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

Muhammad Sajjad¹, Mohd Zuki Yusoff² and Muhammad Ahmed²

¹CISIR, Universiti Teknologi PETRONAS, Perak, Malaysia
mmuhammad_19001746@utp.edu.my; mzuki_yusoff@utp.edu.my
²National 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.
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.

\[
S_{\text{out}} = KV_a + B \quad \text{(1)}
\]
\[
V_a = K_s(S_{\text{out}} - B) \quad \text{(2)}
\]

- \(S_{\text{out}}\): 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, \(K_s S_{\text{out}}\) 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 \((S_{\text{out}} - B)\) would approximately equal to \(S_{\text{out}}\) 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).

\[
K_{sf}(n) = \{1 + a_1 T(n) + a_2 T^2(n) ... + a_n T^n(n)\} \quad (3)
\]

\[
B(n) = \{1 + b_1 T(n) + b_2 T^2(n) ... + b_n T^n(n)\} \quad (4)
\]

\[T(n) : \text{Temperature that needs to be compensated}\]

\[K_{sf}(n) : \text{Scaling factor equation}\]

\[B(n) : \text{Offset Equation}\]

\[a_1, b_1 : \text{Coefficients}\]

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 \(A_{DBL}\) and \(B_{DBL}\) 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
The 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.

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

NOTE:
SgnX, ExpX and ManX are sign, exponent and mantissa of double precision number X.

53-bit multiplication is done on unsigned numbers.
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.

![Figure 3. Flow Chart for Fixed-to-Double Conversion](image1)

![Figure 4. Flow Chart for Double-to-Fixed Conversion](image2)

![Figure 5. Datapath of the Processor (A)](image3)
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**

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

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

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.

\[ B_{cmp} = K_{sf}B_{raw} + a_1T_1 + a_2T_2 + a_3G_1 + a_4G_2 \]  

(5)

**B_{cmp}**: 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

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ksf</td>
<td>1</td>
</tr>
<tr>
<td>a₁</td>
<td>0.10</td>
</tr>
<tr>
<td>a₂</td>
<td>2.00</td>
</tr>
<tr>
<td>a₃</td>
<td>0.23</td>
</tr>
<tr>
<td>a₄</td>
<td>2.40</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Logic Components</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Registers</td>
<td>10,106</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>14,389</td>
</tr>
<tr>
<td>Number of occupied Slices</td>
<td>5,287</td>
</tr>
<tr>
<td>Number of MUXCYs used</td>
<td>3,796</td>
</tr>
<tr>
<td>Number of LUT Flip Flop pairs used</td>
<td>15,684</td>
</tr>
</tbody>
</table>

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.

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


© 2020 by the author(s). Published by Annals of Emerging Technologies in Computing (AETiC), under the terms and conditions of the Creative Commons Attribution (CC BY) license which can be accessed at http://creativecommons.org/licenses/by/4.0.