DESIGN AND FPGA IMPLEMENTATION OF NON-LINEARITY COMPENSATION OF CAPACITIVE PICK-OFF MEMS ACCELEROMETER FOR SATELLITE LAUNCH VEHICLES

Thampi Paul [1], S.Vijin Jenius [1], M. Sasi Kumar [2]

[1]ISRO Inertial Systems Unit, Vattiyoorkavu, Thiruvananthapuram-695013, India.

[2]Dept. of Electronics & Communication, Marian Engg. College, Thiruvananthapuram -695082, India

Emails: thampi_paul@yahoo.co.uk, vijin_jenius@yahoo.com

Abstract- This paper presents the algorithm on the compensator design for eliminating the non-linearity in the capacitive pick-off MEMS open-loop accelerometer and its implementation in the FPGA. A simple and elegant method is presented for the purpose. In the sensor model of compensator, upto 3rd order terms are taken. The first step approximation is derived using linear model. This approximation is improved over iterations to reduce the non-linearity. With this method, the inertial navigational grade performance is achieved. The algorithm is coded in VHDL, simulated, synthesized and implemented in the FPGA and tested. Test results matches closely with that of simulations. The VHDL design can be easily targeted into an ASIC to realize an integrated smart sensor.

Index terms: Accelerometer, Linearity, FPGA, error compensation

I. INTRODUCTION

The accelerometers used for Inertial Navigation in Satellite Launch Vehicles demand excellent performance in terms of sensitivity, resolution, linearity and bias & scale factor stability over time and environmental changes. The required performance can be achieved by adopting a systems based design perspective and developing the appropriate signal processing technique than to try to improve the micro-fabrication process and the design of mechanical sensing element itself. Linearity errors originating both from the characteristics of the device itself and from the imperfections of the interface circuits can be eliminated by suitable compensator. The linearity of the MEMS accelerometer, with open-loop design and capacitive sensing for a dynamic range of ±20g with 200 Hz bandwidth and 250µg resolution for satellite launch vehicle application, has to be improved to achieve navigational grade performance. A compensation module is added to the sensor system as integral part of the sensor. The non-linearity has been characterized and an algorithm has been designed to compensate the linearity errors. This paper describes the algorithm for Non Linearity

Compensation for the capacitive pick-off MEMS open-loop accelerometer for Satellite Launch Vehicle Application and implementation in Field Programmable Gate Array (FPGA). The algorithm has been coded in VHDL, simulated and synthesized. The design has been fused in FPGA. The fused FPGA has been tested and results are presented.

II. ALGORITHM

The non-linear characteristics of the capacitive pick-off MEMS accelerometer are described mathematically, its inverse is found and cascaded in the signal path. Delta capacitance versus deflection of the designed accelerometer with a proof mass dimension of 3000 micron x 3000 micron, initial gap of 2 micron and full scale deflection of 0.6 micron has been fitted as a third order polynomial curve to improve its linearity from 2.38% to 0.036% [1].

This third order polynomial with computed coefficients by curve fitting for linearization is

$y=12.83\,x^3-1.49\,x^2+43.3x-0.469$ --------- (1)

where,

y is the delta capacitance in pF

x is the deflection in micron.

The algorithm for computing the deflection x from the measured delta capacitance y is given below.

1. Define the coefficients of the third order polynomial as in eqn (1).
2. Read the value of y.
3. Compute x from first order equation, neglecting second and third order terms in eqn (1). (ie. y = 43.3x-0.469)
4. With this value of x, compute y by substituting x in eqn (1) and assign the value to y_1.
5. Compute the difference between y & y_1. (error 1)
6. If error1 is greater than 0.0000325 (delta Capacitance corresponding to 25 microg),

   then
7. Decrement x by 0.0000075 (deflection corresponding to resolution of 250 microg).
8. With this value of x, compute y by substituting x in eqn (1) and assigning the value to y_1.
9. Compute the difference between y & y_1. (error)

```
                If error is less than error 1
                Repeat steps 7 to 9 till error < 0.0000325.
            Else
10. Increment x by 0.0000075. (deflection corresponding to resolution of   250
    microg).
11. With this value of x, compute y_1 by substituting x in eqn (1) and assign the value
    to y_1.
12. Compute the difference between y & y_1. (error)
                Repeat steps 1 to 12 till error < 0.0000325.
13. Write the value of x, which is the deflection of the proof mass.
```

Hence the delta capacitance y is given to the compensator module to obtain the corresponding deflection x as discussed in the algorithm steps 1 to 13. The acceleration is computed from the mechanical sensitivity of 0.03 micron /g [1].

III. RTL DESIGN

VHDL code of RTL design has been generated based on the requirements and algorithm. The algorithm is coded in VHDL for FPGA implementation, which can be targeted into an ASIC to realize an integrated smart sensor. The block diagram for the compensator module is given in figure 1.

Various signals/ports of the module are:

1) FSM_IN: This is the input port and is used for providing the input(y) to the module. The input y should be in 2's complement form.

2) CLK: This is the master clock used in the module.

3) RESET: This is the active high, master reset for the module, and should be asserted once before every conversion.

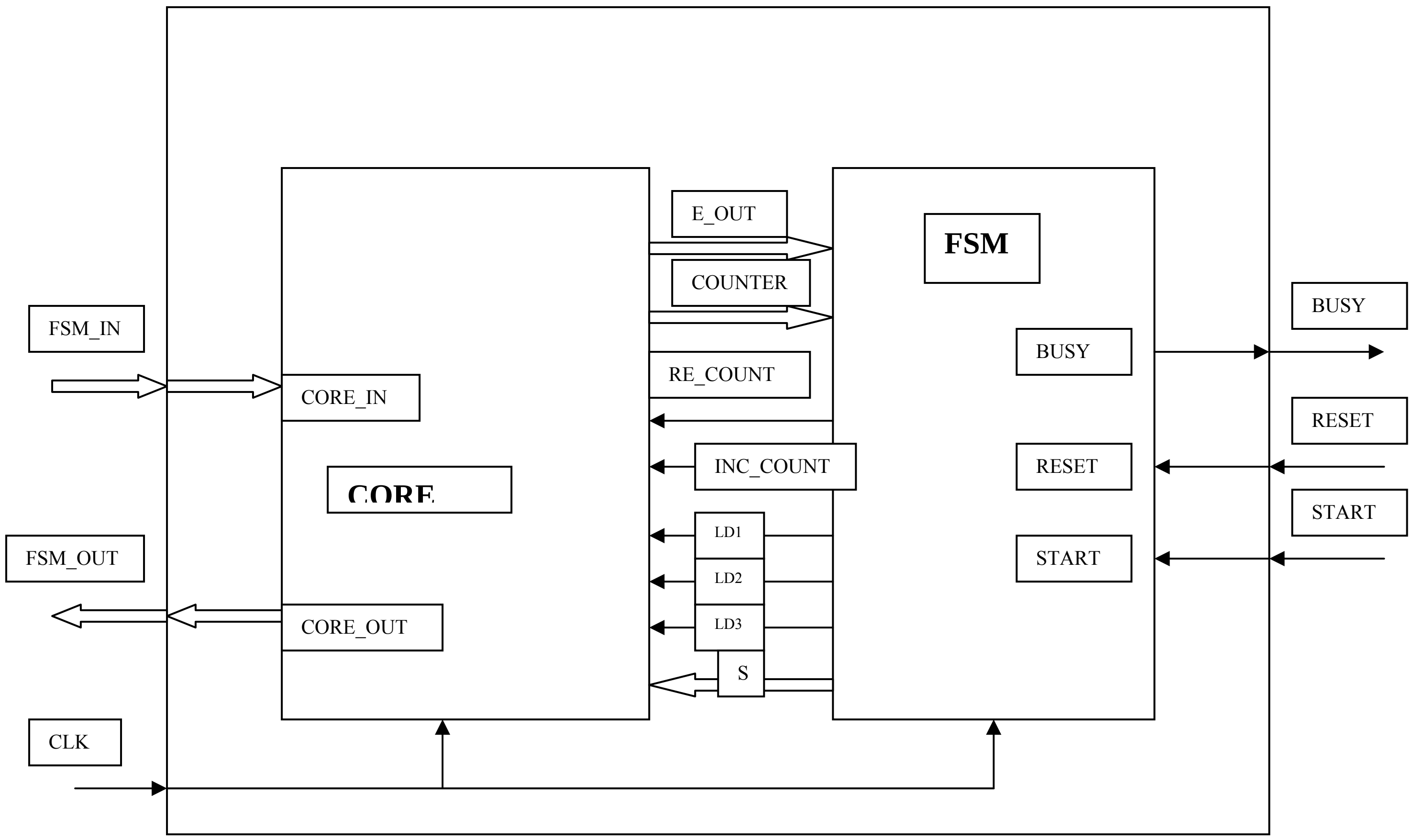

Figure 1. Block diagram of compensator module

4) START: This is the active high, start signal for the module, and should be asserted before starting any conversion (when input is ready at the input port).

5) FSM_OUT: This is the output port of the module, and gives the value for x.

6) BUSY: This pin indicates the busy status of the module. It is active high. i.e. high busy indicates that the module is still busy computing, and the value at the output port has no meaning.

The format for the input (2's compliment form) is as follows.

The input to this module is a 24 bit word. The bit details are

MSB is the sign bit

Next 7 bits i.e. (bit 22 to bit 16, from right) are used for representing the integer part of the input

Rest 16 bits i.e. (bit 15 to bit 0, from right) are used for representing the fractional part of the input.

23	22 16	15 0
S	I	F

. **binary point**

Where:

S: sign bit

I: integer bit

F: fraction bit

The whole module is broadly divided into two parts viz. CORE and FSM. The CORE is the VHDL implementation of the flow diagram in figure 2. Various individual blocks of CORE are given below:

1) REGISTER :

   It is a generic, positive edge triggered register used as a buffer (Reg)

   INPUTS: data input, clock, load

   OUTPUT: data output

2) NEGATOR : it is a block for providing the negative of any number

   INPUTS: data input

   OUTPUT: data output

3) FULL ADDER: this block is used for performing the addition operations.

   INPUTS : data input 1 & data input 2

   OUTPUTS : data output(sum)

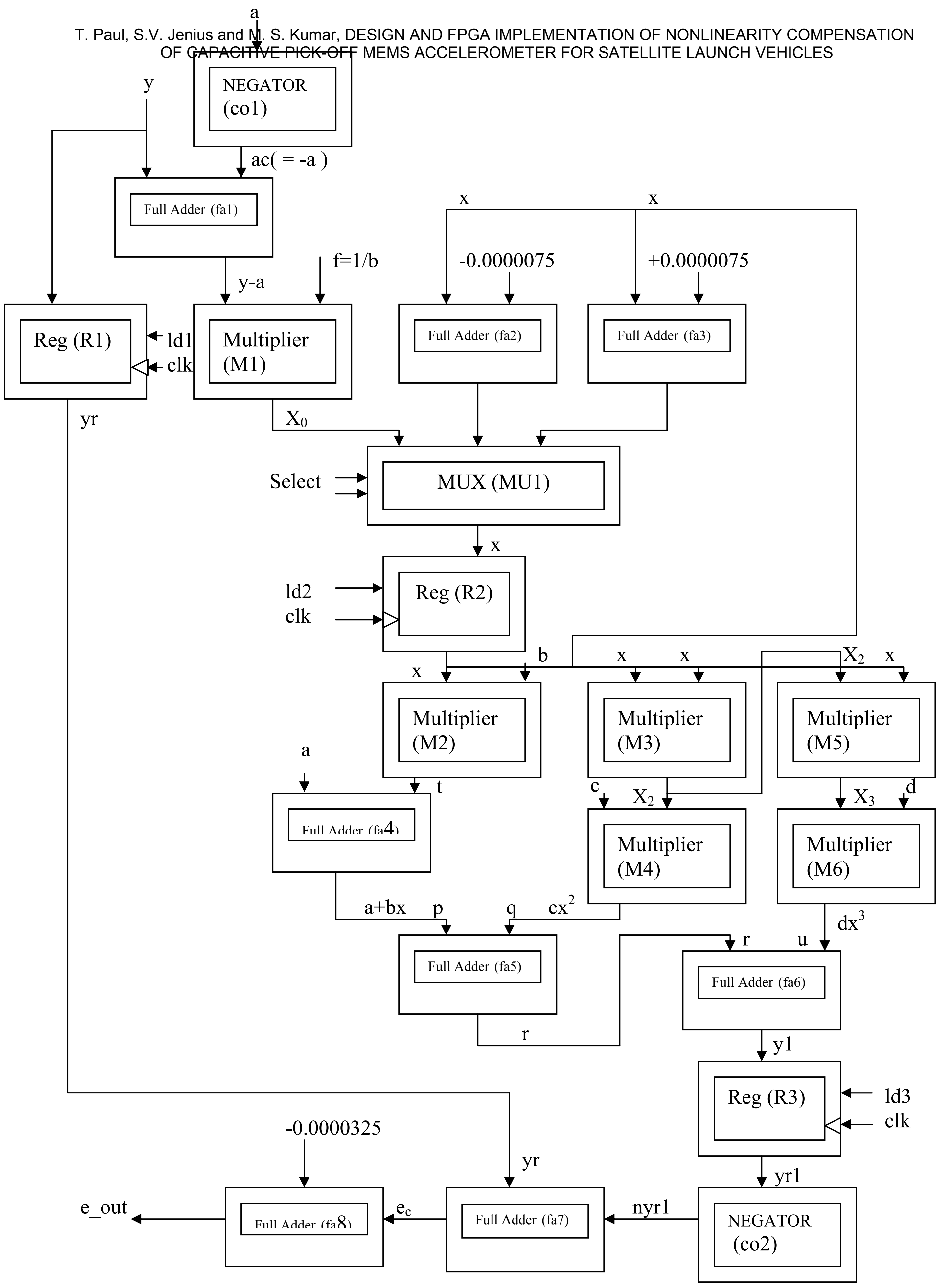

Figure 2. Flow diagram of algorithm

4) MUX: this block is used for routing one of the three input words, in accordance with the 2 bit Select signal

   INPUTS: 3 data input ports, select signal

   OUTPUT : one data output port

5) ROM: this is a cluster of ROM, with each block storing a constant word, required for the execution of the algorithm.

   INPUT: nil

   OUTPUT: one port per ROM for output word

   This module is made programmable by introducing one input word, and introducing one programming mode in the FSM part.

6) COUNTER: this block is used for monitoring the number of iterations that are made during the execution of the program. This block helps in preventing the system in an infinite loop condition.

   INPUT: clock, reset signal, increment signal

   OUTPUT: counter output.

7) MULTIPLIER: this is a single clock, generic, 2's complement multiplier. This block shown in figure 3 consists of various blocks, mentioned below

   a. Binary multiplier: this is an unsigned, single clock, binary multiplier. [7]
   b. 2Comp: this block is used for converting the numbers from 2's complement form to signed magnitude form. The reverse process is also done using the same block.

   A simple logic is also used for deciding the sign of the output (the binary multiplier used here is unsigned multiplier only).

   INPUT: two ports for multiplier and multiplicand

   OUTPUT: one port for the product.

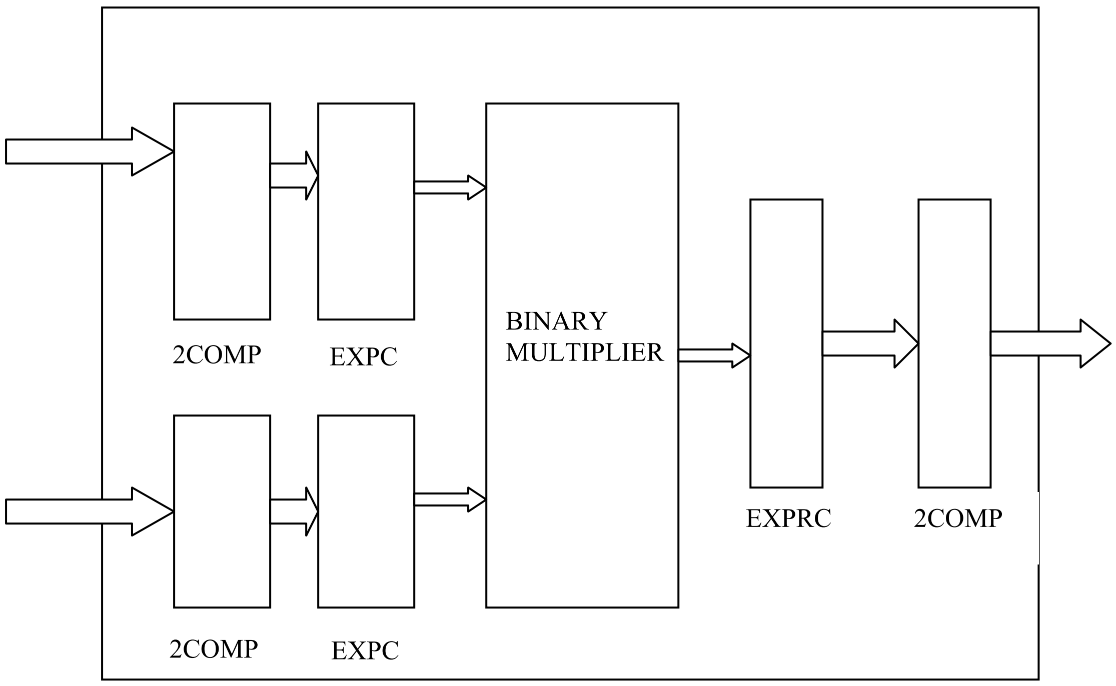

Figure 3. 2's Complement multiplier

This CORE module has various control signals that are generated by the other block ie FSM. Various signals are:

1) ld1, ld2, ld3 : these are the load signals of the three registers used in the core
2) re_count: this signal is used for resetting the counter of the core.
3) Inc_count : this signal is used for incrementing the value of the counter.
4) Select: this is the select signal for the MUX.

Besides this, the other inputs/output ports of the core are

1) CORE_IN: this port is used for feeding the input(Y) to the core
2) CLK: this is the master clock signal, used in the core
3) CORE_OUT: this port is used for outputting the result computed from the core
4) COUNTER: this port is used for outputting the value of the counter.
5) E_OUT: this port is used for outputting the value of the error (used for iterations in the algorithm).

The other part of the module, namely the FSM (FINITE STATE MACHINE), is used for driving the core mentioned above, in accordance with the algorithm. The State Transition Diagram of the FSM is shown in figure 4. FSM generates various control signals of the core module (mentioned above), according to its state, and proceeds on to next state in accordance with the algorithm given below.

FSM – ALGORITHM:

1) The FSM moves on to the next state on every positive edge of the clock, while it decides the next state on every negative edge of the clock.

2) The first step is to activate the reset signal (i.e. keep it at logic ‘1’ for minimum of one clock cycle).

3) When reset is activated, FSM enters the state ‘S0’ on the positive edge of the clock. In this state, all the load signals (ld1, ld2, ld3), the select signal (Select), busy signal, counter signals (reset and increment) are at logic ‘0’. i.e the core is idle and wont perform any operation.

4) When start is activated (it is kept at logic ‘1’, with reset at logic ‘0’), the next state of the FSM is assigned as S1. Thus on the next positive edge of the clock it will go to state S1. In S1, busy goes high, and also ld1 is pulsed high. Thus whatever value is there at the input, gets loaded in register1 (referred to as yr in fig 2). Also value of select signal s is “00”, i.e value of x, before register2 is x = (y-a)/b.

5) On the next positive edge, FSM enters state S2. In this state, ld1 goes low, whereas ld2 is made logic ‘1’. Thus value of x calculated in the previous state is stored in the register2 i.e. xo = y-a. also, increment signal of the counter is pulsed high. So it starts counting from this state onwards.
Now referring to fig2 again, value of x2 (i.e output of multiplier M3) is x*x, whereas value of x3 (i.e. output of multiplier M5) is x*x*x. writing in equation form, the output after s2 happens :
$X_2 = x^2$
$X_3 = x^3$

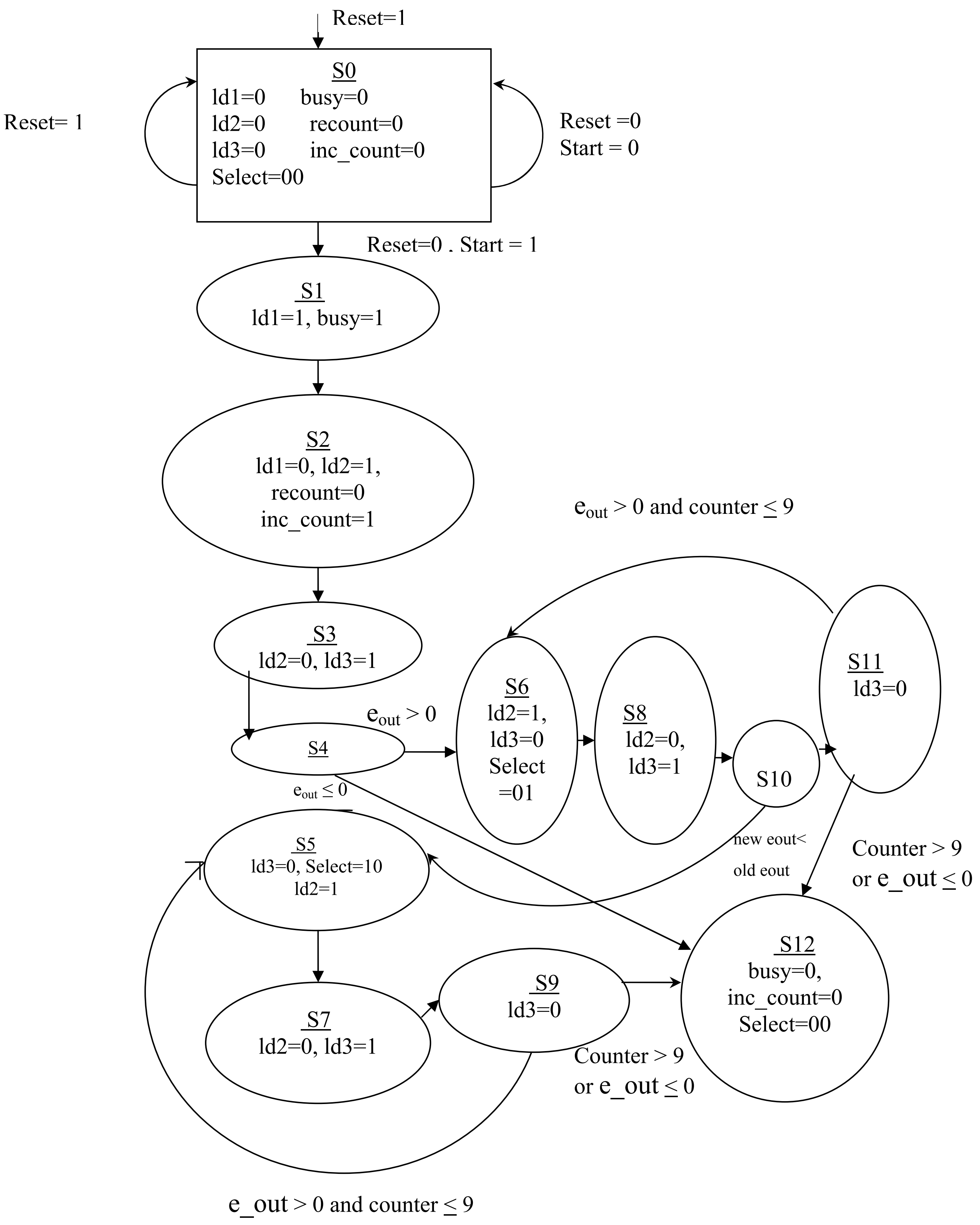

Figure 4. State Transition Diagram

t = x*b

p = a + t = a+ b*x

q = X_2 * c = x^2*c

r = p + q = a + b*x + c*x*x

u = X_3*d = x^3*d

y1 = r + u = y=a+bx+cx^2+dx^3

6) Next state for the FSM is S3. in this state ld2 is pulsed low, and ld3 is pulsed high. Now in state S3, this value of y1 gets stored in register R3 i.e yr1 = y1 (calculated in the previous state). Also following calculations are done

nyr1 = -yr1 = -y1

e_c = |yr+nyr1| = |y-y1}

e_out = e_c +e = e_c-0.0000325

7) Next state for the FSM is S4. In this state, the FSM checks the value of e_out. If e_out > 0 i.e. if e_c>0.0000325 then next state is S6,else next state is s12. No more operations are performed in this state.

8) In state S6, ld3 is pulsed low and ld2 is pulsed high. The value of mux select signal is Select = “01”. i.e. now x = S2 = x-0.0000075. Again the new value of y1 is calculated as in state S2. the next state is S8, where the new value of e_out is calculated. Next it goes to state S10 where it compares the new_e_out with earlier e_out. If new_e_out > e_out then the next state is S5 else it goes to state S11. In the subsequent state (S11) value of e_out is again compared. If e_out < 0 or counter > 10 then the next state is the output state (S12) else the next state is S6.

9) In state S5, again ld3 signal is pulsed low, and ld2 signal is pulsed high. Also the select signal is made Select = “10” i.e. x = s3 = x+0.0000075. Also the new value of y1 is calculated according to this value of x (as in state S2). While in subsequent state, i.e. S7 new value of e_out is calculated as in state S3. after state S7, next state of the FSM is s9. here again value of e_out is compared and if e_out <0 (or if the value of counter > 10) then the next state is S12 (the output state), else the next state is S5. Mathematically, if e_out<0 i.e if e_c < 0.0000325 then the output is ready, else FSM goes back to state S5. (also even if value of e_out > 0.0000325, but the value of

counter > 10, then also the machine goes to the output state. This is incorporated to avoid infinite loops (where value of e_out never goes below 0). In the output state, i.e. state S12, busy is made low, and output is available at the output port. All other signals are just like in the reset state, so no computations will be there, even if there are computations, there will not be any change in the output because ld2 is low.

10) The machine will now remain in this state, till the reset signal is again asserted (that is, it is made logic ‘1’).

OPERATIONAL SEQUENCE OF THE MODULE:

1) Reset signal should be activated, before using the module.
2) At this state module will be in reset condition, with busy signal low.
3) Data should be provided at the input port.
4) Once data signal has settled, start signal should be asserted.
5) The module will start operating and busy signal will go high.
6) The value at the output port is meaningful, only when busy signal goes low again.
7) After busy signal goes low, activate reset again for starting the next conversion, and follow the same steps.
8) Once the start signal is asserted, the input to the module can be altered (after two clock cycles), as the module uses a register for storing the input value. Thus meanwhile the module calculates the output, next value of input can be provided.
9) The start signal has any meaning only when reset signal is de-active. i.e. even if start signal is asserted before de-asserting the reset signal, it (start) should be kept in active position for at least one or two clock cycles, after de-asserting the reset signal.
10) The need for resetting the module, before every conversion can be eliminated, by a slight change in the last state (S12) of the FSM.
11) The module can be made programmable by inserting one programming state in the FSM, and changing the ROM module, accordingly.
12) The input should be in 2’s compliment form.

IV. VHDL CODING

VHDL codes have been generated for all the sub modules in section III and integrated, amounting to approximately 400 lines. A thorough code walk through has been carried out to weed out any error. It consists of multiplier, which involves conversion to exponent form. The design was entered using “Libero VHDL Editor”. A test bench was created using the same editor and functional simulation was carried out using Libero IDE Simulator “Model Sim” to verify that the logic of the design is correct. The Model Sim opens and compiles the source files. It simulates and a wave window opens to display the simulation results as shown in figure 5.

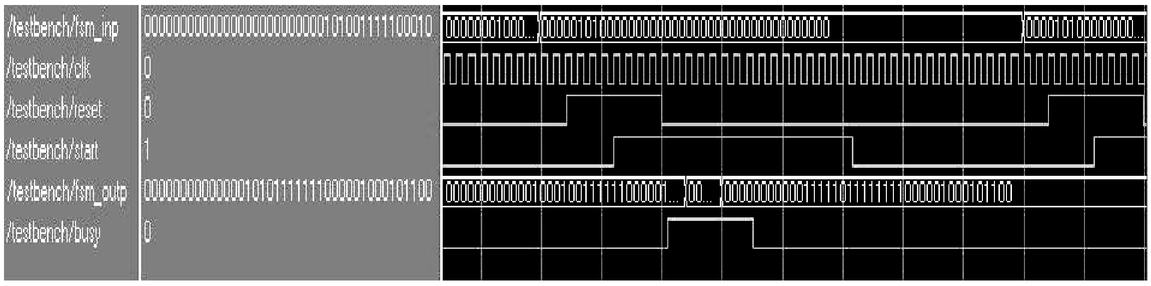
/testbench/fsm_inp 0000000000000000000000000101001111100010
/testbench/clk 0
/testbench/reset 0
/testbench/start 1
/testbench/fsm_outp 00000000000000010101111111000001000101100
/testbench/busy 0

Figure 5. Pre synthesis wave forms

V. SYNTHESIS & NETLIST GENERATION

The gate count required was about 450K. Device A3P1000 from third generation family of Actel Flash FPGA (PROASIC3) with gate complexity of 1000K having pin count of 208 in Plastic Quad Flat Package was selected.

The simulated codes have been synthesized for the selected target device using “Sinplify” to generate an EDIF netlist. Post synthesis simulation is performed and the waveforms captured are shown in figure 6 and are found in order with the simulation results.

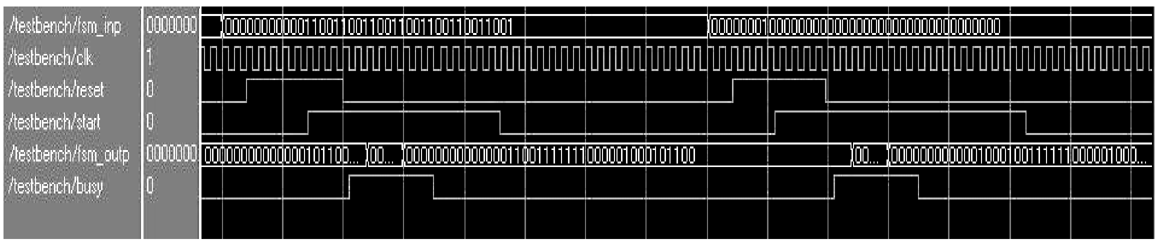

Figure 6. Post synthesis wave forms

The input/output pin assignments of ProASIC3, A3P1000, 208 PQFP were suitably configured for testing convenience.

VI. PLACE AND ROUTE

Libero IDE “Actel Designer” has been used to Place and Route the synthesized design. Timing Simulation with back annotated timing has been carried out with Model Sim HDL simulator for high/low temperature and min/max voltage. Post layout simulation waveforms captured are shown in figure 7.

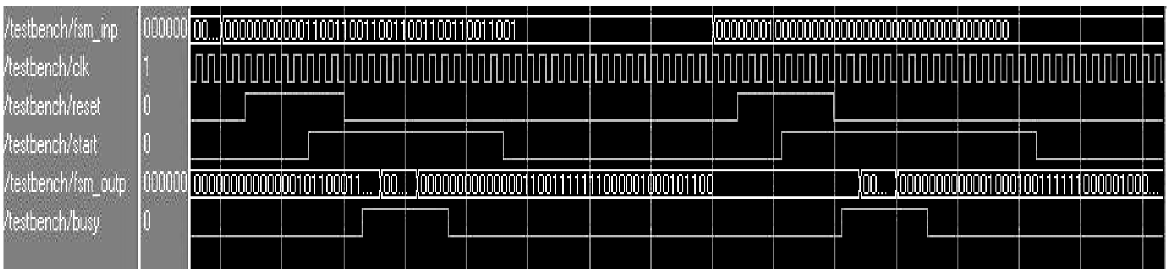

Figure 7. Post layout waveforms

Worst case timing with different clock frequencies (200Hz-20KHz) have been carried out. Timing requirements were met in the first attempt itself.

VII. PROGRAMMING / FUSING

The programming file is generated. The FPGA (ProASIC3, A3P1000, 208 PQFP) is programmed / fused using Actel Flash Pro –3 programmer. After fusing, the FPGA has been verified using “Logic Navigator” for its correctness.

VIII. EVALUATION

The programmed FPGA was placed to the socket of the ProASIC3 Evaluation board. Functionality of the design is verified using Pro ASIC3/E Evaluation Board with silkscreen labeling A3PE-A3PEVAL-BRD-1.

IX. RESULTS

Various capacitance values were fed to the input of the fused FPGA compensator module and corresponding deflection outputs were read. The results match with the simulation results. Three typical results are given in table-1.

Table-1 Typical results

Input (capacitance)		Output (deflection in micron)		
		Simulation		Actual
hex	Decimal (pF)	hex	Decimal	hex
01.0000	1.0	00.089E	0.033661	00.089E
05.0000	5.0	00.1F7E	0.123016	00.1F7E
19.0000	25.0	00.9401	0.578140	00.9401

X. CONCLUSION

The Linearity Error Compensation algorithm for the capacitive pick-off MEMS open-loop accelerometer for Satellite Launch Vehicle Application has been VHDL coded, simulated, synthesized, implemented in FPGA and tested. For navigational grade performance, calibration of the accelerometer has to be performed at individual device level and the coefficients to be generated by curve fitting to eliminate linearity errors originating both from the characteristic of the device itself and from the imperfections from the interface circuits. For a well-controlled fabrication process, the estimated coefficients presented can be fed and non-linearity can be compensated. For a batch where manufacturing tolerances are minimum, batch compensation can be applied after calibrating few samples.

ACKNOWLEDGEMENT

Authors thankfully acknowledge Mr.Kuldeep Chhakara, ISRO Inertial Systems Unit, Thiruvananthapuram for the useful suggestions given during the preparation of this paper.

REFERENCES

[1] Thampi Paul etl "Design and optimization of bulk micromachined accelerometer for space applications" in International Journal on Smart Sensing and Intelligent Systems, 2008, pp 1019-1030.

[2] Elwenspoek, "Mechanical Microsensors," Springer, New York, 2002

[3] Gad-El-Hak & Mohamed, "MEMS Hand Book,"CRC Press, Boca raton, 2002.

[4] Thampi Paul P, Vijin Jenius S and Sasikumar M, "Characterization and Nonlinearity compensation of Capacitive sensing MEMS accelerometer for launch vehcle

applications," in Proceedings of the National Society for Insturmentation of ISOI (NSI-30), 2005, pp. 417-421.

[5] Hsu & Tai-Ran, "MEMS and Microsystems-Design and Manufacture," Tata McGraw-Hill, New Delhi, 2002

[6] Fraden, "Hand Book of Modern Sensors-Physics, Design and Application," Springer, New York, 2004

[7] J. BHASKER, " A VHDL PRIMER", Pearson Education, 3rd e.d., pp. 316-318, New Delhi, 2003