fields, which is replacing Hall Effect sensors in many applications. The project described in this application report uses a processing state machine (PSM) that is different from the quadrature PSM common in many SIF applications.

It is assumed that the reader has studied the SIF chapter in the user's guide and is familiar with the SIF's basic function and architecture.

2 Hardware

The hardware used for this project consists of a standard MSP430 64-pin FET board (available from TI's web site at www.ti.com) with a custom daughtercard attached. Figure 1 is a photograph of this setup.

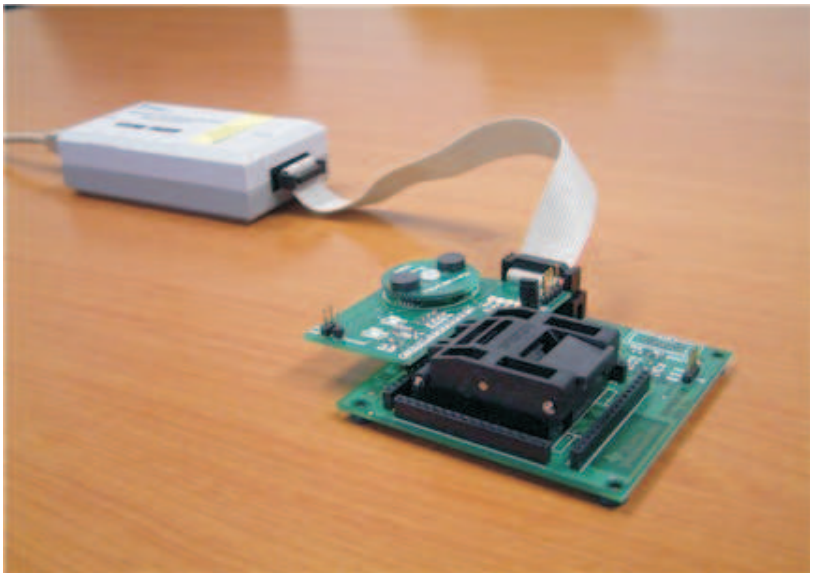


Figure 1. SIF/GMR Hardware

Figure 2 shows the hardware without the wheel, which reveals the sensor ICs underneath.

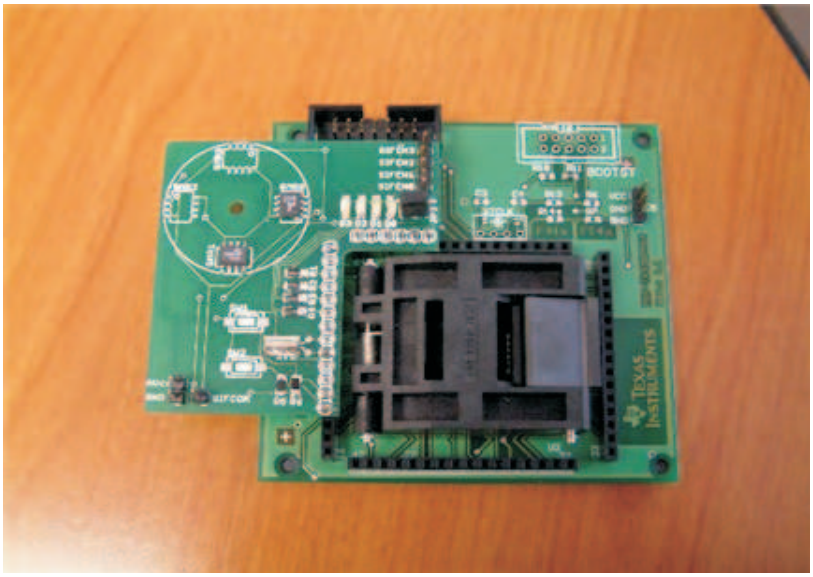


Figure 2. SIF/GMR Hardware Without Wheel

Figure 3 is a side view of the wheel apparatus and identifies its components.

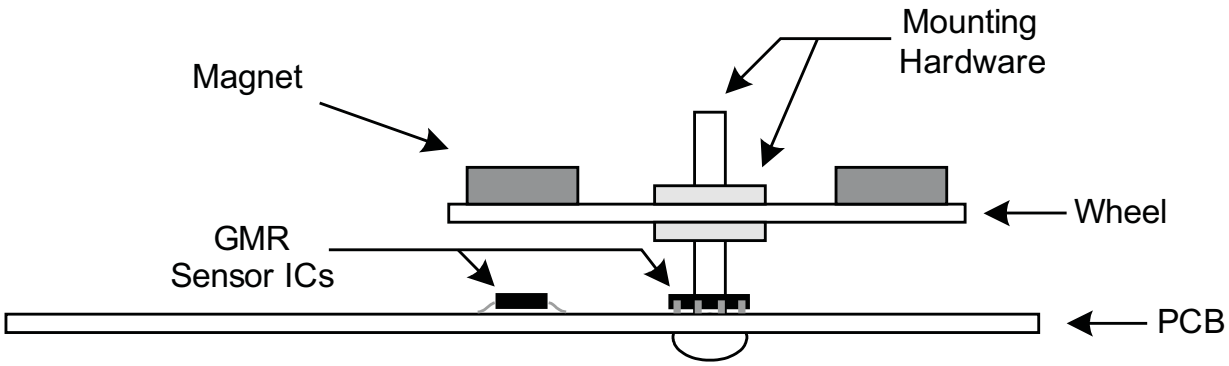


Figure 3. Wheel Apparatus

As can be seen in the photograph and sketch, a wheel fashioned out of standard FR4 PCB material has been mounted above the board plane using a nylon bolt and two nuts. This wheel is meant to be rotated by hand, and the MSP430 measures the rotation. On the topside of this wheel are mounted two small magnets. Beneath the wheel, on the daughtercard, are the GMR sensors. The sensors used in this project are AA002-02 devices, made by NVE. They are mounted in such a way as to optimize their sensitivity to the changing magnetic field. (Refer to the manufacturer's datasheet for more information.) The wheel mechanics are less precise than would be used in most production applications, but are sufficient to produce a reliable demonstration of the SIF's capabilities.

The schematic for the daughtercard can be found in Appendix A. Gerber files to reproduce this board can be found in the code files that accompany this document. Note that four GMR sensors are shown in the schematic, but only two are utilized by the code. In this project, the IC pads connected to SIFCH0 and SIFCH1 were populated, and these pads are oriented 90° to each other. However, these two sensors could have been placed in any two of the four pads. (The code would need to be adjusted to use the other channels.)

2.1 Magnet Selection and Configuration in Relation to the Sensors

The gap between the sensors and the magnets in this setup is approximately 3/16 inch (0.5 cm).

While a complete analysis of the interactions between the GMR sensor and various magnet types is outside the scope of this document, the far-reaching effect of magnet choice and configuration should be noted. For example, larger magnets produce larger fields, reducing the SIF's required sampling rate for a given maximum rotation speed, which saves power. In other words, if a magnet's field covers 10° of the wheel's angular space instead of only 5°, the sensor need only check for the presence of the field half as often (see Section 4).

Another example is that the magnet/sensor configuration – that is, the number of magnets and sensors used and where they are positioned – affects the processing state machine that will be used in the software implementation. This could have an impact on robustness. It also may affect the granularity, that is, the sensitivity to fractions of a full rotation rather than only detecting full or half rotation. If only one sensor and one magnet are used, only full rotations in one direction can be captured. If one sensor and two magnets at opposite sides of the wheel are used, half rotations in one direction can be measured.

2.2 GMR Sensor Configuration and Their Interface to the SIF

The particular GMR sensor chosen for this implementation uses a Wheatstone bridge configuration. One side of the bridge has a shield over the upper resistor that prevents it from detecting magnetism, while the other side has a shield over the lower resistor. If coupled with a differential operational amplifier, this configuration doubles the sensitivity of the sensor. An op amp is also useful in that the single-ended output of the sensor (that is, using only one side of the bridge) is relatively small; in this setup, it was approximately 100 mV, even with relatively close proximity to the magnet. Using the differential signal and adding a modest amount of amplification allows the signal to clearly differentiate between the presence and absence of the magnet.

This particular design, however, does not require an op amp. One side of the Wheatstone bridge (which by definition is a simple voltage divider) is attached directly to the SIFCHn bridge (SIFCH0 and SIFCH1, for sensors one and two, respectively). It was found that reliable operation could be obtained (under laboratory conditions) without differential amplification. Operation at higher temperatures, with weaker magnets, or with greater sensor/magnet distance may cause these additional measures to be required.

Figure 4 shows an oscilloscope plot of the output of one of the sensors as the magnet passes overhead.

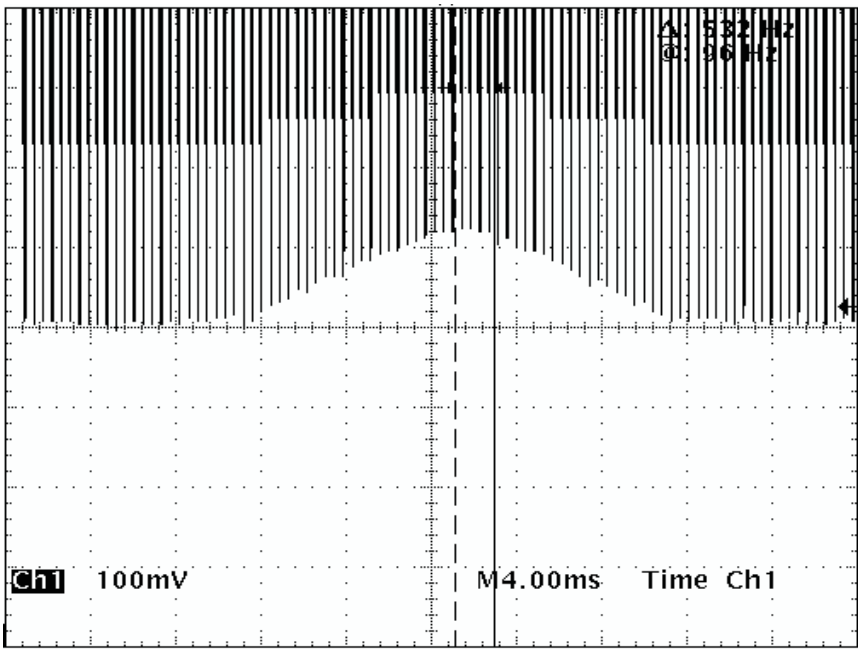


Figure 4. Sensor Output Responding to Magnet

Each pulse represents a time period in which the sensor is being activated (SIFCOM pulled low). The baseline voltage in Figure 4 is the level if no magnet is present. An approximate peak of 120-mV differential can be seen as the magnet passes over the sensor.

If functionality were the only concern, the voltage divider could be attached directly to V_{CC} and ground continuously, and the sensor output could be sampled as fast as possible. However, the design is also very concerned with minimizing power consumption. Remaining true to the MSP430's focus on ultra-low power, and considering that the SIF was designed with this same intent, it makes sense to consider how to configure the GMR setup so as to minimize power.

Rather than connecting the bottom of the voltage divider to ground, it is connected to SIFCOM. A potential configuration for SIFCOM is that it can be connected to ground (SIFVSS) when directed to do so by the SIF's timing state machine (TSM). When not connected to ground, it is in a high-impedance state. As such, the voltage divider does not draw a current when the SIF is sleeping between samples, which is to say it is not drawing a current most of the time. Since the current draw of each sensor is in excess of 0.5 mA, by far the largest consumer in this design, this is a very significant savings.

If an op amp were used to amplify the signal, it would be important to use one with a sleep mode input that could be driven from SIFCOM (active low, if possible). This would allow it, also, to be powered down when not in use. Fast startup time is beneficial in these situations, so that total sampling time can be kept to a minimum.

3 Software

The major benefit of the SIF is that, once configured and enabled, the CPU can be powered down, only waking when the count has reached a particular value or to handle errors. As such, rotation detection is delegated to the SIF.

A basic software implementation for the SIF is shown using pseudocode in Figure 5.

```
void main(void)
{
  configureTSMCal();            // Configure TSM to be used for calibration
  configurePSMCal();            // Configure PSM to be used for calibration
  configureSIFCTLCal();         // Configure control registers for calibration
  calibrateSIFDAC();            // Calibrate DAC levels in the SIF

  configureTSMMain();           // Configure TSM to be used for main operation
  configurePSMMain();           // Configure PSM to be used for main operation
  configureSIFCTLMain();        // Configure control registers
  enableSIF();                  // Set SIF_EN bit to enable SIF

  _BIS_SR(LPM3_bits + GIE);     // Sleep.
}

#pragma vector=SCANIF_VECTOR
__interrupt void ISR_SIF(void)
{
  if countEvent                 // If int generated by count event, handle it
    handleCount();

  if errorEvent                 // If int generated by error event, handle it
    handleError();
}
```

Figure 5. Basic SIF Code Structure

3.1 Calibration

3.1.1 Overview

The SIF's integrated DAC drives the threshold voltage used by the integrated comparator to determine whether the sensor voltage at SIFCHn is high or low. It is very important to find values for the DAC that consistently result in accurate assessment of the sensor's signals. Inaccurate assessments result in PSM errors, and either the CPU is forced to guess as to what happened (resulting in potential inaccuracies), or the user must get involved to decide how to handle it. Neither of these is desirable.

The scheme used for this project is relatively simple. A large number of samples is taken and analyzed while the user turns the wheel at a fast rate. An assumption is made that over the course of these samples, at least a few will represent values near the high and low extremes. A second assumption is that the average between these two extremes will be characteristic of the average of all signals. To quantify the samples, the DAC output is raised or lowered by the CPU until the comparator output is toggled. The CPU reads this result and keeps a record of the extreme highs and lows.

When this process is over, a check is implemented to verify that a minimum gap in the high/low extremes was detected. If that gap did not occur, it could indicate that the extremes were not truly found, and the calibration process is repeated. When the gap is sufficient, the highs and lows are averaged to find the mean threshold for each sensor, and hysteresis is added to produce the DAC0/1 and DAC2/3 pairs.

During the calibration process, LEDs inform the user of calibration progress. There are four LEDs; during calibration, they count upward in binary fashion. When the code has determined that acceptable DAC levels have been acquired, all four LEDs flash together several times. At this point, it is advisable to position the wheel such that neither sensor is in proximity to a magnet, as this the condition under which the PSM expects to begin.

With the scheme used for this project, the goal during calibration is to keep the comparator output as up-to-date as possible, not the minimization of power. As a result, the sampling rate is set to the lowest possible divider (fastest clock). A sample is taken every two cycles of ACLK.

Of course, this is not the only possible SIF calibration scheme. A myriad of options is available to the engineer. Some allow for periodic calibration during main operation, and this helps to compensate against temperature-induced changes or change over time in the sensor's ability to respond to the magnetic field. Involving a user in the calibration process may result in a better calibration, but requiring user involvement may be undesirable.

Calibration is easier if there is a wide difference between "high" (magnet present) and "low" (magnet not present). Here again, adding a differential op amp or using a magnet with a stronger field can increase sensitivity.

3.1.2 Control Register Settings

Table 1 shows the control registers, their settings during calibration, and commentary on why the settings were chosen.

Table 1. Control Register Settings During Calibration

Register	Comment
SIFCTL1	• SIFIFG1 is used to generate an interrupt after each TSM cycle completes, so that the DAC level can be changed. To enable/disable the interrupt, SIFIE1 is toggled during the calibration process. • SIF_EN bit is set to enable the SIF.
SIFCTL2	• This register contains override bits for the DAC, the comparator, and SIFVSS; but the software leaves these components under the control of the TSM. • Default comparator settings are chosen. • The mid-rail voltage generator and the excitation circuit, used as part of an LC sensor implementation, are not used and are therefore disabled. • The test function is not used. • The sample-and-hold function, however, is used when implementing resistive dividers such as GMR sensors, and is therefore enabled.
SIFCTL3	• The S1 input to the PSM is connected with the AFE's SIF0OUT signal (associated with SIFCH0). The S2 input is connected with the AFE's SIF1OUT signal. • No count mode is used during calibration and, therefore, the count-induced interrupts are not used. • No connections to Timer_A.
SIFCTL4	• No count modes used. • Q7/Q6 are not to be used in PSM state calculations. The state machine is not large enough to require this. • Fast TSM rate (ACLK/2).
SIFCTL5	• SMCLK is used as the high-frequency SIFCLK, so the internal oscillator is disabled. • TSM repeats periodically, not continuously.

Note that SMCLK is used as the high-frequency SIFCLK source. The internal oscillator could have been used, but there is little value in using the 4-MHz option in this application, and the 1-MHz option consumes about the same current as the DCO operating at the default 1.048-MHz frequency. (If the DCO were configured for a higher speed, thereby consuming more current, the SIF internal oscillator would be beneficial.)

The SIF has the ability to wake up the DCO; so, it is possible to configure the device for LPM3 without disabling the SIF. The SIF wakes up the DCO only when the TSM operates, and disables it when the TSM is complete. This means that instead of consuming a base current of approximately 92 μA when the TSM is not active, the device consumes a nominal 0.9 μA – a tremendous power savings. The extremely fast startup of the MSP430's DCO makes this possible.

3.1.3 TSM

The GMR TSM stays active for a relatively short time period of ten SMCLK periods. If SMCLK is left at the default frequency (1.048 MHz) with no divider, ten SMCLK periods is 10/1048000 or just under 10 µs. Table 2 shows the steps within the TSM and provides commentary on their function and duration.

Table 2. Timing State Machine Summary

State	Duration	Comments
SIFTSM0	1 SMCLK	Reset dummy cycle; required for any TSM.
SIFTSM1	3 SMCLK	Activate DAC and comparator for use with SIFCH0 input (sensor #1). Also drives SIFCOM low. The MSP430FW427 datasheet specifies a maximum of 2-µs settling time for these components. Two SMCLKs is slightly less than 2 µs. For this reason, and for design margin, three SMCLKs are used.
SIFTSM2	1 SMCLK	Comparator output latched to S1. Only one SMCLK cycle required.
SIFTSM3	3 SMCLK	Activate DAC and comparator for use with SIFCH1 input (sensor #2) . Also drives SIFCOM low.
SIFTSM4	1 SMCLK	Comparator output latched to S2. Only one SMCLK cycle required.
SIFTSM5	1 SMCLK	Stop cycle.

It is recommended to study the TSM values in the software in relation to the descriptions given in the user's guide.

3.1.4 PSM

The PSM is not used during calibration; instead, the CPU reads the comparator output directly. Still, there is no way to disable the PSM, so a "dummy" PSM is created and assigned to the SIFPSMV register.

3.2 Main Operation

3.2.1 Control Register Modifications

To a large extent, the control register settings remain the same for both calibration and main operation. Table 3 shows the modifications that are made.

Table 3. Control Register Modifications Upon Entering Main Operation

Register	Comment
SIFCTL1	Two interrupts are now used: • IFG3 alerts the CPU that the counter has been updated. (Works in conjunction with SIFIS1x in the SIFCTL3 register, which is left at its default setting.) • IFG5 alerts the CPU that an error occurred in the PSM.
SIFCTL4	Count modes are now used: • SIFCNT1 is configured for increment, but not decrement • SIFCNT2 is not enabled. • The sampling rate is lowered to ACLK/14.

See Section 3.2.4 for a discussion on sampling rate selection.

3.2.2 TSM

The TSM is exactly the same as the one used during calibration.

3.2.3 PSM

Once calibration is complete, DAC values are in place for assessing the sensor output. Rotating the wheel in a counter-clockwise direction presents an S2/S1 signal sequence of 00→10→00→01→00→10 (see Figure 6).

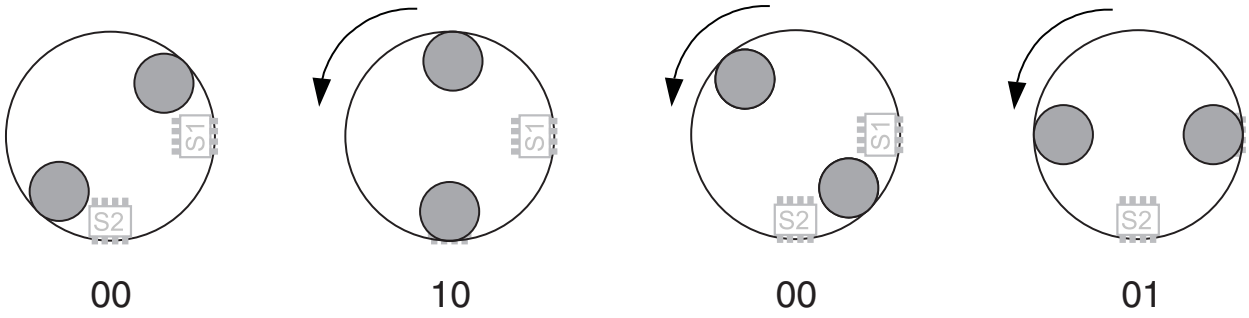


Figure 6. Sensor/Magnet Configurations Resulting in S2/S1 Values

This leads to a problem: there are two S2/S1 states of "00". A transition from "10" or "01" is always to a state of "00". This raises an important question. If the PSM sees a 10→00 transition, how does it know the direction in which the wheel moved? This is important, because it dictates whether the PSM should expect a "01" or "10" as the next S2/S1 state. The PSM could be written to make this decision once it had obtained that next sample. But, if this next sample were to be in error, the error would not be caught.

So, a decision must be made at this point whether to allow these kinds of errors to go undetected, or to make an assumption that all movement will persist in one direction or the other. This design has been implemented with the assumption that all movement is in the counter-clockwise direction. The mechanics of the implementation must ensure that reverse direction is not possible.

Another solution to this problem would have been to change the sensor/magnet configuration to reflect the quadrature sequence common in other SIF applications: 00→10→11→01→00→10. This could be accomplished by moving the magnets such that they are 90° apart from each other instead of 180°.

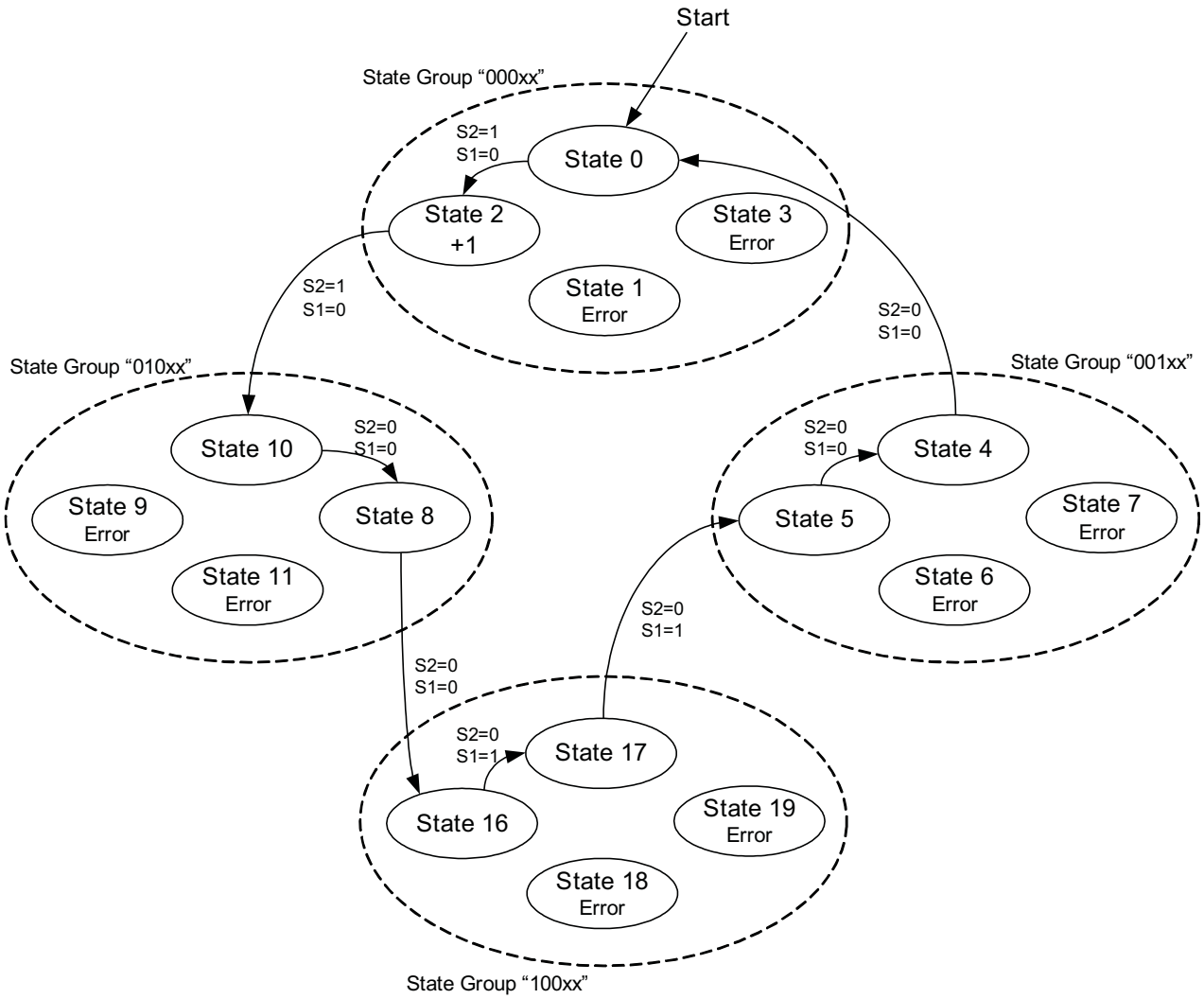


Figure 7. PSM State Machine Diagram

Note that there are now two types of "00": one which is entered upon leaving an S2/S1 state of "01", and one which is entered upon leaving an S2/S1 state of "10". Each has its own state group in PSM memory ("000xx" and "100xx" in Figure 7). One of the available address bits in the PSM data byte, Q4, is used to signal entry from State 8. (This is the "1" in state group "100xx".)

Table 4 represents the way in which the state machine diagram in Figure 7 is implemented in memory.

Table 4. PSM State Machine Entries

Address Offset	Event	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0	Hex	Next State Address (Q7–Q3, Q2, Q1)
00	No rotation	0	0	0	0	0	0	0	0	00	000000??
01	Error	0	1	0	0	0	0	0	1	41	010001??
02	Next step	0	0	0	0	1	0	1	0	0A	000010??
03	Error	0	1	0	0	1	0	0	1	49	010011??
04	Next step	0	0	0	0	0	0	0	0	00	000000??
05	No rotation	0	0	0	0	0	0	0	1	01	000001??
06	Error	0	1	0	0	1	0	0	0	48	010010??
07	Error	0	1	0	0	1	0	0	1	49	010011??
08	Next step	0	0	0	1	0	0	0	0	10	000100??
09	Error	0	1	0	0	0	0	0	1	41	010001??
0A	No rotation	0	0	0	0	1	0	0	0	08	000010??
0B	Error	0	1	0	0	1	0	0	1	49	010011??
0C	Error	0	1	0	0	0	0	0	0	40	010000??
0D	Error	0	1	0	0	0	0	0	1	41	010001??
0E	Error	0	1	0	0	1	0	0	0	48	010010??
0F	Error	0	1	0	0	1	0	0	1	49	010011??
10	No rotation	0	0	0	1	0	0	0	0	10	000100??
11	Next step	0	0	0	0	0	0	0	1	01	000001??
12	Error	0	1	0	0	1	0	0	0	48	010010??
13	Error	0	1	0	0	1	0	0	1	49	010011??
14	Error	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
15	Error	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
16	Error	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•
1F	Error	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

A basic quadrature PSM would only contain entries for addresses 0x00–0x0F. In this PSM, the 0x0C→0x0F state grouping is replaced with the one at 0x10→0x13. To be directed to these states, the previous S2/S1 pair must be "00". These, in fact, are the second "00" states. Note that state 0x08 has Q4 set; this directs it to 0x10 instead of 0x00.

Note that if an S2/S1 signal pair of "11" is encountered, it leads to an error state (0x03, 0x07, 0x0B, etc.). Likewise, states 0x0C→0x0F are considered error states because to arrive at these states, the previous S2/S1 pair would have had to be "11". These are all considered errors because "11" can never occur with the sensor/magnet configuration used in this project. (The magnets are positioned 180° apart from each other, while the sensors are 90° apart from each other.)

Starting at entry 0x00, suppose an S2/S1 pair of "10" is encountered. In calculating the next state, these substitute for the "??" at the far right of the row, resulting in a next state offset of 0x02. Suppose another sample of "10" is encountered; this sends the PSM to offset 0x0A. After a couple more samples of "10", each of which redirect back to 0x0A, a "00" is encountered. Substituting "00" into the "??" in the 0x0A row results in a next state of 0x08, and another sample of "00" results in 0x10. The pattern continues back to the original state entry of 0x00.

3.2.4 Sampling Rate

The "sampling rate" discussed in this document is the rate at which the TSM is triggered, resulting in a new S2/S1 pair being presented to the PSM. It is controlled by the SIFDIV3Bx/Ax fields in conjunction with the SIFDIV2x field in the SIFCTL4 register. Together, they select a divider against ACLK, and the resulting sub-clock drives the TSM. The divider values can range from ACLK/2 (typically 16.4 kHz) to ACLK/3600 (~9 Hz).

Generally speaking, the sampling rate required for a given application is a function of a) the maximum angular velocity of the mechanism and b) the angular width in which the magnetic field is strong enough to cause the comparator to indicate the presence of the field. The latter is, of course, also dependent on the chosen DAC value.

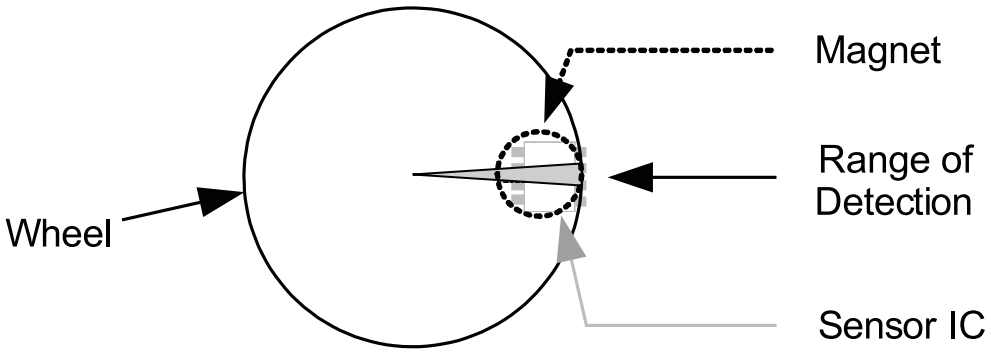


Figure 8. Angular Field Width

In the setup used for this project, the detectable magnetic field width is quite narrow. An empirical observation suggests an approximate width of 2° of the angular range. For a given angular velocity, the wider the field and the lower the sampling rate required to successfully detect it, and vice versa.

The angular velocity can be expressed in rotations/second, and the field width can be expressed in fractions of a rotation – in this case, $2/360 = 55.6 \times 10^{-4}$ rotations. Dividing the latter by the former produces a result with a unit of seconds; this is the amount of time the magnet/sensor combination produces a positive result at the comparator output at this speed.

Given this field width of 55.6×10^{-4} rotations, assume as an example that the maximum angular velocity is 10 rotations/second. Any sampling rate less than 1799 Hz has the possibility of missing the sensor window. Adding a 10% margin for error, a SIFDIV3Bx/Ax divider of 18 or less should be used.

4 Power Analysis

As mentioned earlier, using SIFCOM to activate/deactivate the sensor has great benefits in reducing power. Using the example of the previous section, suppose a divider of ACLK/18 is used in generating a TSM trigger frequency. This produces a sampling rate of approximately 1820 Hz. Also assume a TSM cycle that requires 10 μs to execute. The sensor now has an active duty cycle of 1.8%, reducing average current consumption per sensor from 600 μA (assuming 3.3-V supply) to 10 μA.

Table 5 shows each SIF TSM state, its corresponding theoretical power draw, and its duration. A theoretical average current draw can then be calculated.

Table 5. Theoretical Power Draw(1)(2)(3)

State	Duration (μs)	CPU/Clocks (μA)	DAC (μA)	Comparator (μA)	Sensor (μA)	Total Current (μA)	μA-μs
Idle	417	0.9	0	0	0	0.9	375
SIFTSM0	1	92	0	0	0	92	92
SIFTSM1	3	92	33	35	1200	1251	4080
SIFTSM2	1	92	33	35	1200	1251	1360
SIFTSM3	3	92	33	35	1200	1251	4080
SIFTSM4	1	92	33	35	1200	1251	1360
SIFTSM5	1	92	0	0	0	92	92
						Average Current/Sample:	26.8 μA

(1) Typical values, 3-V AV_{CC} assumed
(2) 2340-Hz sampling assumed
(3) SMCLK cycles rounded to 1 μs

The total number of microamp-microseconds is divided by the number of microseconds in a cycle, and this is the average current (in μA) per sample (TSM cycle plus down time). Since this cycle is repeated indefinitely, the average remains true for operation as a whole.

An additional amount is consumed when the CPU wakes up to handle a count update. It may seem that this would add a significant amount to the overall power consumption, but because of the duty cycle with which this occurs, the amount is fairly insignificant. Suppose the CPU wakes up every four increments of SIFCNT1, and stays awake for 200 MCLK cycles to handle the event. Also suppose that MCLK = SMCLK = 1.048 MHz, and that the angular velocity is a continuous 10 rotations/second, with a two-magnet scheme like the one in this project. This means that SIFCNT1 is incremented 20 times per second, and the CPU must wake up five times in that second. At 1.048 MHz, 200 MCLK cycles requires 190 μs, and doing this five times a second means the CPU is running just under 1 ms every second, a duty cycle of under 0.1%. In active mode with V_{CC} at 3 V, the device consumes 300 μA; so, the periodic CPU wakeup places an additional 0.3-μA of load on the device's power draw. Obviously this amount is affected by the SIFCNT interrupt settings (waking up every 1, 4, or 64 counts), as well as the choice of using two magnets instead of one.

This list summarizes all the actions can be taken to reduce power draw in a SIF GMR application. Note that these actions may be part of a tradeoff that impacts performance in other areas, several of which have been discussed elsewhere in this document.

- Reduce maximum angular velocity, allowing sample rate to decrease
- Use a stronger magnet, increasing sensor window and allowing sample rate to decrease
- Use a differential op amp (an additional source of power drain, but may increase sensor window and therefore reduce the sampling rate)
- Use sensor with lower current draw
- Use fewer sensors
- Minimize the time in which the TSM is active and the time in which the DAC and comparator are active within the TSM
- Decrease the frequency with which the CPU wakes up to handle the count

5 Debugging

Because of its autonomous nature, operating as it does without CPU intervention, the SIF can present challenges when debugging. The TSM's ability to drive the sensor properly can be verified by probing the sensor output. But once this is validated, most of the remainder of the operation is internal to the SIF. Debugging often relies on a combination of the SIFDEBUG register, experimentation, and a solid understanding of how the SIF operates.

If it is detected that something has gone wrong (for example, the SIFCNT registers are not counting), the PSM is usually the best way to trace what happened leading up to the problem. This data is available in the SIFDEBUG register. The following sections describes two tools that can be used to trace PSM function.

5.1 PSM Tracking Function

Simply checking the SIFDEBUG register manually within the debug environment after the error occurred at most shows that the PSM is now in an error state. To determine anything from a PSM trace, it is necessary to capture the series of states that led to the error. As with any kind of signal tracing, it is helpful to get data over a large time period but be able to quickly locate the error.

A PSM tracking function is included in the main function of the accompanying code file, and is shown for reference below.

```
SIFDEBUG = 0x0000;
while(1)
{
  w = v;
  v = (char) SIFDEBUG & 0x00FF;

  if(!(v==w))
  {
    stateList[stateListIndex++] = v;
    if (stateListIndex > 0x2FF)        // In case stateList[] runs past 0x2FF
      stateListIndex = 0;
  }
}
```

When active, the function executes continually in the main function after the SIF is initiated. It loops, checking the SIFDEBUG register. If it finds a value different from the previous value, it records the new value to memory. In so doing, it records the progression of states in the *stateList[]* array, eliminating the duplicate samples.

Because the SIF interrupt is enabled, this tracking function is interrupted periodically. Theoretically this could lead to missed states, but this is unlikely since the ISR operation is brief, and in most cases there are multiple samples of the same S2/S1 pair leading to multiple subsequent occurrences of a given state, increasing the chance of it being captured.

As obtained from TI, the line of code sending the device into LPM3 after enabling of the SIF is uncommented. This means the PSM tracking function is never reached. If the LPM3 line is commented out of the code, this activates the PSM tracking function. If power consumption is being measured, it is necessary to uncomment the LPM3 line of code so that the CPU does not remain active during the measurement.

5.2 PSM Spreadsheet

An Excel® spreadsheet is included with this project that can be used for design and analysis of most SIF PSMs not exceeding 32 state table entries. The spreadsheet can be used for two basic functions: design and manual analysis of state changes, and PSM simulation. The PSM simulator inputs a series of proposed S2/S1 values and executes them against a defined PSM, showing the state changes that would occur.

The spreadsheet is equipped with help comments. Cells with a red triangle in the corner (per Excel convention) contain comments.

5.2.1 Design and Manual Analysis

This portion of the spreadsheet assists in organizing state information and producing a finished PSM. It also assists in manually tracing state transitions within the proposed PSM, based on S2/S1 signal pairs. It does not eliminate the need for a good understanding of the PSM, but it can greatly assist in learning the PSM and visualizing a given state machine.

Cells not shaded in gray can be edited, while those in gray are calculated according to how the PSM operates. Basic information that must be entered for each state are:

- A text label for the state
- Indicate if the state is an error state (Q6)
- Indicate if a transition to this state should increment/decrement the counter (Q2/Q1)
- Configure address bits not dictated by the above factors (Q7, Q5, Q4)

An excerpt from this portion of the spreadsheet is shown in Figure 9 for easy reference.

Table Index	Rotation/ Position Event	State Actions					PSM State Table Entry									Next State Address								PSM in Memory		
		Q6	Q2	Q1	Q3	Q0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0	Hex	Q7	Q6	Q5	Q4	Q3	Q0	Q2	Q1	PSM Addr:		82
		Err	-1	1	Current																			Offset	Abs	Data
00	No rotation	0	0	0	0	0	0	0	0	0	0	0	0	0	00	0	0	0	0	0	0	?	?	00	82	00
01	Error	1	0	0	0	1	0	1	0	0	0	0	0	1	41	0	1	0	0	0	1	?	?	01	83	41
02	Next step	0	0	1	1	0	0	0	0	0	1	0	1	0	0A	0	0	0	0	1	0	?	?	02	84	0A
03	Error	1	0	0	1	1	0	1	0	0	1	0	0	1	49	0	1	0	0	1	1	?	?	03	85	49
04	Next step	0	0	0	0	0	0	0	0	0	0	0	0	0	00	0	0	0	0	0	0	?	?	04	86	00
05	No rotation	0	0	0	0	1	0	0	0	0	0	0	0	1	01	0	0	0	0	0	1	?	?	05	87	01
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figure 9. PSM Spreadsheet (Design and Manual Analysis)

If Q6 is set for a given state, the entire row is cast in red. This becomes useful during analysis when tracking errors.

If the PSM only has 16 entries, all the Q7, Q5, and Q4 bits consist of zeroes. This is the easiest configuration. Setting the other fields still requires a good understanding of PSM operation and also a good state diagram from which to source the transition sequences. Once these are entered, the PSM is automatically generated in the column entitled "Hex". This can be directly transferred to the MSP430 code.

Setting the Q4 bit in any of the state entries has the effect of opening up another 16 entries; or, at least adding enough entries to extend to the address to which that entry now points.

The set of columns labelled "Next State Address" can be used for manual analysis of state transitions. This column represents the reordering of bits the PSM performs when calculating the next state. The "??" represents the new S2/S1 values that have been presented to the SIF by the AFE. To find the resulting next state, mentally replace the "??" with the new S2/S1 values and convert the resulting binary to hex. This is the new PSM offset. Use this value in the "Table Index" column to locate the next state.

The "PSM in Memory" column set represents three different perspectives of each PSM entry. The first is the index or offset against the base PSM address in the SIFPSMV register. The second is the absolute memory address for this entry, which is the sum of the offset and the base address. (For this column to calculate properly, the base address must be entered into the field at the top of this column set.) The last is the data located at that address, which is simply a copy of the "Hex" column. The next section describes a technique that uses this column set and is very useful in SIF debugging.

5.2.2 Combining the PSM Tracking Function With the Spreadsheet

Once a PSM is entered into the spreadsheet, the data placed into the *stateList[]* array by the PSM Tracking Function can be used with the "PSM in Memory" column set to trace operation and determine how errors occurred.

First, it is necessary to update the cell "Base PSM Addr" according to the value of the SIFPSMV register. Note that this pointer can change anytime the code is modified, since its location is assigned at compile time. This cell only contains the last two hex digits of the address, omitting the first two.

Next, run the SIF application. If it is only desired to track normal state transition flow, the code can be stopped at any time. If errors are to be located, ensure that the SIFIE5 bit in the SIFCTL1 register is set prior to operation, and include code like the following:

```
#pragma vector=SCANIF_VECTOR
  __interrupt void ISR_ScanIF(void)
{
  .
  .
  .

  if(SIFCTL1&SIFIFG5)
    SIFCTL1 &= ~SIFIFG5;                    // Breakpoint here to catch errors
}
Set a breakpoint on the line that clears SIFIFG5.  This code will only be executed
if an error state is encountered.
```

Set a breakpoint on the line that clears SIFIFG5. This code is only executed if an error state is encountered.

When execution is stopped, *stateList[]* will contain a list of state transitions (assuming they occurred). Assuming execution is stopped before the array was filled, the end of this trace is easily identifiable because unwritten locations in the array have a value of "FF". SIFDEBUG contains the very last value that likely has not yet been written to *stateList[]*. Start with the first value in the array and compare it against the values in the "Absolute" column. With this, the first state can be located. On this row can be found the state index (the table offset) and the data in the row. If the state is an error state, it is flagged red. Considering the S2/S1 signals that probably occurred next, manually trace to the next probable PSM state, and compare that against what actually occurred in *stateList[]*.

If an error occurred, the errant state is probably either in SIFDEBUG or toward the end of the *stateList[]* array. Locate the first errant state and compare it against the non-errant state just prior to it. The S2/S1 pair that led to the error can be clearly identified. This may provide clues as to the root cause of the error – for example, sampling rate was not fast enough to catch the sensor window, or DAC values were not properly set.

If the code runs for a long enough period of time, the array fills and starts at the beginning again. While the data is still accurate, it is harder to locate the last stored value once this has occurred, since values after the pointer no longer contain "FF" but rather contain former data. The function could rewrite "FF" before rolling over the pointer, but it would likely take longer than one TSM cycle, and therefore SIF data could be lost.

5.2.3 PSM Simulator

The PSM Simulator can be used to analyze what will happen if a given sequence of S2/S1 pairs occur. It is useful during PSM design to ensure proper operation during "perfect" sequences of S2/S1, and also to check robustness during possible "problem" sequences – for example, if a "bounce" condition occurs at the sensor. (A bounce is the momentary appearance of a change in direction when the wheel has not changed direction, resulting from the sensor being on the boundary of two field states. It is analogous to the bounce that occurs on mechanical switches.)

To use the simulator, first enter a complete PSM into the left side of the spreadsheet, as discussed in Section 5.2. Enter an initial PSM state table index. Then, enter S2/S1 signal pairs ("00"," "01", "10", or "11") that could theoretically occur in the sensor/magnet configuration. The resulting transitions will be shown in the column "Next PSM Index" (binary and hex). If a transition to an error state occurs, it is flagged in red. In such a way, errors or vulnerabilities in the PSM can be identified.

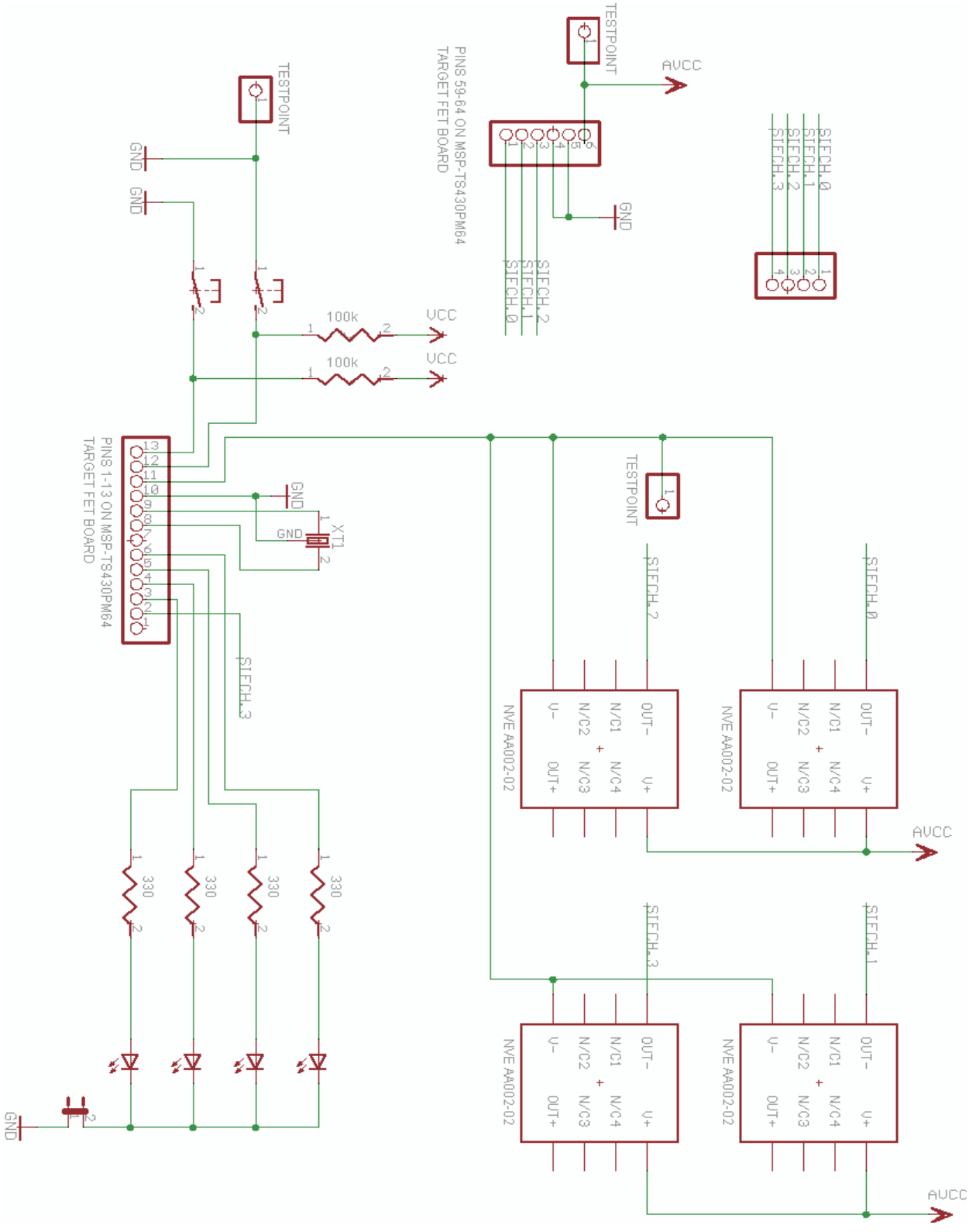
The PSM Simulator is shown in Figure 10.

PSM Simulator				
PSM Table Data		State Transitions		
Offset	First Six Bits	S2/S1	Next PSM Index (binary)	Next PSM Index (hex)
00	000000		00000000	00
01	010001	00		
02	000010		00000000	00
03	010011	00		
04	000000		00000000	00
05	000001	10		
06	010010		00000010	02
07	010011	10		
08	000100		00001010	0A
09	010001	00		
0A	000010		00001000	08
0B	010011	00		
0C	010000		00010000	10
0D	010001	00		
0E	010010		00010000	10
0F	010011	01		
10	000100		00010001	11
11	000001	01		
12	010010		00000101	05
13	010011	01		
14	010000		00000101	05
15	010001	00		
16	010010		00000100	04
17	010011	11		
18	010000		00000011	03
19	010001	01		
1A	010010		01001101	4D
1B	010011	00		
1C	010000		01000100	44

Figure 10. PSM Simulator

The left columns are a recompilation of PSM data entered in the previous portion of the spreadsheet. The entries in red represent error states. The column marked "S2/S1" contains the data arriving from the sensors. It can be seen that near the bottom, after receiving a "11" from the sensors, an error state results, turning the PSM column red. Error states continue to result from this first one.

Appendix A Schematic



IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265