Annals of Emerging Technologies in Computing (AETiC)
Vol. 5, No. 1, 2021

Research Article

A Customized Floating-point Processor Design for FPGA and ASIC based Thermal Compensation in High-precision Sensing

Muhammad Sajjad[1*], Mohd Zuki Yusoff[1] and Muhammad Ahmed[2]

[1]CISIR, Universiti Teknologi PETRONAS, Perak, Malaysia
muhammad_19001746@utp.edu.my; mzuki_yusoff@utp.edu.my
[2]National University of Computer and Emerging Sciences, Islamabad, Pakistan
m_ahmed_17@yahoo.com
*Correspondence: muhammad_19001746@utp.edu.my

Received: 18th November 2020; Accepted: 27th December 2020; Published: 1st January 2021

Abstract: There are many types of sensors which require large dynamic range as well as high accuracy at the same time. Barometric altimeter is an example of such sensors. The signal processing techniques in the sensors are normally implemented using Field Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs). The sensing variable in such type of the sensors is unwantedly environment dependent. So, for ensuring accuracy of the sensors this environmental dependency is minimized using the modeling and compensation techniques. In this work we have proposed a digital architecture for a programmable high precision computational unit which can be implemented in the FPGA or ASIC running the sensing algorithm of the sensors. This architecture can be used to implement polynomial compensation and it also supports reading and writing of the corresponding calibration coefficients even after the development of the sensors. Moreover, the architecture is platform independent. The architecture have been simulated for different FPGAs and ASIC and it has fulfilled the speed, accuracy and programmability requirements of the type of the sensors. The architecture has also been implemented and verified in a prototype of the barometric pressure sensor on Spartan-6 FPGA.

Keywords: ***ASIC; digital signal processing; double-precision floating point unit; FPGA; high precision sensors; thermal compensation; VerilogHDL***

1. Introduction

Nowadays, for development of many high-precision sensors, FPGA is used to meet the requirements of the digital signal processing. FPGA is an ideal choice for the sensors demanding deterministic latency for synchronizing different events and flexibility in the use of input and output pins [1-2]. Performance of many high-precision sensors is dependent upon environmental factors such as temperature, pressure, and humidity. One way to eliminate the error introduced by the dependency is to model or predict the error with the environmental factors and then compensate it by further processing the output of the sensors [3-5]. In most of the cases, the suitable model is a polynomial function in some environmental factors. As a result, the implementation of a polynomial function in digital signal processing of the high-precision sensors is required [6]. Let's discuss the available suitable approaches that can be used to achieve this goal.

Muhammad Sajjad, Mohd Zuki Yusoff and Muhammad Ahmed, "A Customized Floating-point Processor Design for FPGA and ASIC based Thermal Compensation in High-precision Sensing", *Annals of Emerging Technologies in Computing (AETiC)*, Print ISSN: 2516-0281, Online ISSN: 2516-029X, pp. 40-50, Vol. 5, No. 1, 1st January 2021, Published by International Association of Educators and Researchers (IAER), DOI: 10.33166/AETiC.2021.01.004, Available: http://aetic.theiaer.org/archive/v5/v5n1/p4.html.

Firstly, electronics of the sensors contain a microcontroller unit (MCU) or a digital signal processor (DSP) in addition to FPGA. Implementing a polynomial function in the MCU or DSP is a straightforward process because high-level languages are used to program them [7-8]. But such hardware adds complexity and cost to the sensors' development. Secondly, a soft microprocessor core such as MicroBlaze can be embedded in FPGA for implementing the polynomial function [9-11]. MicroBlaze core consumes a large number of resources in FPGA and does not meet the speed and precision requirements in real-time for some sensors. Thirdly, a system on chip (SoC), containing FPGA and processor cores on a single chip, can also be used for this purpose but it is again a costly solution and overkills the problem [12-15]. Fourthly, customized computational units can be designed in FPGA using hardware description language (HDL) [16-17]. Such units normally incorporate different IP cores and their HDL source code cannot be exported directly to other FPGAs, ASICs and integrated software environments (ISEs). So, reusability of the source codes for the high-precision sensors is compromised which is an important feature in any industry.

In order to solve the above issues, a new approach is proposed in [18] to design a computational unit for the high-precision sensors' development. Some procedures are presented in the work to design the hardware capable of performing mathematical and typecasting operations without using an IP core, but the concept is not verified for practical systems in real-time. In this work we extend and verify the platform independency of the design and evaluate the design in real-time using a prototype system. The prototype system consists of a barometric pressure sensor interfaced with a Xilinx Spartan-6 FPGA which is a low cast commercially available device. The functionality of the calibration feature is verified in real-time. So, we can conclude that the design is suitable for incorporating it in practical systems.

The paper is organized in IX sections. In section II environmental dependency of the high-precision sensors is described mathematically. Double-precision floating point unit capable of performing required floating-point operations in IEEE 754 format is described in section III. Section IV and V of the paper comprises the datapath and the control unit of the processor respectively. In section VI, instruction set architecture for the computational unit is described. While in section VII, SPI flash interfacing is described to store required data in RAMs of computational unit. Results and details of simulation and implementation of the design are described in section VIII. Finally, the conclusion of the research activity is presented in section IX.

2. Calibration of High Precision Sensors

High Precision Sensors' output depends not only on sensing variable but also on different parameters such as scaling factor and offset. The calibration of such sensors is required to get the correct value from the sensor. Equation (1) and (2) describes the relation between the sensor's output and sensing variable with these parameters.

$$S_{out} = KV_a + B \quad (1)$$

$$V_a = K_s(S_{out} - B) \quad (2)$$

S_{out} : Output of the Sensor

V_a : Sensing variable

K_s : Scaling factor of the Sensor

B : Offset of the Sensor

The sensor algorithm requires information about scaling factor and offset to get the correct value of sensing variable. Offset is found by giving a small value at the sensor's input. When V_a is small, K_sS_{out} would be small and negligible and output only depends on the B. In this way, offset is found. Similarly, the scaling factor is found by giving large input to the sensor. When V_a is larger, the term (S_{out} - B) would approximately equal to S_{out} and output depends only on K_s. Now we can find sensing variable V_a by using calculated scaling factor and offset in (2).

The problem with most of the high precision sensors is that scaling factor and offset depends upon different environmental variables such as the temperature. In order to get the correct value from sensors, the behavior of the scaling factor and the offset must be modeled according to the required environmental variable. The relation between sensors' parameters and environmental variables is found by acquiring and processing sensors' data against the environmental variable in MATLAB. Then the predicted model is implemented in the system to get compensated output of the sensor against the environmental variable.

The general relation between scaling factor and temperature is described in (3) and offset and temperature is described in (4).

$$\mathrm{K_{sf}(n)} = \{1 + \mathrm{a_1\, T(n)} + \mathrm{a_2\, T^2}(n) \ldots + \mathrm{a_n\, T}^n(n)\} \tag{3}$$

$$\mathrm{B(n)} = \{1 + \mathrm{b_1\, T(n)} + \mathrm{b_2\, T^2}(n) \ldots + \mathrm{b_n\, T}^n(n)\} \tag{4}$$

$\mathrm{T(n)}$	:	Temperature that needs to be compensated
$\mathrm{K_{sf}(n)}$	:	Scaling factor equation
$\mathrm{B(n)}$	:	Offset Equation
$\mathrm{a_i, b_i}$	:	Coefficients

The scaling factor and the offset equations must be computed in real time to compensate changes occurred due to environmental variables in the sensors' data. Also, the equations are implemented using Double-Precision IEEE-754 format because it offers high precision and range as compared to single-precision format and fixed-point format [19]. The temperature sensor used for thermal modeling gives data in fixed point format so it must be converted into the double-precision format. Then, after solving a required polynomial in double-precision format, the output must also be converted into the fixed-point format. Therefore, the equations shown here required a minimum of four operations for its computations that include Fixed to Float Conversion, Double-precision Addition, Double-precision Multiplication, and Float to Fixed Conversion. The floating-point unit described in the next section is designed to solve these floating-point operations.

3. Floating-point Unit (FPU)

The floating-point unit of the computational unit is a double-precision IEEE-754 compliance integrated unit. It can perform four different floating-point operations such as fixed to float conversion, float to fixed conversion, floating-point addition (64-bit) and floating-point multiplication (64-bit). A 2-bit operation code is given to FPU to select the operation to perform. The enable signal is used to trigger the FPU to start the required operation.

IP cores are available to solve these required floating-point operations in Xilinx-ISE tool. But we have targeted our computational unit for ASIC design so IP cores cannot be feasible. Therefore, all the four floating-point operations are implemented using algorithmic state machine (ASM) using VerilogHDL to get complete hardware description of these operations.

Each floating-point operation performed by the FPU takes multiple cycles to complete its operation. Therefore, a handshaking system is developed between the control unit and the FPU so that control unit can keep track of the FPU busy status using FPU ready signal. A fixed-point multiplier is also used to perform the floating –point multiplication [20].

The floating-point addition involves a step by step procedure that is implemented using ASM. The complete procedure for the addition of given two double-precision floating-point numbers such as $\mathrm{A_{DBL}}$ and $\mathrm{B_{DBL}}$ is described in Fig. 1. The floating-point multiplication also involves different steps that are described in Fig. 2. The required steps for converting the fixed-point number into its equivalent floating-point number are shown in Fig. 3. The complete procedure required for converting a floating-point number into a fixed-point number is described in Fig. 4. The floating-point unit described in this section is used with datapath of the computational unit to perform the required operations for polynomial solving. The architecture of the datapath with its peripherals is described in the next section.

4. Datapath Design

The datapath of the computational unit is designed to control the data operations commanded by the control unit. The datapath contains features of Harvard Architecture in which separate memories for instructions and data are used. Also, all the instructions in datapath are executed in multiple clock cycles. Therefore, a handshaking system is developed between the control unit and datapath to ensure successful execution of each instruction by the FPU. FPU ready signal in datapath serves as a feedback signal between the FPU and the control unit.

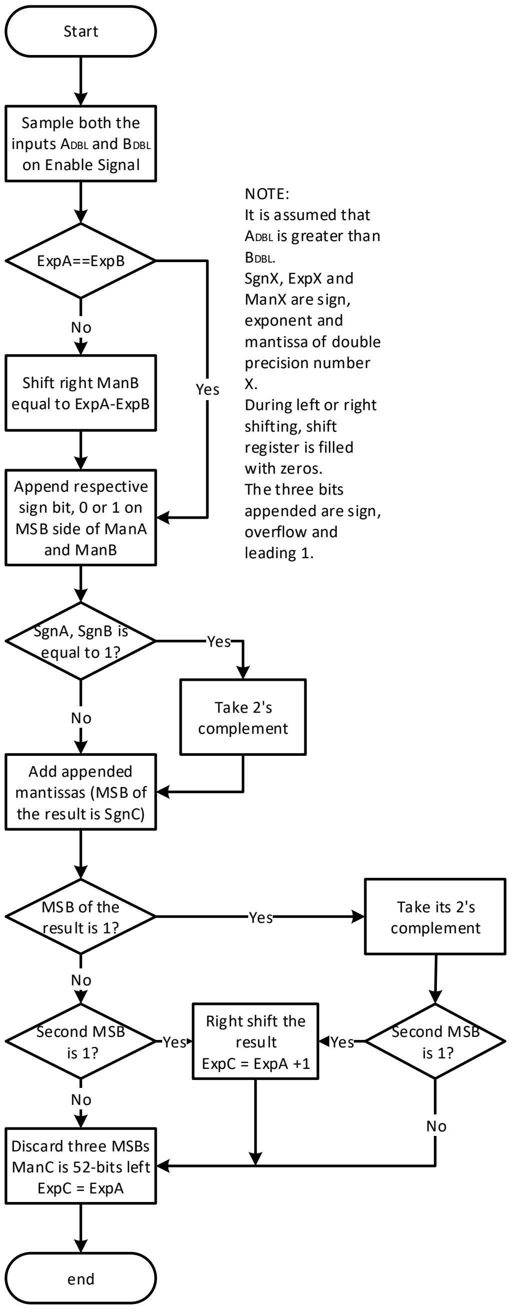

Start
Sample both the inputs ADBL and BDBL on Enable Signal
ExpA==ExpB
No
Yes
Shift right ManB equal to ExpA-ExpB
Append respective sign bit, 0 or 1 on MSB side of ManA and ManB
NOTE: It is assumed that ADBL is greater than BDBL. SgnX, ExpX and ManX are sign, exponent and mantissa of double precision number X. During left or right shifting, shift register is filled with zeros. The three bits appended are sign, overflow and leading 1.
SgnA, SgnB is equal to 1?
Yes
No
Take 2's complement
Add appended mantissas (MSB of the result is SgnC)
MSB of the result is 1?
Yes
Take its 2's complement
No
Second MSB is 1?
Yes
Right shift the result ExpC = ExpA +1
Yes
Second MSB is 1?
No
No
Discard three MSBs ManC is 52-bits left ExpC = ExpA
end

Figure 1. Double-precision Floating-point Addition

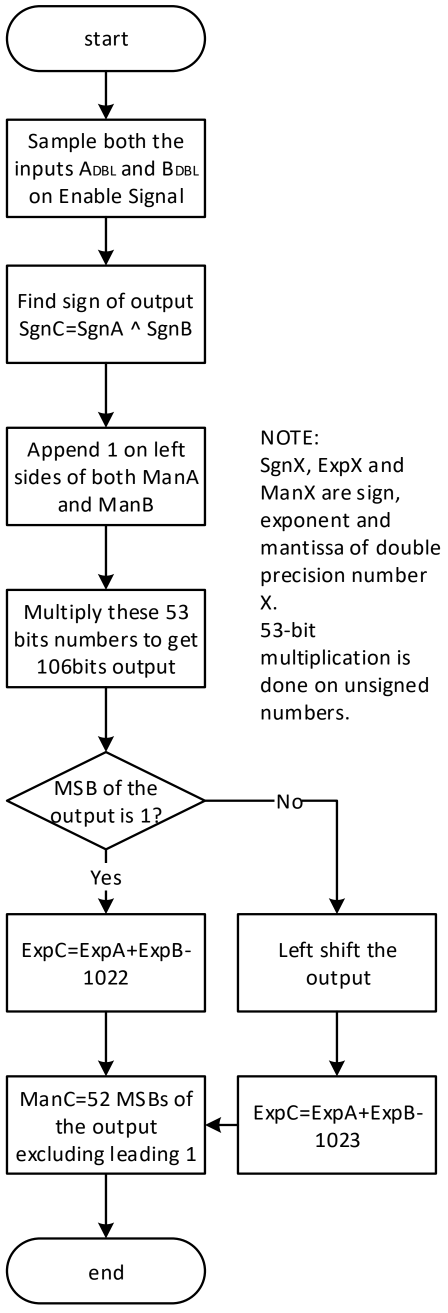
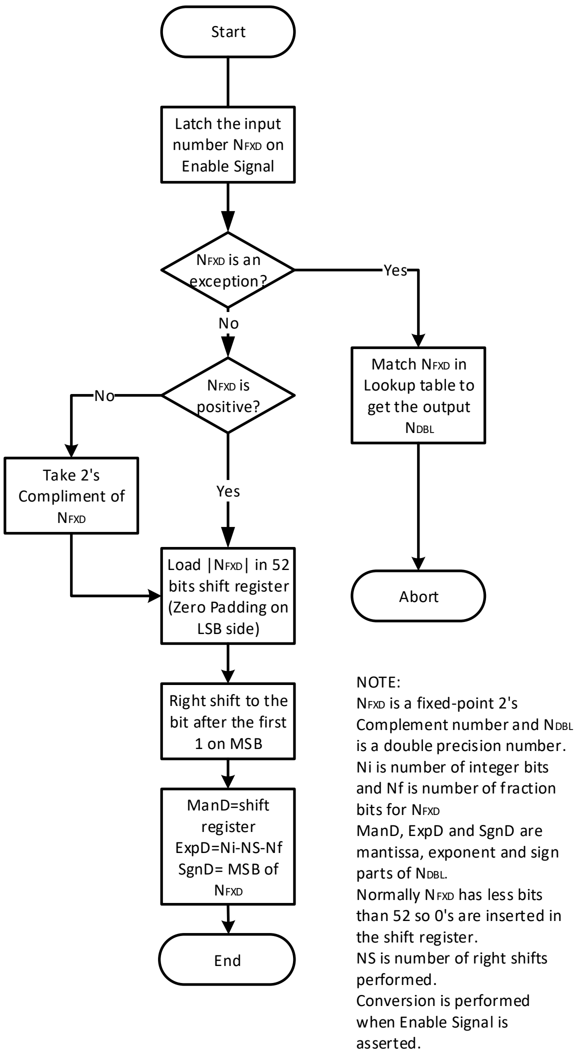

start
Sample both the inputs ADBL and BDBL on Enable Signal
Find sign of output SgnC=SgnA ^ SgnB
Append 1 on left sides of both ManA and ManB
NOTE: SgnX, ExpX and ManX are sign, exponent and mantissa of double precision number X. 53-bit multiplication is done on unsigned numbers.
Multiply these 53 bits numbers to get 106bits output
MSB of the output is 1?
No
Yes
ExpC=ExpA+ExpB-1022
Left shift the output
ManC=52 MSBs of the output excluding leading 1
ExpC=ExpA+ExpB-1023
end

Figure 2. Double-precision Floating-point Multiplication

The datapath consists of following peripherals such as program memory, instruction memory, instruction register, program counter, an output register, and a multiplexer. The complete datapath for the computational unit processing is shown in Fig. 5 and Fig. 6. Program Memory for Register File is a 64-bit dual port RAM divided into two parts. The upper part of the RAM is used to store coefficients required in the solving of the polynomial while the lower part is used to store the results of the instructions to use it in further instructions. Instruction Memory is an n-bit single port RAM used to store instructions required by the computational unit. The address of this RAM is controlled by the control unit to execute given instructions sequentially.

Program Counter (PC) is an n-bit counter that controls the address bus of the instruction memory. PC is incremented by the control unit when an instruction is successfully executed by the processor. It is reset when all the instructions for the required polynomial are executed by the computational unit. The fetched instruction is decoded into different control signals that are required in the datapath. These control signals are stored in the instruction register for the use of the datapath. Output Register is used in the datapath to store the output of the solved polynomial so that it can be used by the sensing algorithm for further processing. Multiplexer shown in the datapath is used to get data from the required sensor using its Select Lines. The selected sensor data is passed to the FPU to perform the required operation for the given instruction. The datapath needs different control signals to execute given instruction that are described in Table 1. These control signals are provided by the control unit according to the given instructions.


Start
Latch the input number NFXD on Enable Signal
NFXD is an exception?
Yes
No
Match NFXD in Lookup table to get the output NDBL
Abort
NFXD is positive?
No
Yes
Take 2's Compliment of NFXD
Load |NFXD| in 52 bits shift register (Zero Padding on LSB side)
Right shift to the bit after the first 1 on MSB
ManD=shift register ExpD=Ni-NS-Nf SgnD= MSB of NFXD
End
NOTE:
NFXD is a fixed-point 2's Complement number and NDBL is a double precision number.
Ni is number of integer bits and Nf is number of fraction bits for NFXD
ManD, ExpD and SgnD are mantissa, exponent and sign parts of NDBL.
Normally NFXD has less bits than 52 so 0's are inserted in the shift register.
NS is number of right shifts performed.
Conversion is performed when Enable Signal is asserted.

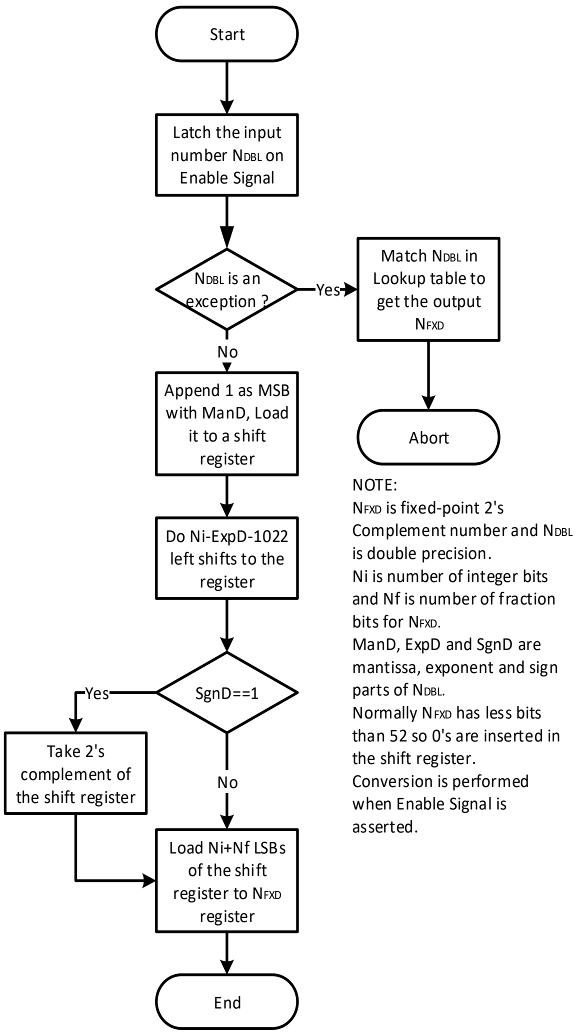

Start
Latch the input number NDBL on Enable Signal
NDBL is an exception ?
Yes
Match NDBL in Lookup table to get the output NFXD
Abort
No
Append 1 as MSB with ManD, Load it to a shift register
Do Ni-ExpD-1022 left shifts to the register
SgnD==1
Yes
Take 2's complement of the shift register
No
Load Ni+Nf LSBs of the shift register to NFXD register
End
NOTE:
NFXD is fixed-point 2's Complement number and NDBL is double precision.
Ni is number of integer bits and Nf is number of fraction bits for NFXD.
ManD, ExpD and SgnD are mantissa, exponent and sign parts of NDBL.
Normally NFXD has less bits than 52 so 0's are inserted in the shift register.
Conversion is performed when Enable Signal is asserted.

Figure 3. Flow Chart for Fixed-to-Double Conversion

Figure 4. Flow Chart for Double-to-Fixed Conversion

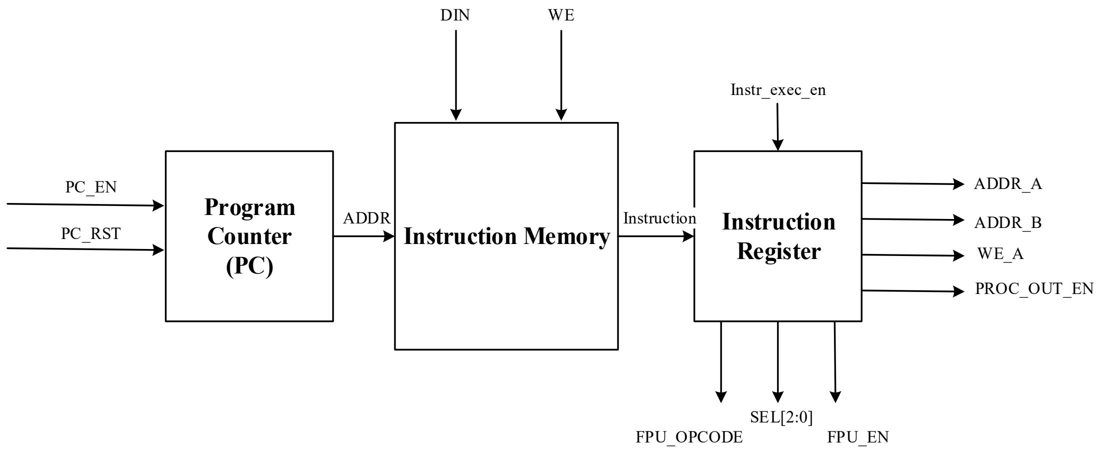


Figure 5. Datapath of the Processor (A)

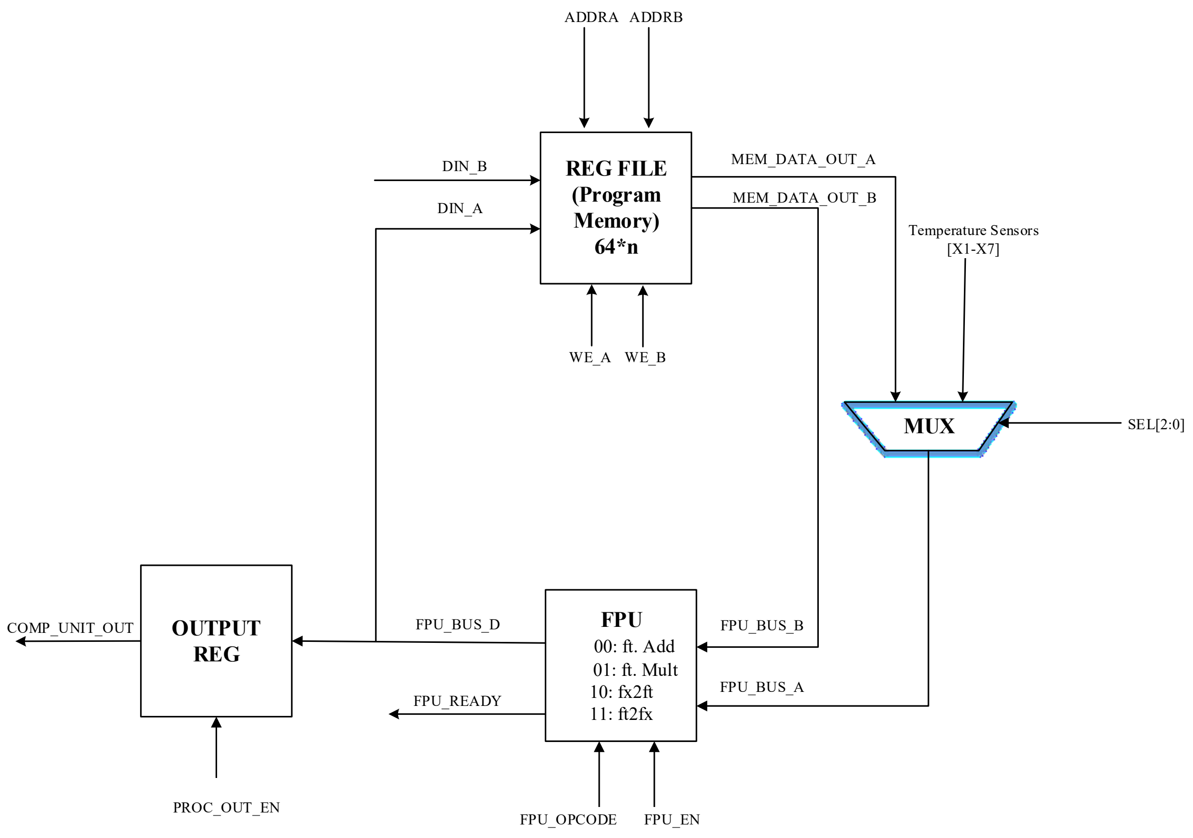


Figure 6. Datapath of the Processor (B)

5. Control Unit

The control unit of the processor controls the datapath, memory and I/O according to the program instructions stored in the instruction memory. FSM is implemented in the control unit to control required signals by datapath. FSM consists of 6 states such as memory configuration, idle, fetch, decode, execute and instruction complete.

Table 1. Control Signals

No.	Control Signals	Description
I	ADDR_A	Address of Bus A of Register File
II	ADDR_B	Address of Bus B of Register File
III	FPU_EN	Enable Signal for FPU
IV	FPU_OP	Operation Code for FPU
V	MUX_SEL	Select Lines of MUX for selecting data from different sensors
VI	PROC_OUT_EN	Enable Signal for output register
VII	WEA, WEB	Write enable signals for Register File Bus A & B
VII	PC_EN	Enable Signal for Program Counter
IX	PC_RST	Reset Signal for Program Counter
X	INSTR_EXEC_EN	Enable Signal for Instruction Register

1. CONFIG_MEM: Control unit waits for the loading of instructions and coefficients from the SPI Flash in the instruction memory and the register file respectively.
2. IDLE: When memories configuration is completed, control unit waits in this state for the external trigger to start solving polynomial according to given instructions.
3. FETCH: The control unit fetches the instruction from the instruction memory.

4. DECODE: Fetched instruction is decoded into different control signals required by the datapath. Here, the control unit also checks for the END PATTERN. If the end pattern is found in the instruction, it goes to INSTR_COMP state to load result of the calculated polynomial on the output bus. End pattern is a special pattern in the program instruction that shows all the required instructions for polynomial solving are executed by the processor.
5. EXECUTE: Here, data from the register files or temperature sensor is passed to the FPU to perform the desired operation. On completing the required operation, instruction result is written at the given destination address in the register file.
6. INSTR_COMP: Here, the polynomial result is loaded in output register to load on Data out bus of the processor.

The control signals required by the datapath are generated according to fetched instruction. Each instruction represents an FPU operation required in the target polynomial. The instruction used here consists of a specific format that is explained in section 6.

6. Instruction Set Architecture

The instructions used in the processor consist of 5 different fields that are described in Table 2. The size of each field can be selected according to the memories using in the processor and the number of temperature sensors required in thermal modeling of the designed sensor. The coefficients and instructions required for polynomial are stored in an attached SPI flash. The interfacing between SPI flash and the computational unit is described in the next section.

7. SPI Flash Interfacing

The SPANSION SPI flash S25FL256S is interfaced with the computational unit to store coefficients and instructions. The typical hardware architecture for interfacing SPI Flash with the designed computational unit is shown in Fig. 7. The computational unit can be implemented on FPGA/ASIC depending upon the requirement. SPI flash is attached with FPGA/ASIC to store data required by the computational unit. Data stored in SPI flash can be modified from PC using UART interface. RS-232/RS-422 IC is used here to translate voltage levels required by FPGA/ASIC.

Table 2. Instruction Fields

Instruction Field	Description
MUX_SEL	Select lines for the multiplexer to select data from the required sensor or Register File to pass it to the FPU.
FPU_OPCODE	Operation code for FPU to select required floating-point operation.
SRC-ADDR-1	Source Address for BUS-A of the Register File.
SRC-ADDR-2	Source Address for BUS-B of the Register File.
DEST-ADDR	Destination Address at which result of the executed instruction is to store.

By default, SPI Flash will be in reading mode and the data will be transferred from flash to the respective RAMs of the processor. When it is required to modify data, SPI Flash mode can be changed using flash_op signal. Now user can write data in SPI flash using UART Interface. When flash_op = 0, the flash is in read state and when flash_op = 1, the flash is in write state.

1. *SPI Flash Reading Module:*

SPI interface is implemented to read data from flash and store it in their respective RAMs. First of all, it read coefficients from flash one by one and stored it in REGFILE using its BUS B. Then, it read instructions and stored it in Instruction Memory. The connectivity between the SPI Flash reading module and processor RAMs is shown in Fig. 8.

2. *SPI Flash Writing Module:*

For modifying coefficients and instructions stored in the SPI Flash, first of all, data will be read from PC using UART Interface and then it will be write-in the SPI flash. So, two different types of modules are designed here.

- UART Receiver: To receive data from PC using UART interface
- SPI Flash Writing: To write data received from PC on SPI Flash

Handshaking signals are developed between UART Receiver and SPI Flash Writing Modules to maintain synchronization between the data. As soon as a byte is received from PC using UART receiver module, it is written on the SPI flash using SPI flash writing module.

8. Software Simulation and Hardware Results

The typical thermal modeling equation for finding compensated output of the sensor is shown in (5). The values of the scaling factor and the coefficients can be found using offline analysis of the sensors' data and the temperature sensors' data with the help of MATLAB software. The nominal coefficients for a sensor are shown in Table 3.

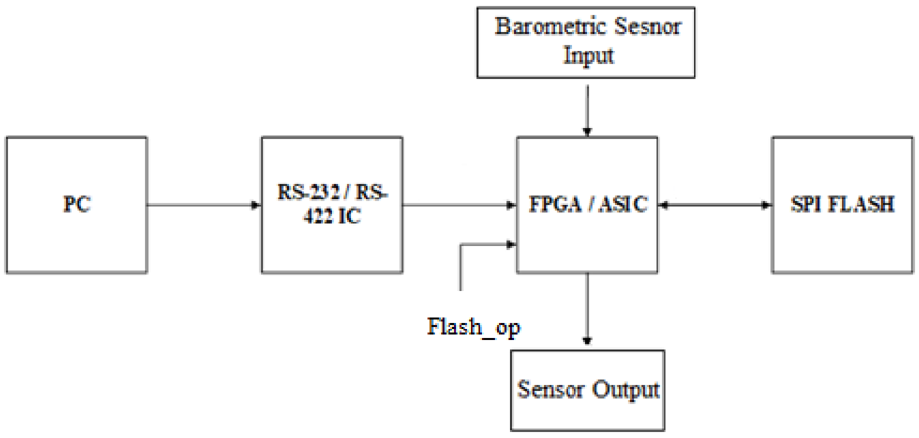


Figure 7. SPI Flash Interface with designed system

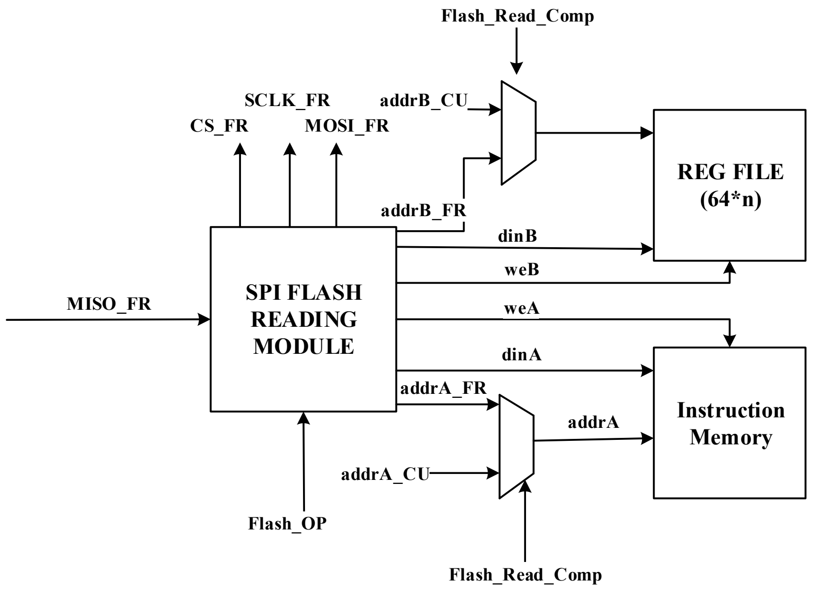


Figure 8. Dataflow between Flash Reading Module, Register File and Instruction Memory

$$B_{cmp} = K_{sf}B_{raw} + a_1T_1 + a_2T_2 + a_3G_1 + a_4G_2 \quad (5)$$

B_{raw}	:	Uncompensated or raw offset of the sensor (uncompensated output)
$T_{1,2}$	:	Temperature Sensors (1, 2)
$G_{1,2}$	:	Gradients of Temperature Sensors (1, 2)
K_{sf}	:	Scaling Factor
a_i	:	Coefficients of temperatures and Gradients (1, 2, 3, 4)

Table 3. Coefficients Values

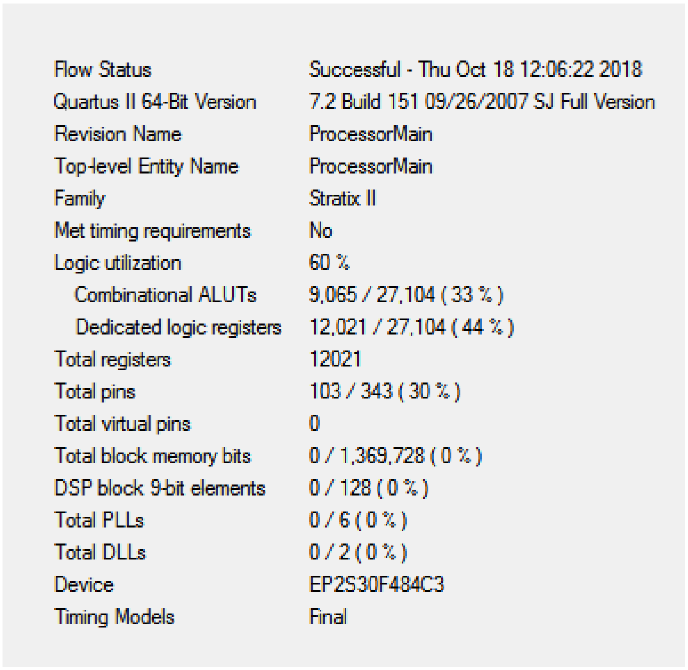
Coefficients	**Values**
K_{sf}	1
a_1	0.10
a_2	2.00
a_3	0.23
a_4	2.40

The polynomial described in (5) is implemented with the help of described computational unit using Xilinx Spartan 6 device. It is simulated and verified using Xilinx-ISE Post-Layout Simulation Tool. The computational unit was also simulated on the SimVision tool for ASIC implementation. The designed computational unit is flexible enough and can be implemented on any FPGA and ASIC tool. The design is synthesized on Xilinx-ISE tool and Quartus-II tool for FPGA implementation. It is also verified with Cadence Encounter tool for ASIC design implementation. The resources utilization summary for the Spartan6 device in Xilinx ISE tool is described in Table 4. Similarly, the Quartus II tool result for this design is shown in Fig. 9. Table 5 described the gate count results for ASIC implementation of the design in Cadence Encounter tool.

Table 4. Resources Utilization Summary for Spartan-6 Device

Logic Components	Utilization
Number of Slice Registers	10,106
Number of Slice LUTs	14,389
Number of occupied Slices	5,287
Number of MUXCYs used	3,796
Number of LUT Flip Flop pairs used	15,684

The results of the sensor with or without temperature compensation are discussed with MATLAB in Fig. 10. The uppermost graph describes the behaviour of the temperature sensors. The graph shown in the middle represents the gradients of temperature sensors. The output of barometric pressure sensor or altimeter is described with and without thermal modelling in the last graph. From these graphs, it can be verified that temperature sensors and the gradients of the temperature sensors doesn't have any effect on the compensated output of the sensor. It can be concluded from these graphs that thermal modelling of temperature sensors in (5) has removed errors from the compensated output of sensor. The thermal modelling or compensation has also been verified for the pressure sensor by keeping it in a temperature varying setup.



Flow Status	Successful - Thu Oct 18 12:06:22 2018
Quartus II 64-Bit Version	7.2 Build 151 09/26/2007 SJ Full Version
Revision Name	ProcessorMain
Top-level Entity Name	ProcessorMain
Family	Stratix II
Met timing requirements	No
Logic utilization	60 %
Combinational ALUTs	9,065 / 27,104 (33 %)
Dedicated logic registers	12,021 / 27,104 (44 %)
Total registers	12021
Total pins	103 / 343 (30 %)
Total virtual pins	0
Total block memory bits	0 / 1,369,728 (0 %)
DSP block 9-bit elements	0 / 128 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2S30F484C3
Timing Models	Final

Figure 9. Synthesis Report of the design in the Quartus II

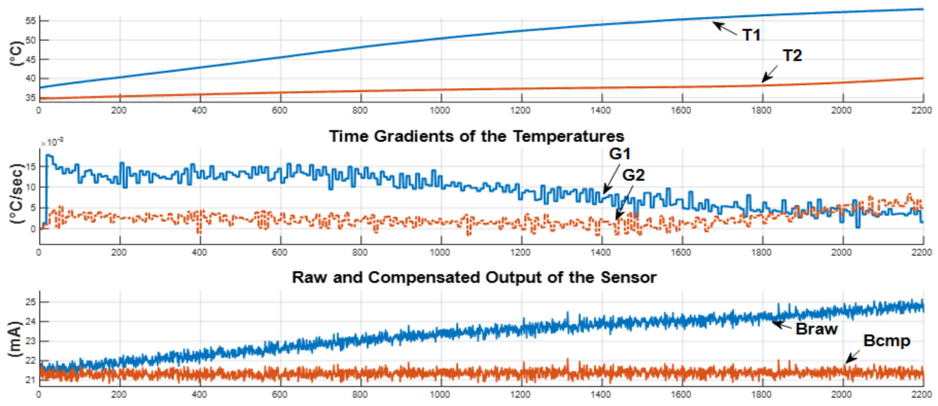


Figure 10. Thermal Compensation Results

Table 5. Resources Utilization Summary for ASIC Implementation

Gates	131805
Cells	36135
Area	6975155.9 ($\boldsymbol{um^2}$)

9. Conclusion

The paper has presented the design of a computational unit in FPGA or ASIC to enhance the performance of the high-precision sensors by modelling environment induced errors. The design of the unit features reusability and full programmability to facilitate the development and calibration of the sensors. The design is implemented using VerilogHDL. The reusability of the design is verified by simulating it in different ISEs for FPGA and ASIC development. Precision and programmability of the design are verified by implementing it in Spartan-6 FPGA. Performance of the sensors is equivalent in Real-time and offline compensation of the sensor output. So, the computational unit meets the requirements of the high-precision sensors in the industry.

Acknowledgement

This work is supported by the Graduate Assistantship Scheme of Universiti Teknologi PETRONAS, and in part by the YUTP Fund under Grant 015LC0-239.

References

[1] A. De La Piedra, A. Braeken and A. Touhafi, "Sensor systems based on FPGAs and their applications: A survey", *Sensors,* vol. 12, no. 9, pp. 12235-12264, 2012.

[2] G. J. García, C. A. Jara, J. Pomares, A. Alabdo, L. M. Poggi and F. Torres, "A survey on FPGA-based sensor systems: towards intelligent and reconfigurable low-power sensors for computer vision, control and signal processing", *Sensors,* vol. 14, no. 4, pp. 6247-6278, 2014.

[3] M. V. Gheorghe, "Advanced calibration method for 3-axis MEMS accelerometers", in *2016 International Semiconductor Conference (CAS)*: IEEE, pp. 81-84, 2016.

[4] R. Mijarez, D. Pascacio, R. Guevara and J. Rodriguez, "Signal processing algorithm for thermal drift compensation in high-temperature down-hole instrumentation systems", *Transactions of the Institute of Measurement and Control,* vol. 39, no. 8, pp. 1161-1168, 2017.

[5] A. R. Buzdar, A. Latif, L. Sun and A. Buzdar, "FPGA prototype implementation of digital hearing aid from software to complete hardware design", *International Journal of Advanced Computer Science and Applications (IJACSA),* vol. 7, no. 1, 2016.

[6] D. Kumbhar and K. S. Bodani, "Designing hardware architecture for polynomial matrix multiplications", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, vol. 4, no. 3, pp. 1451-1454, 2015.

[7] C. Stehning and G. A. Holst, "DSP-based measuring system for temperature-compensated fiber optical oxygen sensors", in *Fiber Optic Sensor Technology and Applications,* vol. 4578: International Society for Optics and Photonics, pp. 259-270, 2002.

[8] M. Tomasi, S. Pundlik and G. Luo, "FPGA–DSP co-processing for feature tracking in smart video sensors", *Journal of Real-Time Image Processing,* vol. 11, no. 4, pp. 751-767, 2016.

[9] V. Kale, "Using the microblaze processor to accelerate cost-sensitive embedded system development", *White Paper: MicroBlaze™ Embedded Processor, WP469 (v1. 0.1)*, 2016.

[10] J. Seely, S. Erusalagandi and J. Bethurem, "The MicroBlaze Soft Processor: Flexibility and Performance for Cost-Sensitive Embedded Designs", Technical Report, Xilinx, 2017. Accessed on: Dec. 15, 2020, Available: https://china.xilinx.com/support/documentation/white_papers/wp501-microblaze.pdf.

[11] B. Wajszczyk, "Analysis of using a MicroBlaze processor for hardware implementation of algorithms for data processing in electronic recognition devices and systems based on the example of a XILINX FPGA system", in XII Conference on Reconnaissance and Electronic Warfare Systems: International Society for Optics and Photonics, 2019, doi: doi.org/10.1117/12.2525056.

[12] B. David,D. V. Julien, H. Cédric, B. François, D. François *et al.*, "A 25MHz 7µW/MHz ultra-low-voltage microcontroller SoC in 65nm LP/GP CMOS for low-carbon wireless sensor nodes", in *IEEE International Solid-State Circuits Conference 2012*, pp. 490-492, 2012.

[13] L. H. Crockett, R. A. Elliot, M. A. Enderwitz and R. W. Stewart, "Applications and Oppertunities" in The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC. Glasgow: University of Strathclyde Academic Media, 2015. ch. 5, pp. 101–130.

[14] H. Y. Jie, T. T. Hsuen, L. T. Wei, H. C. Wei, Y. P. Wen *et al.*, "A self-powered CMOS reconfigurable multi-sensor SoC for biomedical applications", *IEEE Journal of Solid-State Circuits*, vol. 49, no. 4, pp. 851-866, 2014.

[15] V. Kathail, J. Hwang, W. Sun, Y. Chobe, T. Shui and J. Carrillo, "SDSoC: A higher-level programming environment for Zynq SoC and Ultrascale+ MPSoC", in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 4-4.

[16] M. Amiri, F. M. Siddiqui, C. Kelly, R. Woods, K. Rafferty, and B. Bardak, "FPGA-based soft-core processors for image processing applications", *Journal of Signal Processing Systems*, vol. 87, no. 1, pp. 139-156, 2017.

[17] S. Athar, M. A. Siddiqi and S. Masud, "Teaching and research in FPGA based digital signal processing using Xilinx system generator", in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012: IEEE, pp. 2765-2768.

[18] M. Sajjad, M. bin Zuki Yusoff and M. Ahmed, "Design of Double-Precision Fully-Programmable Computational Unit for FPGA and ASIC", in *2020 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, 2020: IEEE, pp. 21-26.

[19] V. Rajaraman, "IEEE standard for floating point numbers", *Resonance*, vol. 21, no. 1, pp. 11-30, 2016.

[20] T. BHAVANI and D. KPadmaja, "High Speed Signed Multiplier for Digital Signal Processing Application", International Journal of Scientific Engineering and Technology Research, vol. 02, no. 07, pp. 546-551, 2013.