ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 6, Issue 4, pp: (167-173), Month: October - December 2018, Available at: www.researchpublish.com

FPGA Based Optimized Implementation of Control Algorithm for Semi-active Suspension Systems

Sanjay Eligar [1], R M Banakar [2]

[12] Department of Electronics & Communication
[12]BVB College of Engineering & Technology, Hubli, India

Abstract: **Historically system design approaches favored the use of digital signal processors to implement controllers for embedded systems. Processor based implementation lack the speed advantage offered by custom hardware and concurrent processing because of their sequential nature. In some applications using a DSP would be overkill in design, and other approaches need to be explored. Custom hardware is an option where in the implementation could be done using FPGA, ASIC or an SoC. FIRs are a major component of signal processing applications and the design and implementation in FPGA of sigma 1 controller for semiactive suspension is presented in this paper. Alternate architectures for multiplication are explored and results are presented. Since multiplication by constant coefficients is needed for FIR implementation CSD notation is employed which minimizes the number of non-zeros in the constant multiplier. The results indicate area savings up to 74% and improvement is speed of computation by 7.2 times. Further research in this area needs to explore the possibility of using transposed form of FIR structure and possibility of optimization in resources consumed.**

Keywords: **velocity estimation, FIR filters, CSD, multiplier, shifter.**

I. INTRODUCTION

Semiactive suspension control is one of the leading areas of research in automotive sector, especially in the passenger vehicle segment. The primary goal of the controller algorithm is to minimise the trade off between passenger comfort and road handling which are presented in [1] along with passive and active suspension systems. The semi-active suspension is chosen over other options because of the simplicity in design and efficiency in power. The control algorithm is designed and validated at system level of design, but the actual implementation needs to be done in hardware. The semi-active suspension controller application discussed here demands that the solution be of low cost and consume less power.

Implementation of control algorithms is done using special processors like a digital signal processor (DSP) which provides software interface to fine tune the controller parameters on the fly [2]. The design and simulation of the controller is done using system level software which may use the vehicle dynamics and the MR damper model in the initial stages. Once the controller parameters are finalised, the validation is done using a rapid prototyping unit like the dSPACE AutoBox [3] in which the controller is implemented using DSP as shown in Fig. 1.

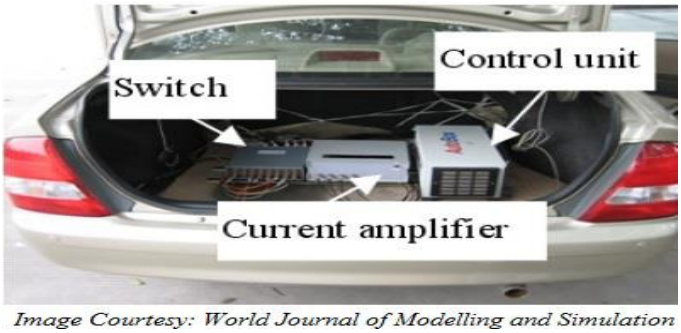

Image Courtesy: World Journal of Modelling and Simulation

Fig. 1. Controller validation using DSP development platform

After experimental validation the controller is then implemented on a dedicated DSP chip set based embedded system. Thus, in controller formulations implemented using system level software there is a constraint of using an off-the shelf DSP, which may be an overkill if simple controllers are needed. The solution is then to implement a custom hardware without using the DSP. The flexibility in model based design comes at the cost of performance or speed of computation since typically most of the algorithms use digital Finite-Impulse-Response (FIR) filters. FIR filters are characterised by typical mathematical operations on sampled signals as shown in equation (1), where each output sample (y[n]) is the sum of a finite number of weighted samples of the input signal and its delayed samples(x[n], x[n − 1]...x[n − k]). The weights are the FIR coefficients (b0, b1...bN).

$$y[n] = \sum_{k=0}^{N} b_k x[n-k] \tag{1}$$

The computation of the output requires multiply and accumulate (MAC) arithmetic and delay elements along with memory to store the coefficients in appropriate format. DSPs are special type of microprocessors which optimize the MAC operations through a customized ALU and memory. DSPs are easily programmable using high-level languages and the designer need not worry about architectural decisions. This approach is suitable if the sample rates are in the range of a few kHz [4]. The instructions in a DSP are executed sequentially and hence limit the maximum speed of operation to a few clock cycles. But such approaches are very generic and not suited if performance, area, cost and low power dissipation are the criteria for design. FIR computation involves similar MAC operations, but the length of computation depends on the number of taps in the filter. The algorithm slows down if the length of the filter increases, thereby slowing down the throughput of the system.

To exploit the parallelism in the FIR computation a non sequential approach using dedicated hardware like an Application Specific Integrated Circuit (ASIC) is also possible. The hardware is replicated based on the length of the filter, thereby allowing a possibility of operating the system at highest possible sampling rate. This is achieved since the entire computation can now be done in one clock cycle, instead of the sequential processing, as is the case with DSPs. Here the architecture could be a full custom solution which aims to optimize the algorithm implementation based on needs of a specific application. The only disadvantage is the lack of flexibility in the hardware once the architecture is frozen. The advent of Field Programmable Gate Arrays (FPGA) in late 1990s allowed the designer to utilize the parallelism in hardware and provide flexibility through reprogramming on the fly [5]. FPGAs are the better choice than DSPs for high data rate applications up to a few Mbps and filters having larger lengths. The only disadvantage is the per unit cost of the FPGA as well as a possibility of overkill in design.

This paper presents the implementation of the velocity estimation module of the sigma 1 controller in hardware. The technical specifications of the design and the signal interfaces are discussed in section II along with the choice for data representation. In section III the data representation and signal flow graph at the algorithmic level for the chosen controller is presented. Section IV discusses the alternate solutions for implementing the MAC operations using parallelism in hardware. The results are presented and analyzed in section V followed by conclusion.

The controller chosen is a variable structure controller of type sigma 1 as discussed in [6]. The formulation of the controller is given in equation (2) which needs the difference between x_2 and x_4, where x_2 is the un-sprung mass velocity and x_4 is the sprung mass velocity.

$$\sigma_1: \quad v(x_2, x_4) = \frac{V_{max}}{2}[sgn(x_4 - x_2) + 1] \tag{2}$$

One possible way of implementing is to sense the velocities using appropriate sensors and calculate the difference. Usually LVDT sensors and accelerometers are used to sense displacement and acceleration respectively, velocity is then estimated or calculated using either of these values. In either case the calculation needs to estimate the velocity and then find the difference. The hardware used thus will be quite complex needing two sensors and two velocity estimators. Instead the digital suspension displacement (x_3) input is used to estimate the velocity, which in turn is the difference in velocities of sprung mass and unsprung-mass ($\dot{x_3} = x_4 - x_2$). This approach reduces the complexity in design and implementation by using only one sensor and one velocity estimator. The controller design favours the use of

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 6, Issue 4, pp: (167-173), Month: October - December 2018, Available at: www.researchpublish.com

displacement sensor since it directly yields the velocity difference. The input to the controller is the suspension deflection and the output is the power signal which feeds the magneto-rheological (MR) damper as shown in Fig. 2.

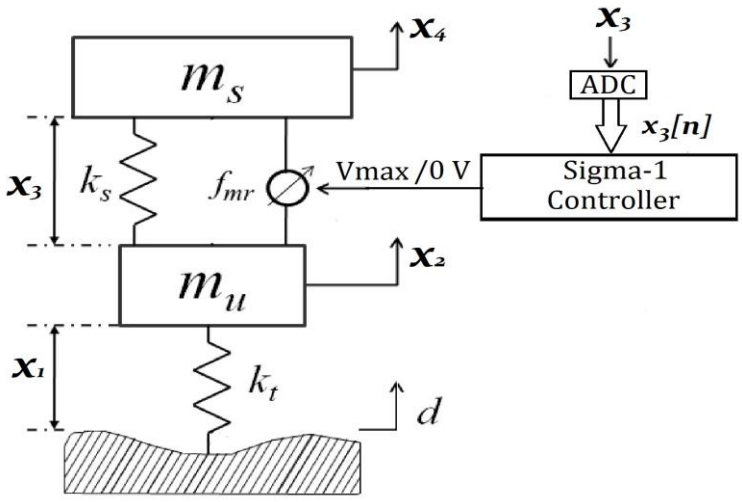

Fig. 2. Semiactive suspension system with the controller

Here d is the road disturbance velocity and x_1 is the tyre displacement. The input to the controller is the sampled digital values of x_3 and the output is the analog voltage to the MR damper, which is either V_{max} or 0V.

II. DESIGN SPECIFICATIONS

System level design and verification precedes the hardware design. The specifications at the hardware level evolve from the system level specifications. The functionality of the blocks is decided at the system level and so are the interfaces between them. Typical specifications to be considered are the word-size, data format, data-rate, resolution, tolerable quantization error and expected range of input and intermediate digital outputs within the velocity estimator.

The internal structure of the semiactive suspension controller is shown in Fig. 3. The decision block either outputs an analog voltage of V_{max} or 0V based on the sign of the velocity difference as given in equation (2).

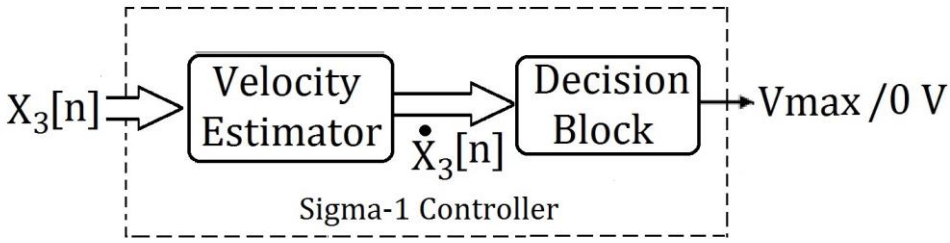

Fig. 3. Signal interfaces in controller

A passenger vehicle is fitted with either two or four of MR dampers with individual controllers, which need to be compact and possibly implement the entire system in a single chip. The controller also needs to be fitted into a vehicle along with interfaces for sensors. Considering all the factors mentioned here it is desirable to have a custom design for semi-active suspension system which allows optimization at all levels of implementation.

The velocity estimator is implemented using a linear-phase FIR filter and the internal architecture is shown in Fig. 4. The data representation has to take into consideration the accumulation in output value because of the MAC operations, but try to retain an acceptable level of accuracy in the intermediate stages of computation. So the possibility of using different precision within the system is explored here.

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 6, Issue 4, pp: (167-173), Month: October - December 2018, Available at: www.researchpublish.com

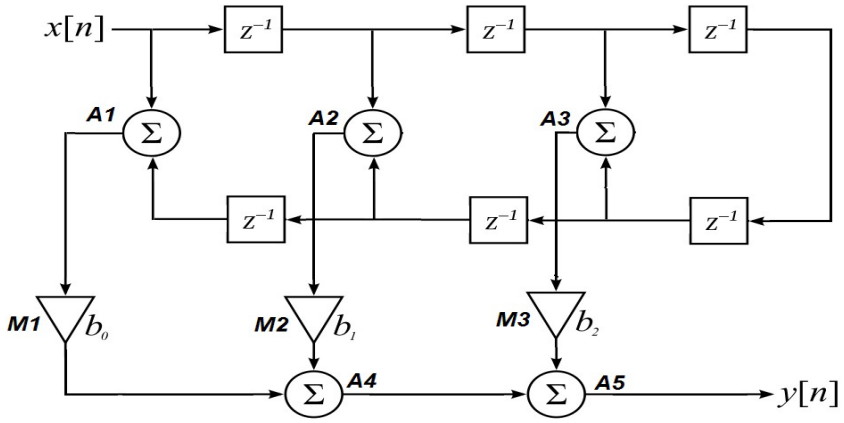

Fig. 4. Optimized architecture for FIR filter of length M=7

III. DATA REPRESENTATION

The data representation is done using either fixed-point or floating-point format. The critical factor in deciding the format is the much greater computational power offered by floating-point DSPs [7]. DSPs using floating-point format are also desirable because of the ease of use since real arithmetic can be implemented directly into floating-point hardware, while arithmetic in fixed-point DSPs needs additional software routines that add to the development time. But floating-point DSPs require complex internal circuitry with twice the size of data paths as compared to fixed-point DSPs thereby increasing the cost of devices. The cost disadvantage is somewhat overshadowed by the advent of System on Chip (SoC) integration of multiple DSPs into a single chip because of miniaturization of transistors. The ease of use advantage is also applicable to a great extent to fixed-point DSPs with advent of efficient C compilers and exceptional tools that provide code-execution visibility. So, the decision between fixed-point and floating-point now essentially boils down to the whether floating-point arithmetic is needed by the data set of the target application. Floating-point DSPs offer greater accuracy through single or double precision formats which use 32 and 64 bit representations respectively as per the IEEE 754 format. The range of values in the data set of a target application and the extent of tolerable errors induced due to rounding-off and data representation decide the format to be used.

In the controller formulation discussed here, the computation of the control algorithm is carefully observed at the system level of validation, where the output of the system swaps between either Vmax or 0V. The ranges of values observed for the suspension displacement are −0.15667 to 0.123313 and for velocity difference are −2.69507 to 3.022631. The precision in binary arithmetic will vary significantly from the precision of decimal arithmetic. The conversion itself will introduce quantization errors into the computation, but that is something which cannot be avoided. The least computation could be done using 8-bit data, but multiplication of two 8-bit data would produce a 16-bit data, and the result has to be rounded off by either trimming the integer portion or truncating the fractional value. Since the design chosen here intends to integrate the entire system onto a single chip without going for a software frontend, a more simple fixed-point arithmetic is preferred. Looking into the range of values expected in the computation and also the accuracy required, two types of fixed point notations are chosen which are Q15 (for internal computation) and Q3.12 (at the output of MAC adders). Q15 data representation gives a range of −1 to 0.999969482421875 (8000h to 7FFFh) and Q3.12 gives a range of −8 to 7.999755859375 (8000h to 7FFFh).

IV. FIR ARCHITECTURES

The data representation is chosen as a 16-bit fixed point format using Q15 and Q3.12 as discussed in section III. Once the data type is chosen it is imperative that the mathematical computations that need to be done for velocity estimator implementation be explored further. The predominant computational unit inside the controller is the velocity estimator implemented as a linear phase FIR filter of length 7 as discussed in [8]. The filter coefficients are in the range ±0.94234994673082473. The optimised architecture that takes into account the anti-symmetry property of the design shown in Fig. 4 depicts the computational requirements of the system. The combinational elements are essentially 16-bit signed add/subtract and multiply, while the sequential elements are the 16-bit registers implementing a 1-bit delay at each clock cycle. Based on the inputs from the system level design the adders in the MAC chain (A4 and A5) are implemented

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 6, Issue 4, pp: (167-173), Month: October - December 2018, Available at: www.researchpublish.com

using Q3.12 format, while all other adders and multipliers use Q15 format. This ensures that the internal data-path is always of 16-bit and is optimised for maximum accuracy and minimum complexity.

The hardware resource requirements for multipliers are much greater than for adders. The area and power requirements differ based on whether the target hardware is an ASIC or a FPGA. In order to minimise the cost of implementation the area needs to be lesser which leads to optimization efforts in this domain. Digit serial systems process multiple numbers of bits every clock cycle and are best suited for applications requiring moderate sample rate, where area and power consumption are critical. The number of add operations required in a constant coefficient multiplication equals one less than the number of non-zero bits in the constant coefficient. The initial design of the FIR filter is implemented using simple multiply and accumulate operations using 16-bit booth multipliers and 16-bit adders. This is the most basic architecture which does not optimize the multiplication operation. This architecture is referred to as 'D1' and it does not include recoding of the filter coefficients using CSD notation.

In order to further reduce the area and power consumption, the constant coefficient can be encoded such that it contains the fewest number of non-zero bits which can be accomplished using "Canonic Signed Digit" (CSD) representation. CSD code is a ternary number system with unique features which make it useful in certain DSP applications focussing on low-power, efficient area and high-speed arithmetic. CSD is a unique way of representation and has two main properties: the number of non-zero digits is minimal, no two consecutive digits are both non-zero which leads to a reduction in the number of additions in arithmetic operations. CSD recoding enables the reduction in the number of partial products during multiplications that must be calculated fast.

A W-bit CSD representation number A when multiplied by a variable X is represented as in equation (3) where the bits a_i, $0 \leq i \leq W-1$ are either 0,1 or -1.

$$A * X = (a_{W-1}a_{W-2} \dots a_1 a_0) * X = \sum_{i=0}^{N-1} a_{W-1-i} 2^{-i} \quad (3)$$

The maximum number of non-zeros in a W-bit CSD number is $(W+1)/2$. Among the W-bit numbers in the range (-1,1) the average number of nonzero bits is $W/3 + 1/9 + O(2^{-W})$. Hence on average, CSD numbers contain about 33% fewer non-zero bits than a conventional 2's complement number representation [9].

The velocity estimator uses the 16-bit CSD representation for the three coefficients b0, b1 and b2. Both the 2's complement and CSD representations are shown in Table I.

TABLE I: FIR FILTER COEFFICIENTS

Coefficient	2's comp	CSD
b_0	0001001101101101	$0001010010\bar{1}0\bar{1}01$
b_1	1100101010111111	$0\bar{1}010\bar{1}0\bar{1}0\bar{1}000000\bar{1}$
b_2	0111100010011111	$1000\bar{1}00010100000\bar{1}$

The reduction in number of additions in the velocity estimator computation depends on the reduction in the number of non-zero bits in the filter coefficient representation which is listed in Table II.

TABLE II: HARDWARE COMPLEXITY REDUCTION IN USING CSD IMPLEMENTATION

Coefficient	Non 0's 2's comp	Non 0's CSD	% Reduction
b_0	8	6	25.00
b_1	11	6	45.45
b_2	10	5	50.00

It is observed that there is an average reduction of 40.15% in the number of non-zeros which is a substantial reduction in the number of additions required during multiplication by these constants. An example of the architecture for performing multiplication of $b_0(x[n] - x[n-6])$ using M1 as in Fig. 4 is shown in Fig. 5.

ISSN 2348-1218 (print)
International Journal of Interdisciplinary Research and Innovations ISSN 2348-1226 (online)
Vol. 6, Issue 4, pp: (167-173), Month: October - December 2018, Available at: www.researchpublish.com

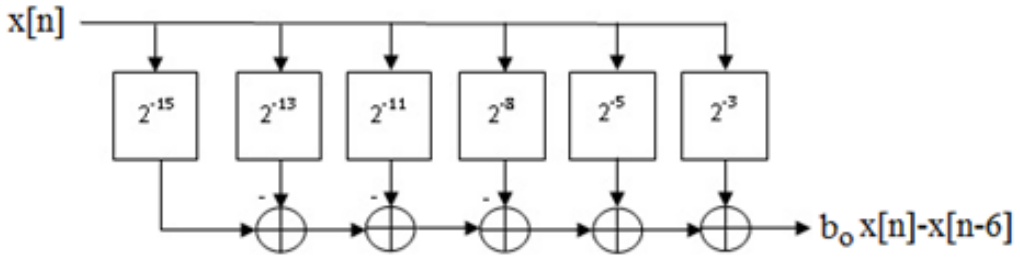

Fig. 5. CSD multiplier architecture

The number of adders and different shifters required for computation of CSD multiplication is given in Table III.

TABLE III: ADDERS AND SHIFTERS NEEDED FOR CSD IMPLEMENTATION

Multiplier	Computes	Adders	Shifters
M1	$b_0(x[n]-x[n-6])$	5	6 (15,13,11,8,5,3)
M2	$b_0(x[n-1]-x[n-5])$	5	6 (15,9,7,5,3,1)
M3	$b_0(x[n-2]-x[n-4])$	4	4 (15,10,8,4)

Here the multiplier M1 requires 5 adders and 6 shifters of varying right shift proportions like 'right-shift by 15' (>>15), 'right-shift by 13' and so on, till 'right-shift by 3' as shown. The reason to indicate separate shifts is because of use of multiple shifters, instead of a single shifter. If one uses a single shifter, then the execution of shift operations become sequential in nature which is not concurrent processing, which was the initial reason to choose custom hardware over programmable DSPs. Also, because of dedicated shifters needed for each sub-operation within a multiplier, these can be hardwired for a particular shift. For eg., a barrel shifter needs a control input of '1111' for a >>15 operation, then instead of using control inputs they can all be tied to logic '1'. Similarly the shift control inputs of all shifters are hard wired. Among the three multipliers (M1, M2, M3) M3 requires the least number of shift and add operations. The architecture which uses barrel shifter with hard-wired control inputs is referred to as 'D2'.

The need for a custom shifter instead of generic or barrel shifter is explored further for optimization of design architecture. A barrel shifter is designed for different magnitudes of left-shift, right-shift or rotate operations. The functionality needed here is just a right-shift of a given fixed amount. Optimizing the circuit to design multiple custom shifters is carried out and a total of 11 unique shifters (1,3,4,5,7,8,9,10,11,13,15) are designed and implemented. The FIR filter that uses this design is referred to as 'D3'.

V. RESULTS AND DISCUSSION

The prototyping of the velocity estimator is done on a FPGA development board using multiple architectures as discussed here. The design D1 is implemented using booth multipliers for computing the output of FIR filter in which the filter coefficients are represented using Q15 notation as standard binary values. In designs D2 and D3, the coefficients are represented using CSD notation and accordingly the multiplication is done using concurrent shift-add algorithms. All shifters in D2 are implemented using the same barrel shifter, while in D3 custom shifters are used for each type of shift. The results of synthesis of all the three architectures are demonstrated in Table IV.

TABLE IV: COMPARISON OF VELOCITY ESTIMATOR ARCHITECTURES

Parameter	D1	D2	D3
Slice registers	244	435	435
Slice LUTs	2302	324	324
LUT FF pairs	2481	541	541
Adders/Subtractors	108	54	54
Latches	119	560	428
Multiplexers	135	960	272
Timing (ns)	31.845	3.877	3.877

The number of adders used up in D1 is 108, while in both D2 & D3 the number is 54. The results show that there is a 50% reduction in the number of adders used for CSD computation as compared to the standard binary representation. Also in terms of the area estimate the number of Registers, LUTs and FFs consumed in D1 is 5027 while in both D2 & D3 it is 1300, which indicates a 74% reduction in resources consumed for CSD implementation. Among D2 & D3 it is observed that the number of latches and multiplexers used in D3 reduces to 700 from 1520 in D2, which is a further reduction of 54% if a custom shifter is used instead of a generic shifter. The timing analysis yields the minimum input interval between two input samples. It is 31.845 ns for D1 while for D2 and D3 it is only 3.877 ns, which means that these two architectures are 7.2 times faster as compared to a conventional binary fixed point multiplier.

The results demonstrate the huge savings in area for CSD implementations as well as huge improvement in performance as compared to standard binary fixed point representations.

VI. CONCLUSION

The design and implementation in FPGA of sigma 1 controller for semiactive suspension is presented in this paper. It is shown that velocity estimator is the primary component in the design which is implemented using a linear phase FIR filter of length 7 for the given specifications. The system approach to implementation uses DSP based approach while it is shown that there are benefits in using custom hardware. Among the custom hardware various architectures are presented and implemented, namely D1, D2 and D3. D1 uses optimized structure of FIR with booth multiplier and standard fixed point notations of Q15 and Q3.12. An alternate architecture to booth multiplier is the shift and add operation, the complexity of which is entirely dependent on the number of non zeros in the input data. Since multiplication by constant coefficients is needed for FIR implementation CSD notation is employed in D2 and D3 which minimises the number of non zeros in the constant multiplier. The results indicate that designs D2 and D3 require 74% lesser area as compared to D1. The implementation using custom shifter (D3) needs only 960 multiplexers as compared to a regular shifter which consumes 272 multiplexers which is a further 71.6% reduction in design complexity. Finally in terms of speed of computation both D2 and D3 are faster by 7.2 times as compared to D1. Further research in this area needs to explore the possibility of using transposed form of FIR structure and replacing half the multipliers with NOT gates by exploiting the anti-symmetry in design of linear-phase FIR filter.

REFERENCES

[1] S. Eligar and R. M. Banakar, “A survey on passive, active and semiactive automotive suspension systems and analyzing tradeoffs in design of suspension systems,” International Conference on Recent Innovations in Electrical, Electronics and Communication Engineering, 2018.

[2] G. Koch, E. Pellegrini, S. Spirk, and B. Lohmann, “Design and modeling of a quarter-vehicle test rig for active suspension control,” Tech. Rep., Lehrstuhl f¨ur Regelungstechnik, 2010.

[3] X. Dong, M. Yu, C. Liao, and W. Chen, “Rapid control prototyping development of vehicle semi-active control scheme,” 2007.

[4] R. Zatrepalek, “Using FPGAs to solve tough DSP design challenges deciding between traditional DSP and FPGA,” 2012.

[5] D. Nenni, “A brief history of FPGAs,” 2012.

[6] S. Eligar and R. M. Banakar, “A model based approach for design of semiactive suspension using variable structure control,” International Journal of Technical Research and Applications, pp. 2320–8163, 2014.

[7] G. Frantz and R. Simar, “Comparing fixed and floating point DSPs,” Texas Instruments, Dallas, TX, USA, 2004.

[8] S. Eligar and R. M. Banakar, “Optimization of control algorithm for semi-active suspension system,” International Conference on Intelligent Computing and Sustainable System, 2018.

[9] K. K. Parhi, “VLSI digital signal processing systems: design and implementation,” John Wiley & Sons, 2007.