Applying Micro-Electro-Mechanical Systems (MEMS) Technology To Digital Flight Controls

(Preliminary, Revision 1)

by Richard Moscatiello, October 28, 2009

Table of Contents

Table of Figures

Applying Micro-Electro-Mechanical Systems (MEMS) Technology To Digital Flight Controls (Preliminary, Revision 1)

By Richard Moscatiello

Abstract

This paper discusses the possibility of replacing existing aircraft gyroscope technologies (e.g. Laser gyros, spin gyros) with silicon-based micro-sensors. The author proposes that micro-accelerometers, arrayed along flight axes and processed using statistical, analog summing, or FFT methods, can be a viable substitute for existing gyroscope technologies. The obvious advantages of such an approach are reduced cost and reduced weight of avionics. A MEMS-based gyro module with a "fit-and-function" interface could be designed as a direct replacement for existing gyro installations, precluding redesign of aircraft wiring harnesses or related avionics. This paper will cover basic MEMS theory, a [proprietary] application method of MEMS devices to the gyro implementation, MEMS Gyro system LRU concept, preliminary individual circuit card design (to as much detail possible at this time), and fundamental data processing concepts to inform software design.

Discussion

"Micro-Electro-Mechanical Systems (MEMS) is the integration of mechanical elements, sensors, actuators, and electronics on a common silicon substrate through microfabrication technology. While the electronics are fabricated using integrated circuit (IC) process sequences (e.g., CMOS, Bipolar, or BICMOS processes), the micromechanical components are fabricated using compatible "micromachining" processes [see Figure 1] that selectively etch away parts of the silicon wafer or add new structural layers to form the mechanical and electromechanical devices." [1]

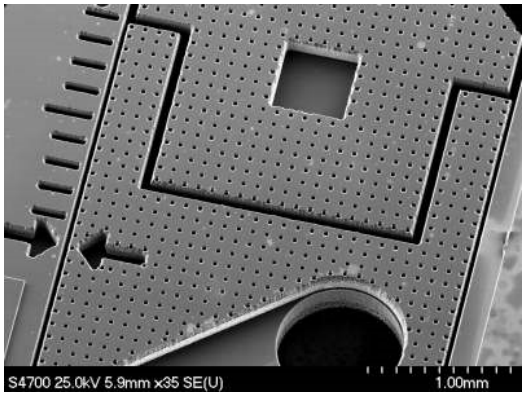


Figure 1. Example of MEMS device [2]

The overview concept for the MEMS Gyro Module is to create a Line Replaceable Unit (LRU) that will provide the Digital Flight Control Computer with preprocessed rate data that predict the aircraft's attitude. Figure 2 is a simplified illustration of that concept.

[1] Anonymous, http://www.memsnet.org/mems/what-is.html

[2] Ibid

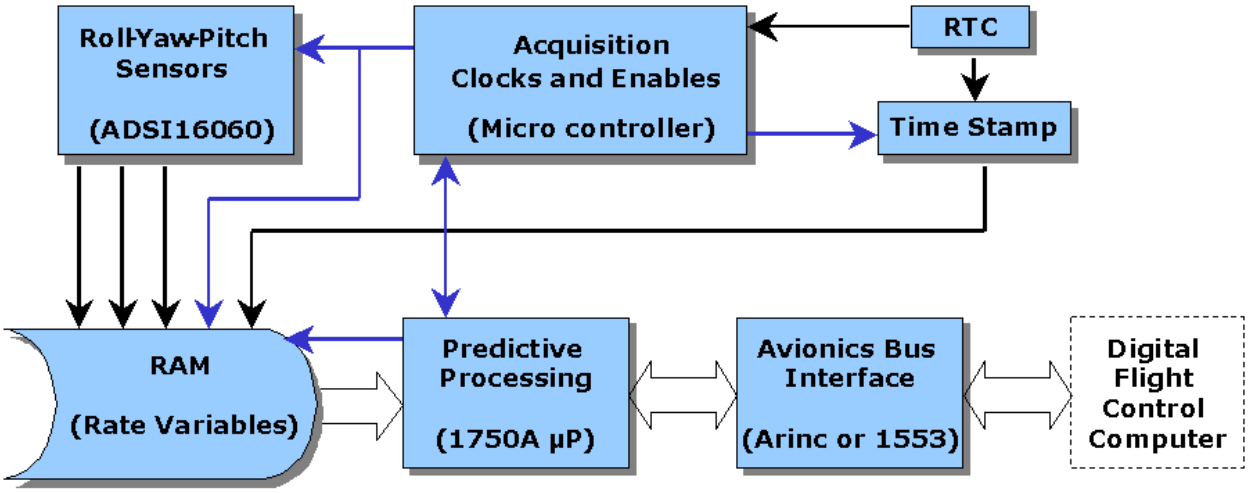


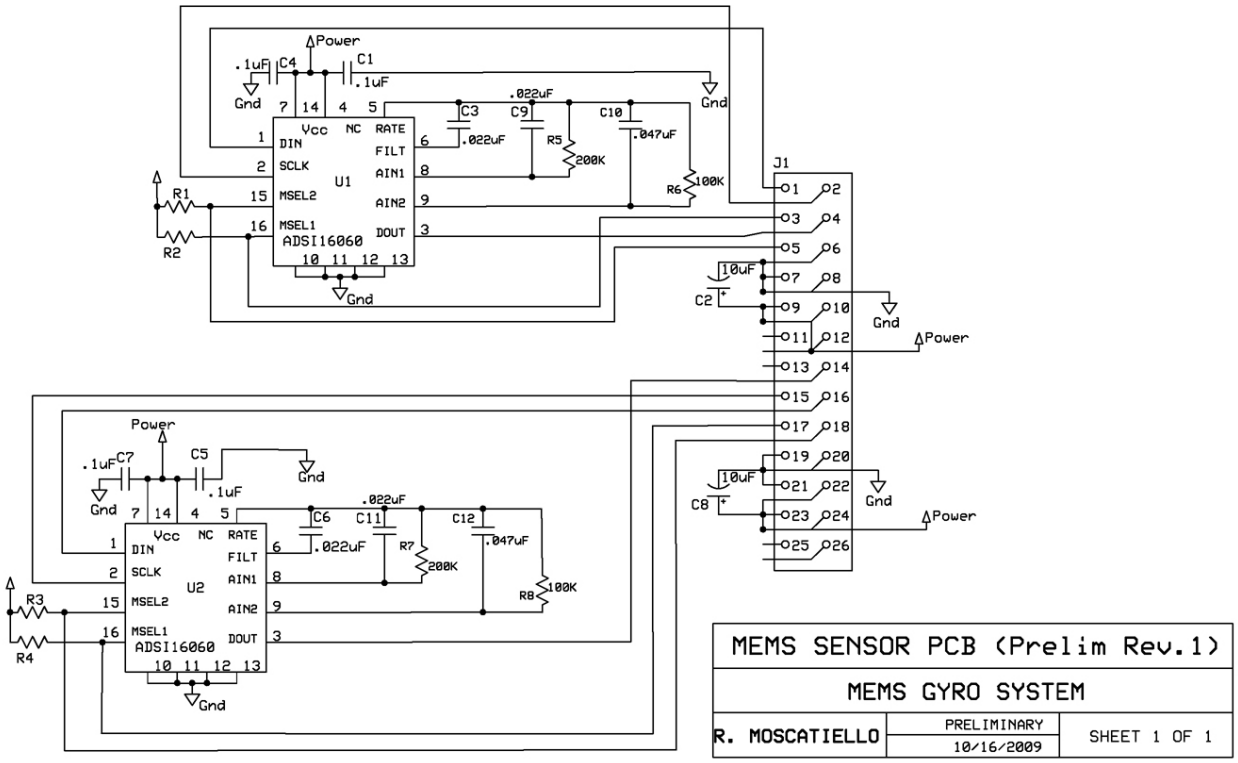
Figure 2. MEMS Gyro Simplified Block Diagram

In the design of a MEMS Gyro there are several factors to consider:

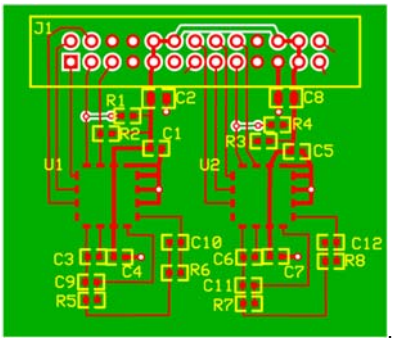
- Processing time:
  - The devices and software must be able to process attitudinal updates at a rate equal to or better than the update rate of fly-by-wire control surfaces.
  - Attitudinal updates must be predictive to compensate for processing and overhead lag time. In effect, given the current attitudinal position of the aircraft, the software must predict where it will be by the time the data are received and processed by the fly-by-wire system.
- MEMS accelerometer accuracy:
  - MEMS have not previously been applied to flight gyros because they are, at this time, not as accurate as ring laser gyros. Also, range of measurement is limited. A range of ±400°/s would be appropriate for an aerobatic aircraft. But in its basic configuration the range of the Analog Devices ADSI16060 is ±50°/s. With the addition of external components (per Application Note AN-942, see Appendix) the MEMS device can be configured for measurements up to ±200°/s.
  - Because the MEMS are inexpensive and small, it would be possible to place more than one on a PCB to measure a single axis. The combined outputs would then represent a data point distribution that can be filtered to a high probability of accuracy using methods such as Least Squares Regression.
  - Using more than one MEMS per axis has the advantage of precluding any single point of failure. A single failed MEMS device would result in degradation of accuracy but not loss of control.

MEMS Sensor PCB

A preliminary MEMS Gyro Module sensor PCB is shown in Figure 3.



Note: MEMS circuit configuration by Mark Looney as detailed in *Optimizing MEMS Gyroscope Performance with Digital Control*, AN-942 ©Analog Devices Inc. 2008



PCB design by R. Moscatiello, dimensions approximately 2" x 1.5"

Figure 3. MEMS Sensor PCB

The three MEMS Sensor PCBs will be installed in a module that has card guides and back-plane connectors positioned parallel to each rotational plane. The module is installed in a MEMS Gyro Line Replaceable Unit (LRU). In addition to the MEMS Gyro Module, the LRU will have three MEMS Interface PCBs, a 1750 Processor PCB, an Avionics Bus Interface PCB, a Low Voltage Power Supply to convert aircraft power, a chassis electrical back-plane for interconnection, and aircraft connection wiring harness. (See Figure 5 next page.)

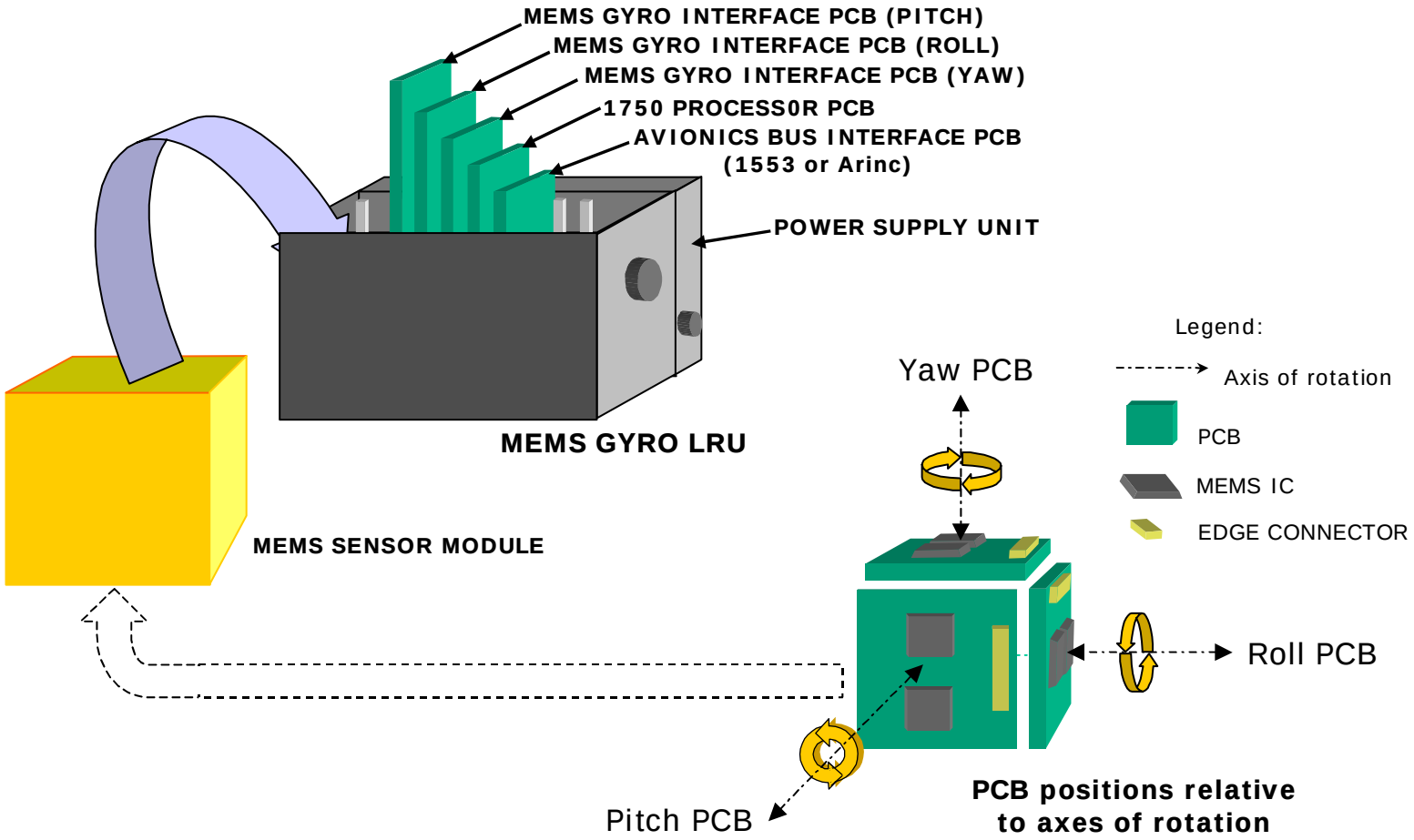


Figure 5. MEMS Gyro Line ReplaceableUnit (LRU)

MEMS Gyro Interface PCB

The purpose of the MEMS Gyro Interface PCB is to collect, store the data generated by the MEMS Gyro Sensor PCBs, and process it. Where data processing tasks are distributed in a flight control system largely depends on bus overheads and processing capacity. One approach is for the MEMS Gyro Module hardware to be restricted to supplying raw data (3 axes of unprocessed time-stamped accelerations) over the avionics bus (1553 or Arinc) to the Digital Flight Control Computer (DFCC). The obvious drawback of that approach is twofold: 1) the large amount of data uses up 1553 bandwidth, potentially overwhelming capacity, and 2) the DFCC will be required to process the data serially, placing loads on processor time.

A better approach is for the MEMS Gyro Module to process the raw data prior to transmitting it to the DFCC. The data transmitted to the DFCC need only be the polynomial function coefficients that describe the aircraft's path with *t=time* as the independent variable. That allows the DFCC to be dedicated to the more pressing task of calculating control surface movements based on the MEMS' function timeline, flight environment data (e.g. airspeed), and stick inputs.

There will be 3 MEMS Interface PCBs, one for each of the 3 (pitch, roll, yaw) MEMS Gyro PCBs. Each Interface PCB circuit will incorporate serial-to-parallel registers, real time clock, SRAM, card-edge data/address transceivers, and a micro-controller to provide clocks, timing, and enables.

On each Interface PCB the MEMS serial data are shifted to parallel and stored in SRAM. Each SRAM address location contains two 14-bit MEMS data sets and time stamps. Data will be written to the SRAM in time-ordinal sequential addresses and will be read back on a last-in-first-out basis for processing. Axial data sets would be acquired 100 times per second, and each data set would be blocks of one hundred sequential 14-bit MEMS readings. Each block of readings would contain 100 total MEMS sequential 14-bit data points and would take approximately 1ms, leaving 9 ms per cycle to process the fit function. This is illustrated in the timing diagram, Figure 6.

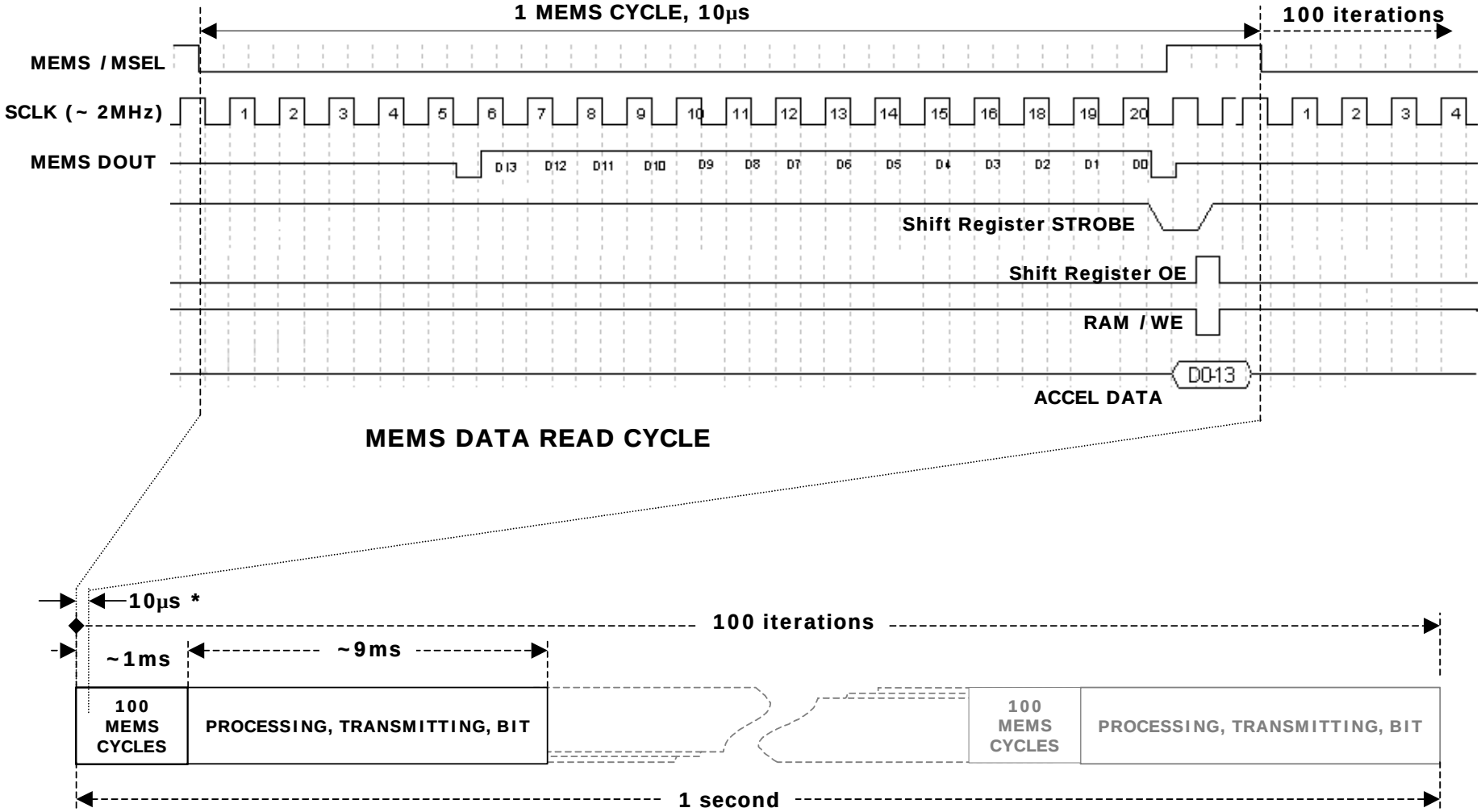


Figure 6. MEMS Timing Diagram

The two axial MEMS devices are to be read alternately, so there will be 50 measurements from each MEMS device (100 total) in a 1ms MEMS cycle. (See Figures 7, 8, 9.) The polynomial fit function processing will be performed on a separate 1750A Processor PCB connected via backplane (e.g. VMEbus). This will preclude the need for pre-processing on the MEMS Interface PCB, eliminating a microprocessor. The 1750A Processor will access the SRAM on the three MEMS Interface PCBs and compute the fit polynomial coefficients.

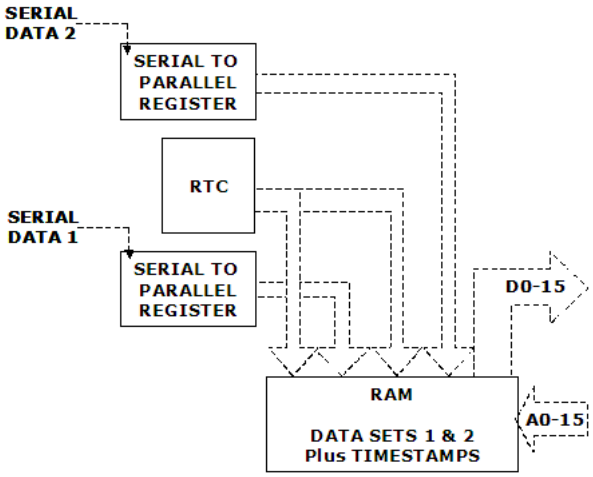
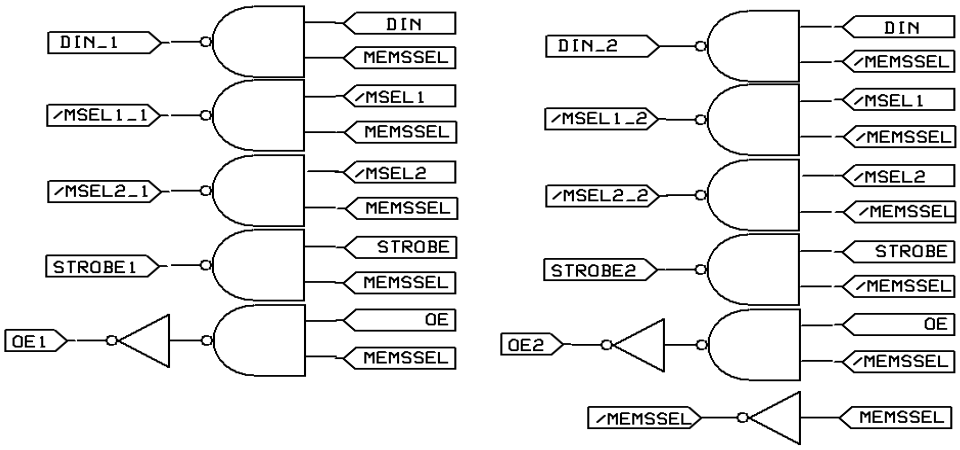


Figure 7. MEMS Data Read Block Diagram



Note: this circuit detail incorporated into CY37064 PLD

Figure 8. MEMS Selection Alternation Circuit Detail

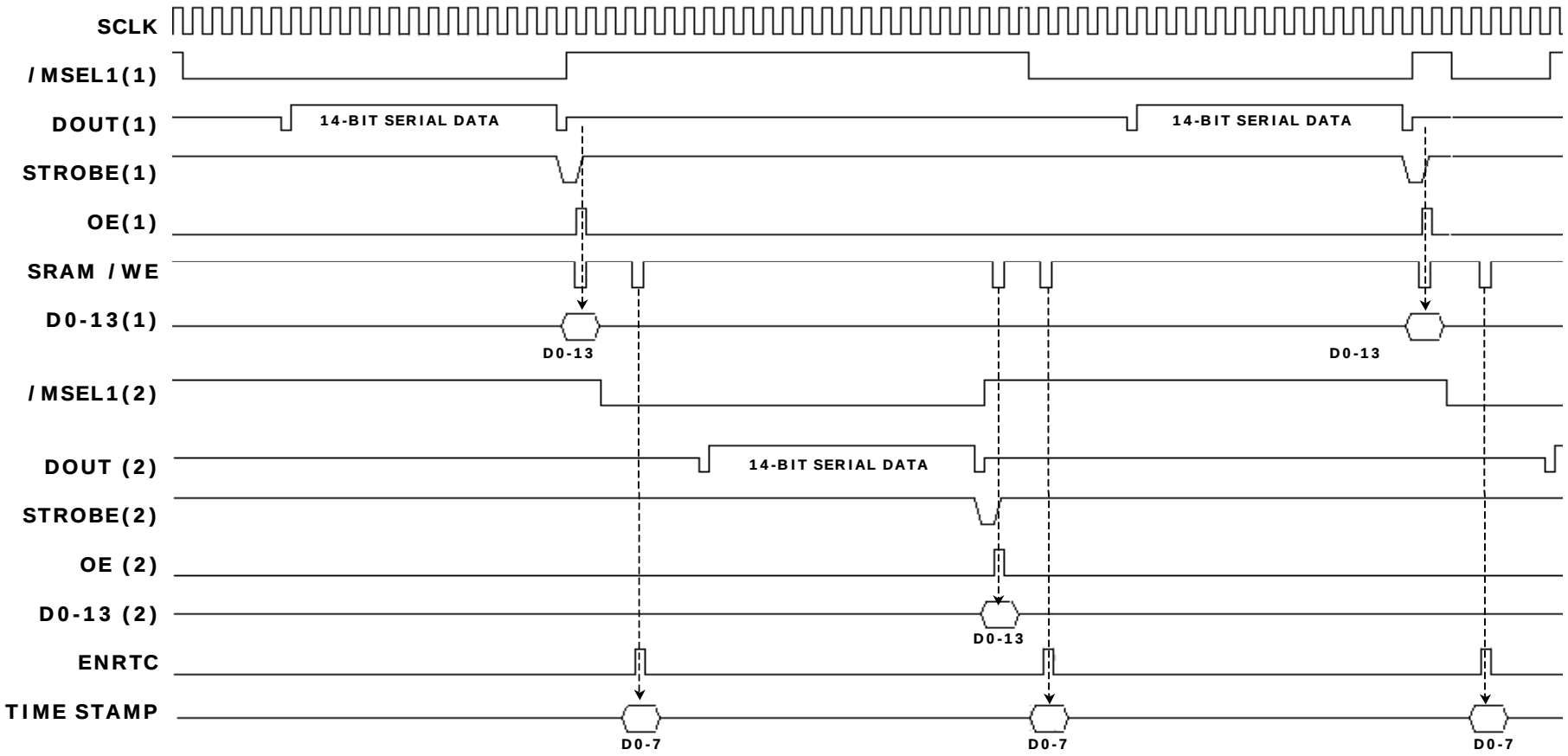


Figure 9. Alternating MEMS Cycle Timing

A Cypress CY37064 PLD (64 I/O pins) was chosen to control addressing, timing, chip enables for running the MEMS Data Read Cycle, and arbitrating 1750A access to SRAM data over the system bus. (See Figure 10.) The Cypress WARP design tool facilitates graphically assisted logic design and programming of the PLD device.

Functions assigned to CY37064:

- Provides clocks and enables to operate MEMS data processing cycles
- Provides sequential addresses for writing MEMS data into the SRAM.
- Connects 1750 Processor signals SERDATA and SERCLK to 1) command MEMS self-test and mode of operation, and 2) perform CY37064's JTAG test.
- Issues an interrupt request to 1750 at the end of each 1ms MEMS cycle.
- memory map and I/O port decode.
- 1750 bus arbitration.

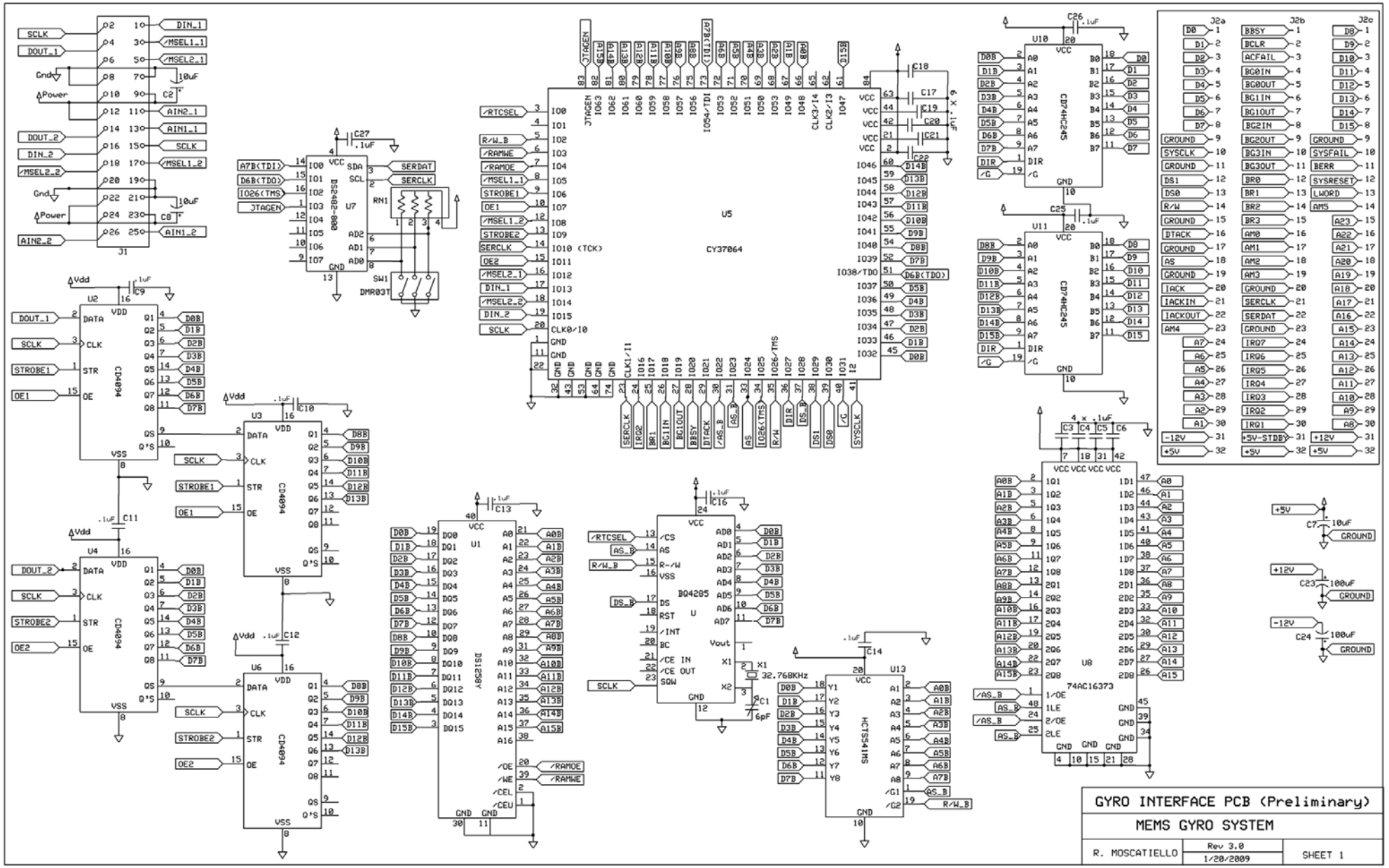


Figure 10. MEMS Interface Circuit Schematic

1750A Processor PCB, Avionics Bus Interface PCB, & Chassis Electrical Hardware

A 1750A small board computer would satisfy the processing and dimensional vision for the LRU. This is a "make or buy?" decision. Since 1750A is a standard architecture (see Figure 14), commercial off-the-shelf (COTS) Processor Boards designed for aerospace applications are available. BAE Systems' RAD750 has a 3U VME format that would fit well into the LRU's expected dimensions. COTS VMEbus backplanes are also available (e.g., Elma Bustronic) that would make prototyping easy. The same is also true for the Avionics Bus Interface PCB. COTS 1553B and Arinc interface PCBs are available that are virtually plug-and-play on a VMEbus. Since the most critical part of the MEMS Gyro is the processing software, using COTS hardware makes the most sense (Figure 11).

BAE Systems RAD750 | **Elma Bustronic VME backplane** | **1553-3U**

Figure 11. COTS Hardware

Processing Software

The output of the MEMS Gyro System will be the coefficients of a polynomial that describe a 3-axis acceleration function. The function's independent variable will be time, and the function will be used to predict the near future location of the platform based on near-real-time sampling. Method is to collect 3 data sets (x,y,z axes) of 100 time-stamped axial velocities during a 1ms window every 10ms. Each data set will be processed as vector product of the rate of change of the axial velocities. Then the last 100 vector products taken during the last 1-second period will be processed as the polynomial using a least squares regression method. The processing will be recursive in the sense that each successive polynomial computation will drop the earliest vector product, always using the most recent 100 vector products so that the future prediction is constantly evolving with changing platform accelerations.

To understand the output from the MEMS, let the MEMS device spin-up from 0°/s at t=0 to 25°/s at t=1 with a constant acceleration. During the period t=0 to t=1 the MEMS measurements will ramp from 0°/s to 25°/s as a linear change in velocity. However, acceleration would be 25°/s/s. Now at t=(1+n) let acceleration stop but let the MEMS continue to rotate at a constant velocity of 25°/s. In that case the MEMS would measure a velocity of 25°/s at any point t=1 to t=(1+n) and acceleration would be 0°/s/s, as illustrated in Figure 12.

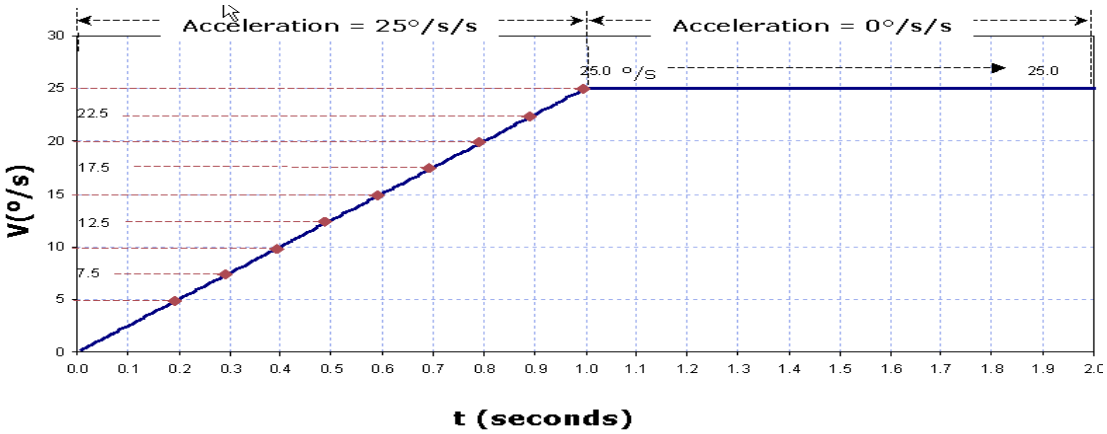


Figure 12. Velocity over time

Average rotational acceleration between two points of angular velocity can be expressed as:

$a = \frac{v_2 - v_1}{t_2 - t_1}$. In practice any two sequential MEMS samples divided by 10µs can be considered an instantaneous acceleration.

There are two sets of data stored in the SRAM: (1) the set of 3 times 100 axial data points that occurred in the last 1ms, and (2) the 3 sets of data points that are each a summation of the last (100) 1ms readings. So there is no confusion, they will hereafter be referred to as (1) the "1ms data" and (2) the "1 second data," respectively.

For simplicity's sake and because the "1ms data" window is relatively narrow, the 3 streams of "1ms data" points will be processed for their median values, which should give the most probable and practical description of the actual measurement. (If more accuracy is required and processing overhead is not prohibitive, that approach can be refined in later revisions; for example, running a regression on the 100 (by 3) data points to obtain linear vectors.) The procedure is as follows:

1. Subtract each consecutive value in the "1ms data" set by its preceding value, and divide each result by 10µs. This will result in 99 instantaneous acceleration values.
2. Find the median value of the 99 acceleration points. This will result in a median value every 10ms.

3. Let t= timestamp of each median value and m = the associated median value. Each set of 100 median values ("1 second data") that have been processed within the last 1 second will be processed as follows:

4. Obtain summations $\sum_{n=1}^{100} t_n$, $\sum_{n=1}^{100} m_n$, $\sum_{n=1}^{100} m_n^2$, $\sum_{n=1}^{100} m_n^3$, $\sum_{n=1}^{100} m_n^4$, $\sum_{n=1}^{100} tm_n$, and $\sum_{n=1}^{100} tm_n^2$

5. The summations will be used to compute the coefficients of a 2nd order polynomial in the form $m= a+bt+ct^2$ that will describe the trend of angular acceleration along a single axis during the last second.

6. A matrix of the above summations in the form $Ax=b$ would be:

$$\begin{bmatrix} 100 & \sum_{n=1}^{100} m_n & \sum_{n=1}^{100} m_n^2 \\ \sum_{n=1}^{100} m_n & \sum_{n=1}^{100} m_n^2 & \sum_{n=1}^{100} m_n^3 \\ \sum_{n=1}^{100} m_n^2 & \sum_{n=1}^{100} m_n^3 & \sum_{n=1}^{100} m_n^4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum_{n=1}^{100} t_n \\ \sum_{n=1}^{100} tm_n \\ \sum_{n=1}^{100} tm_n^2 \end{bmatrix}$$

7. To find the coefficients (a,b,c) the matrix must take the form $x = A^{-1}b$, where A^{-1} is the inverse of matrix A using the identity matrix method:

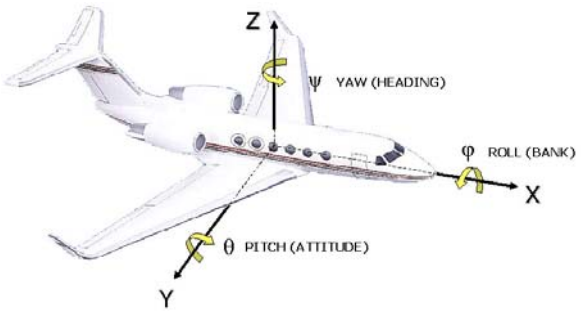
$$\begin{bmatrix} 100 & \sum_{n=1}^{100} m_n & \sum_{n=1}^{100} m_n^2 \\ \sum_{n=1}^{100} m_n & \sum_{n=1}^{100} m_n^2 & \sum_{n=1}^{100} m_n^3 \\ \sum_{n=1}^{100} m_n^2 & \sum_{n=1}^{100} m_n^3 & \sum_{n=1}^{100} m_n^4 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} A_{11}^{-1} & A_{12}^{-1} & A_{13}^{-1} \\ A_{21}^{-1} & A_{22}^{-1} & A_{23}^{-1} \\ A_{31}^{-1} & A_{32}^{-1} & A_{33}^{-1} \end{bmatrix}$$

$$A^{-1}b = x \ : \quad \begin{bmatrix} A_{11}^{-1} & A_{12}^{-1} & A_{13}^{-1} \\ A_{21}^{-1} & A_{22}^{-1} & A_{23}^{-1} \\ A_{31}^{-1} & A_{32}^{-1} & A_{33}^{-1} \end{bmatrix} \begin{bmatrix} \sum_{n=1}^{100} t_n \\ \sum_{n=1}^{100} tm_n \\ \sum_{n=1}^{100} tm_n^2 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

8. The above matrix operations are performed for each axis and result in three predictive polynomials with respect to time t :

$$Yaw = a_y + b_y t + c_y t^2 \qquad Pitch = a_p + b_p t + c_p t^2 \qquad Roll = a_r + b_r t + c_r t^2$$

The NASA Standard Aeroplane is the model used to designate axes and their order of application, as shown in Figure 13.



Angle applied 1st	Z, ψ, yaw (heading)
Angle applied 2nd	Y, θ, pitch (attitude)
Angle applied 3rd	X, φ, roll (bank)

Figure 13. NASA Standard Aeroplane

The quaternion matrix method will be used to describe rotational motion, which allows for each successive rotation to be multiplied against a reference rotation for translation into a new reference rotation. The following steps describe the operational method:

1. Convert the predictive polynomials for *t=0* to quaternion format
2. Convert the *t=0* quaternion to matrix format
3. Convert the predictive polynomials for *t=(0+n)* to quaternion format
4. Convert the *t=(0+n)* quaternion to matrix format
5. Multiply the two matrices to obtain a transform matrix defining the predicted rotation of the aircraft
6. Convert the matrix back to quaternion format.

Converting predictive polynomials to quaternion format

A rotation quaternion takes the form $Q_R = w + x\mathrm{i} + y\mathrm{j} + z\mathrm{k}$,[3] Q_R comprised of 4 rotational operators, where:

$$\text{Scalar } w = \cos\frac{roll}{2} * (\cos\frac{pitch}{2} * \cos\frac{yaw}{2}) + \sin\frac{roll}{2} * (\sin\frac{pitch}{2} * \sin\frac{yaw}{2})$$

$$\text{Vector } x = \sin\frac{roll}{2} * (\cos\frac{pitch}{2} * \cos\frac{yaw}{2}) - \cos\frac{roll}{2} * (\sin\frac{pitch}{2} * \sin\frac{yaw}{2})$$

$$\text{Vector } y = \cos\frac{roll}{2} * \sin\frac{pitch}{2} * \cos\frac{yaw}{2} + \sin\frac{roll}{2} * \cos\frac{pitch}{2} * \sin\frac{yaw}{2}$$

$$\text{Vector } z = \cos\frac{roll}{2} * \cos\frac{pitch}{2} * \sin\frac{yaw}{2} - \sin\frac{roll}{2} * \sin\frac{pitch}{2} * \cos\frac{yaw}{2}$$ [4]

[3] NB. The complex numbers [i, j, k] are used as place-holders. They can otherwise be discarded for this application.

[4] Bobick, Nick, *Rotating Objects Using Quaternions,* Game Developer [online magazine] February 1998

Before converting to matrix format it's prudent to normalize the quaternion to prevent cumulative errors.

- Obtain the magnitude of the quaternion: $M_{agnitude} = \sqrt{w^2 + x^2 + y^2 + z^2}$
- $Q_{Norm} = (w + x + y + z)/M$

Converting quaternion format to matrix format

The quaternion matrix format is:

$$Q = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix}$$

Obtaining a transform matrix

Successive rotations are solved such that a reference matrix Q_a multiplied by a subsequent matrix Q_b will result in a new reference matrix.[5] The multiplication of two 4x4 matrices will result in a new matrix:

$$Q_a * Q_b = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} =$$

$$\begin{pmatrix} a_{00}*b_{00} + a_{01}*b_{10} + a_{02}*b_{20} & a_{00}*b_{01} + a_{01}*b_{11} + a_{02}*b_{21} & a_{00}*b_{02} + a_{01}*b_{12} + a_{02}*b_{22} \\ a_{10}*b_{00} + a_{11}*b_{10} + a_{12}*b_{20} & a_{10}*b_{01} + a_{11}*b_{11} + a_{12}*b_{21} & a_{10}*b_{02} + a_{11}*b_{12} + a_{12}*b_{22} \\ a_{20}*b_{00} + a_{21}*b_{10} + a_{22}*b_{20} & a_{20}*b_{01} + a_{21}*b_{11} + a_{22}*b_{21} & a_{20}*b_{02} + a_{21}*b_{12} + a_{22}*b_{22} \end{pmatrix} = Q_c$$

$$Q_c = \begin{pmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{pmatrix}$$

[5] NB. The order of multiplication is $Q_a * Q_b$ and is not commutative: $Q_a * Q_b \neq Q_b * Q_a$.

Converting the matrix back to quaternion format

$$w = \pm 0.5\sqrt{(1 + c_{00} + c_{11} + c_{22}}$$

$$x = \frac{c_{21} - c_{12}}{4 * w}$$

$$y = \frac{c_{02} - c_{20}}{4 * w}$$

$$z = \frac{c_{10} - c_{01}}{4 * w}$$

Thus $Q_{t=(0+n)} = w + x\mathrm{i} + y\mathrm{j} + z\mathrm{k}$. The value of $t_{(0+n)}$ for each prediction will be assigned taking into account the overheads associated with processing the DFCC's request for gyro data, avionics bus traffic, DFCC computation lag, and control surface response.

This an initial design suggested as a starting point for concept development, subject to changing requirements, conditions, and engineering tradeoffs. There are other design paths for a MEMS gyro. For example, the MEMS' analog outputs could be used to drive a design concept based on sample-and-hold integration amplifiers and PLL, with final conversion to digital at the Avionics Bus Interface.

References

Beal, Robert M., *Derivation Of The Equations Of Gyroscopic Motion,* (May 2003) http://www.gyroscopes.org/math2.asp

Layton, Garrison P., *A Review of Recent Developments in Flight Test Techniques at the Ames Research Center, Dryden Flight Research Facility*, NASA Technical Memorandum 86039, April 1984

Shy, Karla S.; Hageman, Jacob J.; Le, Jeantette H., *The Role of Aircraft Simulation in Improving Flight Safety Through Control Training*, NASA/TM-2002-210731, August 2002

Moes, Timothy R.; Smith, Mark S., *Flight Investigation of Prescribed Simultaneous Independent Surface Excitations for Real-Time Parameter Identification*, NASA/TM-2003-212029, October 2003

Williams-Hayes, Peggy S., *Selected Flight Test Results for Online Learning Neural Network-Based Flight Control System,* NASA/TM-2004-212857, 2004

Fisher, David F., *Six Decades of Flight Research: An Annotated Bibliography of Technical Publications of NASA Dryden Flight Research Center, 1946–2006,* NASA/TP-2007-213684, May 2007

Williams-Hayes, Peggy S.; Bosworth, John T., *Flight Test Results from the NF-15B Intelligent Flight Control System (IFCS) Project with Adaptation to a Simulated Stabilator Failure,* NASA/TM-2007-214629, December 2007

Kalman, R. E., *A New Approach to Linear Filtering and Prediction Problems*, Transactions of the ASME Journal of Basic Engineering, 82 (Series D): 34-45 ©ASME 1960

Looney, Mark, *Optimizing MEMS Gyroscope Performance with Digital Control,* AN-942 ©Analog Devices Inc. 2008

Bergeron, Joseph; Looney, Mark; Analog Devices, Inc., *Making MEMS accelerometers work in motion control*, URL: http://www.planetanalog.com/showArticle?articleID=201001402, Jul 13, 2007

Anonymous, *UT1750AR RadHard RISC Microprocessor Data Sheet*, ©Aeroflex Colorado Springs, Inc., May 2003

Baker, Martin John, *http://www.euclideanspace.com,* ©1998-2009

Haeussler and Paul, *Introductory Mathematical Analysis (9th Ed.)*, ©Prentice-Hall, 1999

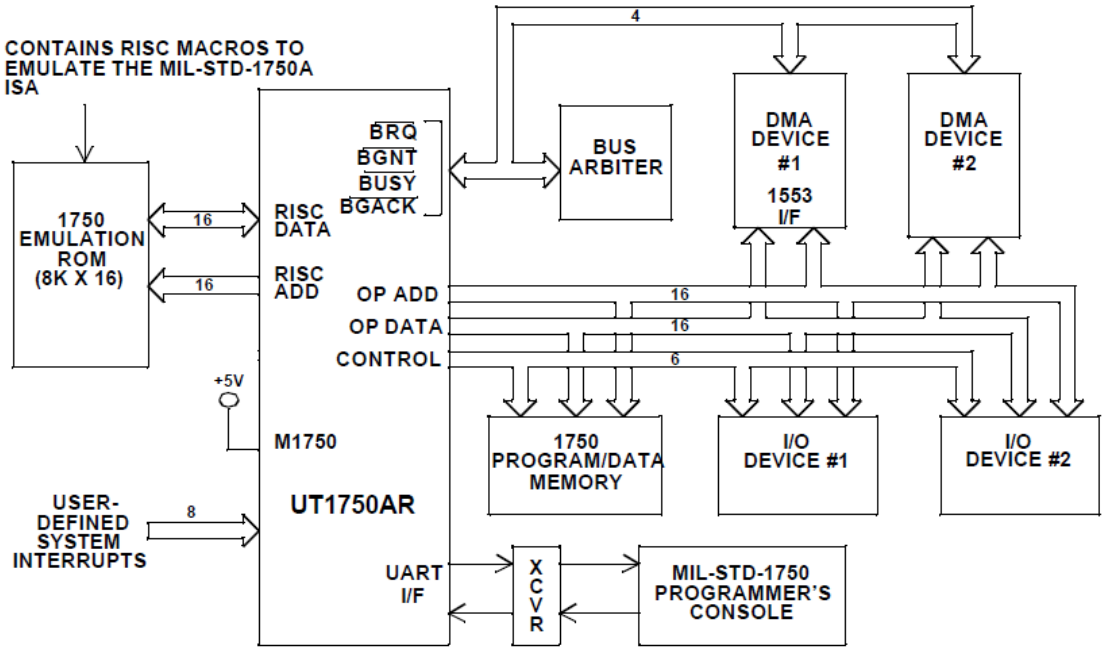


Figure 14. 1750A Processor RISC Architecture

Pin	Name	Pin	Name	Pin	Name
a1	D00	b1	BBSY*	c1	D08
a2	D01	b2	BCLR*	c2	D09
a3	D02	b3	ACFAIL*	c3	D10
a4	D03	b4	BG0IN*	c4	D11
a5	D04	b5	BG0OUT*	c5	D12
a6	D05	b6	BG1IN*	c6	D13
a7	D06	b7	BG1OUT*	c7	D14
a8	D07	b8	BG2IN*	c8	D15
a9	GROUND	b9	BG2OUT*	c9	GROUND
a10	SYSCLK	b10	BG3IN*	c10	SYSFAIL*
a11	GROUND	b11	BG3OUT*	c11	BERR*
a12	DS1*	b12	BR0*	c12	SYSRESET*
a13	DS0*	b13	BR1*	c13	LWORD*
a14	WRITE*	b14	BR2*	c14	AM5
a15	GROUND	b15	BR3*	c15	A23
a16	DTACK*	b16	AM0	c16	A22
a17	GROUND	b17	AM1	c17	A21
a18	AS*	b18	AM2	c18	A20
a19	GROUND	b19	AM3	c19	A19
a20	IACK*	b20	GROUND	c20	A18
a21	IACKIN*	b21	SERCLK*	c21	A17
a22	IACKOUT*	b22	SERDAT*	c22	A16
a23	AM4	b23	GROUND	c23	A15
a24	A07	b24	IRQ7*	c24	A14
a25	A06	b25	IRQ6*	c25	A13
a26	A05	b26	IRQ5*	c26	A12
a27	A04	b27	IRQ4*	c27	A11
a28	A03	b28	IRQ3*	c28	A10
a29	A02	b29	IRQ2*	c29	A09
a30	A01	b30	IRQ1*	c30	A08
a31	-12V	b31	+5V STDBY	c31	+12V
a32	+5V	b32	+5V	c32	+5V

Figure 15. VME Bus Connector Signals

Code Fragments

Euler-to-quaternion conversion.

```
EulerToQuat(float roll, float pitch, float yaw, QUATERNION* quat)
{
	float cr, cp, cy, sr, sp, sy, cpcy, spsy;


// calculate trig identities
cr = cos(roll/2);

	cp = cos(pitch/2);
	cy = cos(yaw/2);


	sr = sin(roll/2);
	sp = sin(pitch/2);
	sy = sin(yaw/2);

	cpcy = cp * cy;
	spsy = sp * sy;


	quat->w = cr * cpcy + sr * spsy;
	quat->x = sr * cpcy - cr * spsy;
	quat->y = cr * sp * cy + sr * cp * sy;
	quat->z = cr * cp * sy - sr * sp * cy;
}
```

Bobick, Nick, *Rotating Objects Using Quaternions,* Game Developer [online magazine] February 1998

```
**// C++ code for converting Matrix to Quaternion


inline void CalculateRotation( Quaternion& q ) const
{
float trace = a[0][0] + a[1][1] + a[2][2] + 1.0f;

if( trace > M_EPSILON )
{
float s = 0.5f / sqrtf(trace);
q.w = 0.25f / s;
q.x = ( a[2][1] - a[1][2] ) * s;
q.y = ( a[0][2] - a[2][0] ) * s;
q.z = ( a[1][0] - a[0][1] ) * s;
}
else
{
if ( a[0][0] > a[1][1] && a[0][0] > a[2][2] )
{
float s = 2.0f * sqrtf( 1.0f + a[0][0] - a[1][1] - a[2][2] );
q.x = 0.25f * s;
q.y = (a[0][1] + a[1][0] ) / s;
q.z = (a[0][2] + a[2][0] ) / s;
q.w = (a[1][2] - a[2][1] ) / s;

}
else if (a[1][1] > a[2][2])
{
float s = 2.0f * sqrtf( 1.0f + a[1][1] - a[0][0] - a[2][2] );
q.x = (a[0][1] + a[1][0] ) / s;
q.y = 0.25f * s;
q.z = (a[1][2] + a[2][1] ) / s;
q.w = (a[0][2] - a[2][0] ) / s;

}
else
{
float s = 2.0f * sqrtf( 1.0f + a[2][2] - a[0][0] - a[1][1] );
q.x = (a[0][2] + a[2][0] ) / s;
q.y = (a[1][2] + a[2][1] ) / s;
q.z = 0.25f * s;
q.w = (a[0][1] - a[1][0] ) / s;
}
}
}
```

by Angel, http://sourceforge.net/users/nobody/