Distance and Speed Measurements using FPGA and ASIC on a high data rate system

Abdul Rehman Buzdar, Liguo Sun, Azhar Latif, Abdullah Buzdar
Department of Electronic Engineering and Information Science
University of Science and Technology of China (USTC)
Hefei, Peoples Republic of China

***Abstract*—This paper deals with the implementation of FPGA and ASIC designs to calculate the distance and speed of a moving remote object using laser source and echo pulses reflected from that remote object. The project proceeded in three phases for the FPGA implementation: All-in-C design using Xilinx Microblaze soft core processor system, an accelerated design with custom co-processor and Microblaze soft core processor system, and full custom hardware design implemented using VHDL on Xilinx FPGA. Later the complete system was implemented on ASIC. The ASIC implementation optimized the modules for area and timing for a 130nm process technology.**

Keywords*—*Distance; Speed; FPGA; MicroBaze; Co-Design; ASIC

I. Introduction

Distance and speed measurement systems are widely used in many areas including automobiles, defense etc [1-15]. The system can measure the time interval between two laser pulses, one reference pulse which is sent out by the system and one echo pulse which is reflected back to the system. With that time information the system should calculate the distance to the object on which the echo pulse is reflected. The system level view of the project is shown in Fig. 1 and the reference and echo signals are depicted in Fig. 2. There are to be two phases in the project, in the first phase the target hardware is a Xilinx FPGA [16], in the second phase the target hardware is an application specific integrated circuit (ASIC) solution where VHDL and standard cells is to be used. The target process is a 130nm CMOS process from the foundry ST Microelectronics [20]. Cadence [21] EDA tools are used for the ASIC implementation. The FPGA system implementation phase can be further divided into three sub phases. The first of these phases aims to deliver a software only product, where the entire system is written in C programming for Xilinx MicroBlaze soft processor system [17]. The second is a mixed hardware-software implementation where a part of the system is implemented in C and part of it in hardware. The hardware is to be designed using VHDL hardware description language. The third and final product is to be implemented completely in hardware and implemented on Xilinx XUP Virtex-II Pro Development System [18]. Lacking a real laser detector, a laser emulator is to send different test vectors to the distance measurement system. It should feed the digital filter block through two channels, each channel operating at 100 MHz. A test vector generator emulating the laser detector i.e emulate the laser signal pulses "Reference" and "Echo" is developed in VHDL and is named "AD model". Each measurement should comprise of 256 samples. Finally as the system should be able to measure speed and the emulator should also be able emulate movement. The second block is the digital filter, it is a five tap correlation filter. The signals must be filtered using a digital filter shown in Equation (1) to suppress noise. where A-E are the filter coefficients. Initial filter coefficients have been provided but the user should be able to change coefficients during use.

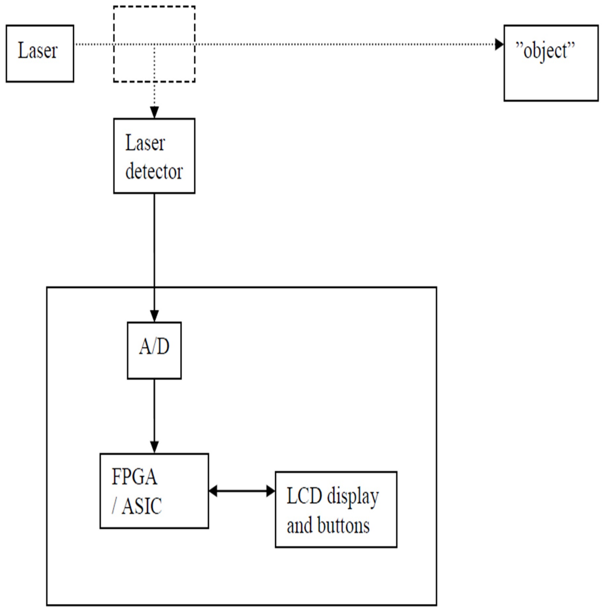

Figure 1: System Level View of Project

After filtering, the maximum point of the reference signal is found by searching for the maximum value of $d_1[i]$ in the range 0 - 20 samples (i = 0 .. 19). 20 samples are supposed to be the longest possible delay between "laser-trig" goes high and the laser pulse is transmitted. In the same way a possible echo signal is searched in the interval 21 - 255 samples (i = 21 .. 255).

$$\begin{aligned} d_1[i] = A * d_0[i-2] + B * d_0[i-1] + C * d_0[i] \\ + D * d_0[i+1] + E * d_0[i+2] \end{aligned} \quad (1)$$

To increase the resolution of i an interpolation using the two values surrounding the maximum value $d_1[i]$ is performed according to Equations (2), (3), and (4). This gives the final time sample point j.

$$b = d_1[i-1] - d_1[i+1] \tag{2}$$

$$c = 2 * (d_1[i-1] - 2 * d_1[i] + d_1[i+1]) \tag{3}$$

$$j = i + b/c \tag{4}$$

The third block is the distance and speed measurement. Main requirement for this block is that the distance should have a precision of one decimal. The distance to the object can be found by calculating the time difference between the reference and the echo signals. Assuming the sampling frequency is 200 MHz and using the speed of light, we get Equation (5):

$$Distance = (j_{echo} - j_{ref}) * c/(2 * 200 * 10^6) \tag{5}$$

Fourth block is a user interface utilizing an LCD for printouts and a keypad for input. Maximum range for the system should be 250 meters. Measurements should be possible at a 100 KHz repetition frequency, that is each measurement should be performed in less than 10us.

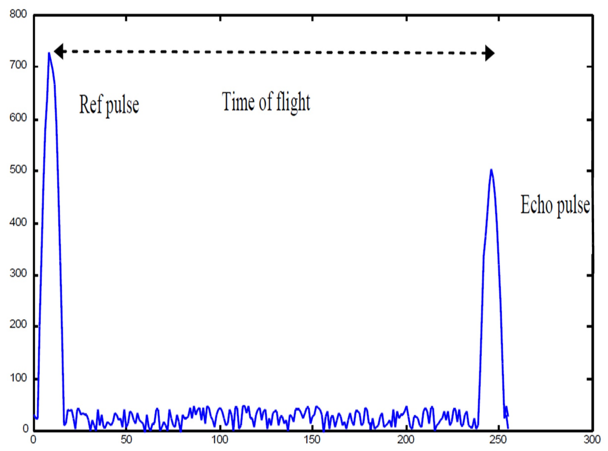

Figure 2: Reference and Echo Signals

II. Software Implementation

In this phase everything was implemented in C programming language. The C code was first tested on personnel computer (PC) to see how the filter and other blocks are working. This implementation made the idea behind the project clear. After successfully testing it on PC, the same approach was used for Xilinx MicroBlaze soft processor system [17]. As floating point operations are expensive therefore it was decided not to use floating point unit. Due to the lack of a floating point unit a fixed point system was implemented. The actual speed of light is 299792458 but it was rounded off to $3 * 10^{-8}$. This rounding off resulted in a little deviation in the decimal part and that was expected. We first printed the results using Hyper-terminal by connecting serial port of Microblaze with the serial port of PC. Later we integrated LCD to display the results. Fig. 3 shows the block level diagram of software implantation.

In this phase LCD was derived and values were received from the keypad using Software routines. Timers were also used for profiling. Fast Simplex Link (FSL) [19] was used to get values as test vectors. FSL is 32 bits wide, but only ten bits were used as the test vectors are 10 bits wide. In this phase everything was sequential. The Fig. 4 shows the software implementation flow. The sample test vectors were buffered first in an array which was then fed to the filter. After the filtration the max value was found and index was located, later interpolation was carried out and finally distance was calculated. The profiling was done to figure out the time consuming parts of the code. The table I shows the results from profiling.

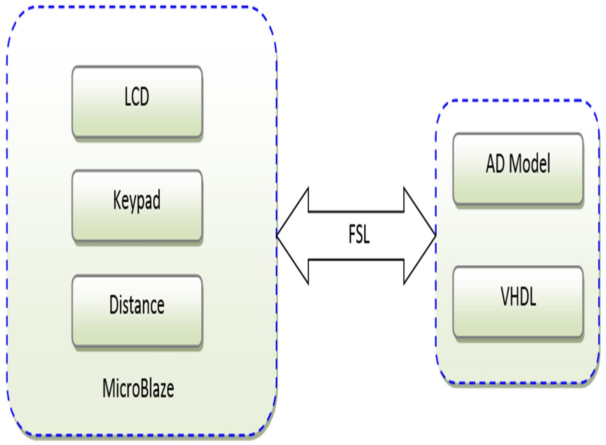

Figure 3: Block diagram of software implementation

Table I: Result of C routine Profiling

C routine	Time[us]
Read sample vector	25
Filter	789
Find Ref	5
Find Echo	14

A. Keypad Implementation

The keypad has four rows and four columns. The columns were sourced and output was sensed on rows. The keys acts like simple switches. The signal put on the columns was active low and then the rows were sensed. If there is a low signal present on the any of the row then the routine figures out which key is pressed. A routine was included for resolving debounncing issue. As the switches are kind of mechanical switches so whenever any of the key is pressed there is kind of fluctuation at the output pin and the routine can register the key multiple times.

To fix this de-bouncing problem this routine was included. The routine makes sure that it should register the key once until the key is depressed. Four keys were used for changing test vector they are 1 2 3 A as shown in Fig. 5. The Fig. 6 shows the flow for Keypad routine.

B. Distance Calculation

For the distance calculation FSL link was used to get the vector values from AD model, then they were fed to the distance calculation routine. After getting these vectors, filtering and interpolation was carried out. There was little variation in decimal place as the speed of light was rounded off. Table

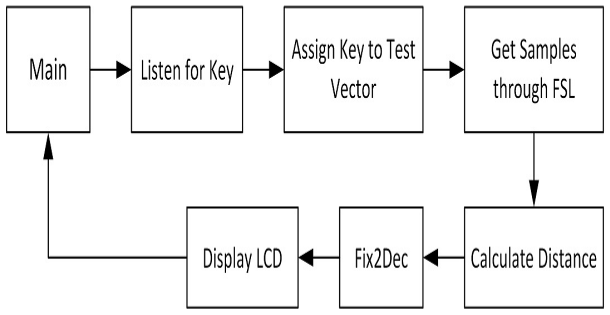

Figure 4: Software Implementation Flow Chart

Figure 5: Keypad Keys

I shows the result of profiling on the Distance calculation routine. The Fig. 7 shows flow for Distance calculation routine.

1) Fix2Dec: This routine takes the digit from the distance calculation routine and separates it into integer and decimal part. These values are then sent to the LCD routine to be displayed on LCD.

C. LCD Implementation

The LCD used is a 16x1 display format. It contains three control signals and 8 bit wide data bus to receive commands and data. The control signals are Enable, Read/Write and Register Select. Data and commands should be latched on the Enable signal. For displaying to the LCD, write signal should be low and for command and data Write Register should be Low and High respectively. Fig. 8 shows the flow for LCD routine.

III. Mixed HW-SW Implementation

In mixed HW-SW Implementation Phase some parts of the project were implemented in software and some in Hardware. The Hardware-Software co-design is a well established technique, which improves the performance of the system [22-36] . The main interface i.e. Menu System, LCD and keypad was implemented using Software routines and the distance chain was implemented in hardware as shown in the Fig. 11. Here is a brief description of each routine.

A. Main Interface Implementation

The main interface involves a menu system which takes user input through a keypad and displays the results on the LCD. The menu system is implemented in software. The menu has the options to change coefficients, test vectors, speed and display the calculated value of distance on the LCD. Fig. 9 shows the Main Interface flow chart. The same code was used to implement the Keypad and LCD as described in software implementation phase.

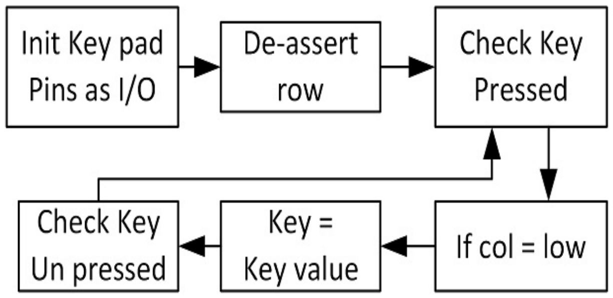

Figure 6: Keypad Routine Flow Chart

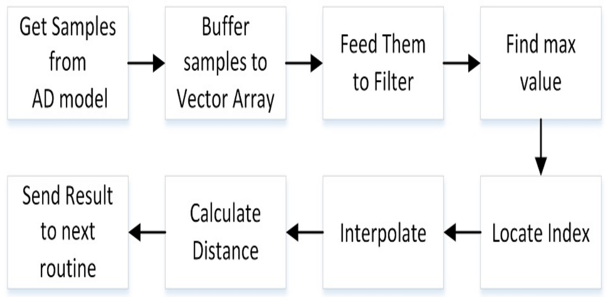
Get Samples from AD model
Buffer samples to Vector Array
Feed Them to Filter
Find max value
Locate Index
Interpolate
Calculate Distance
Send Result to next routine

Figure 7: Distance Chain Routine Flow Chart

B. Menu Routine Implementation

In mixed HW-SW Implementation Phase the main interface is implemented using only four buttons i.e. plus, minus, enter and back. The reason of using less number of keys is to minimize the number of pins out from our design.

When the device is turned ON the user can choose between different options from the main menu and they are displayed on the LCD i.e. change test vectors, coefficients and speed. To move backward and forward in the menu the plus/minus keys are provided to the user and to select any of these options enter key is used.

C. Distance Chain Implementation

In mixed HW-SW Implementation Phase we decided to implement the whole distance chain in VHDL and only keep non-timing sensitive stuff inside the MicroBlaze. The reason for doing this is that we know that we would have to do it in complete hardware Implementation Phase and this way could save time although it might be harder. The Fig. 10 shows the overview of the datapath of distance chain for mixed hw-sw implementation. The Distance chain is compromised of these blocks:

D. AD Model

Very small changes was done to this block from software implementation phase, we implemented a start signal and then a start filter signal to show when the filter would have its first two values. Together with the start signal the test vector chosen is sent.

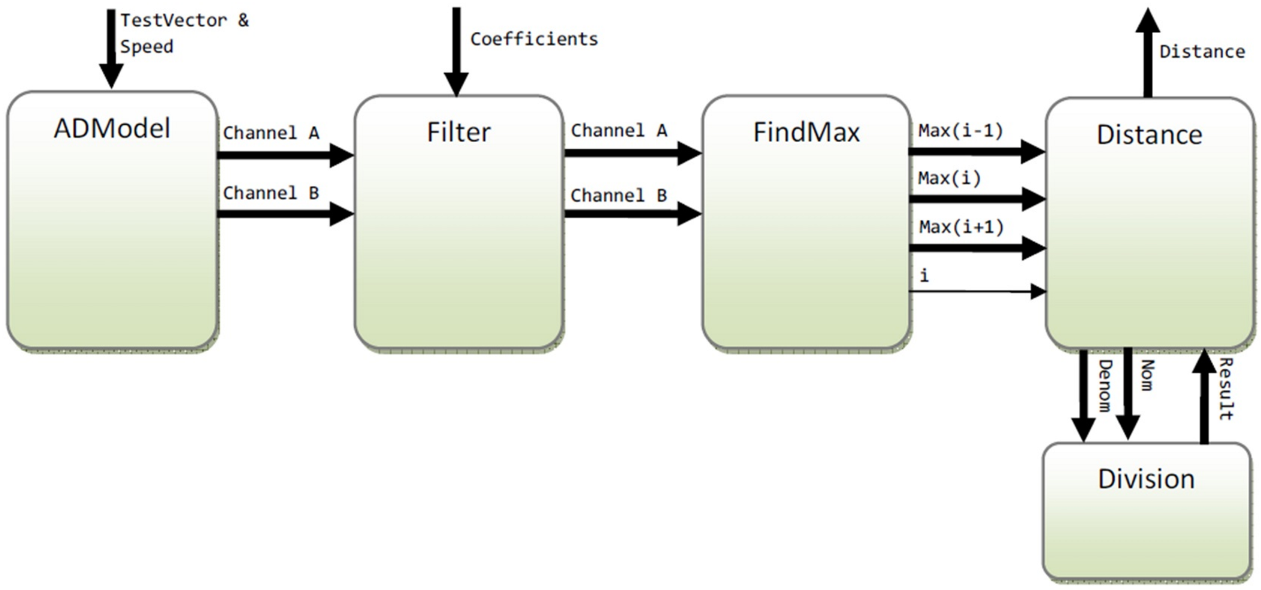

Figure 10: Overview of the datapath of distance chain for mixed hw-sw implementation.

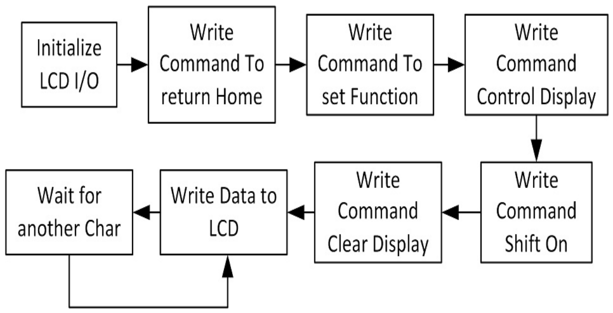
Initialize LCD I/O
Write Command To return Home
Write Command To set Function
Write Command Control Display
Write Command Shift On
Write Command Clear Display
Write Data to LCD
Wait for another Char

Figure 8: LCD Routine Flow Chart

E. Filter Block Implementation

The filter was a bit more complicated, it was decided to try to implement a filter that could work independently of the speed of the Admodel with a FIFO buffer. That idea later got scrapped for the final version that is a block with a 5 word wide circular buffer that updates with both channels from the Admodel and completing two filtrations each clock cycle. To be able to accomplish that with all the multiplications and additions a three stage pipeline was implemented to lower the load of the FPGA. Changing the coefficient was also implemented as parallel ports sent directly from the controller block.

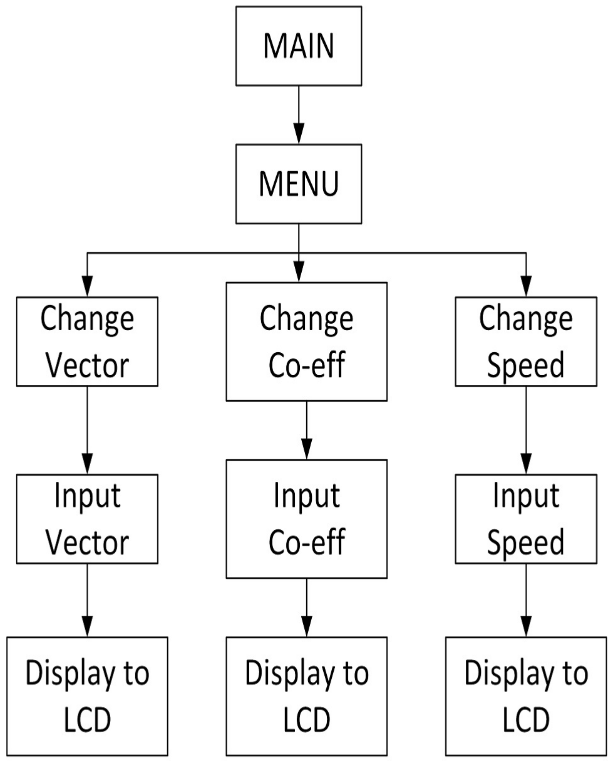

Figure 9: Main Interface Flow Chart

F. FindMax Block Implementation

The next stage in the chain was to find the position of the reference and echo pulse. As these operations are identical the block was implemented to run for 20 clock cycles send out the maximum $value(i)$, $value(i+1)$, $value(i-1)$ and the index to the distance block. When doing this it would start the distance and interpolation unit so it could work in parallel to findmax and do some precalculations as it was waiting for the value and position of the echo pulse. Findmax would then restart and try to find the echo pulse and then signal the distance unit again. Findmax is implemented as a simple comparator that first compare the two filtered values against each other and then to the current maximum value. If the value is larger than the last maximum the previous and next values are saved in a buffer until they are sent out. To be able to save the next value

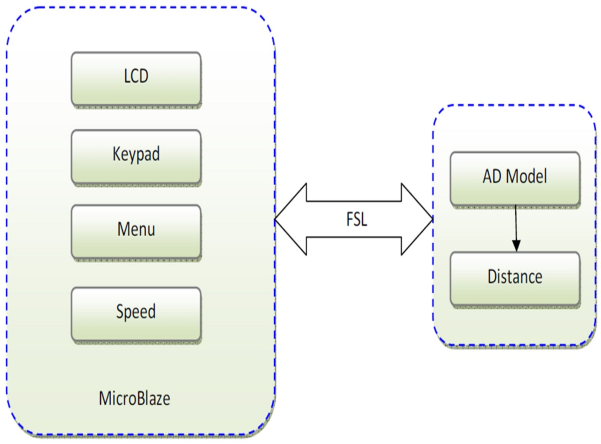

Figure 11: Block diagram of Mixed implementation

a flag was created to be able to save it the next clock cycle when it was fed from the Filter.

G. Distance and Interpolation

For both the reference and the echo, division has to be done to calculate the interpolation. The division is handled by a separate block that implements fast non-restoring division. The division can't handle negative numbers so the sign of the value b has to be checked and made to its two-complement and then the result from the division has to be made to its two complement to make the correct calculation when calculating j. This is done two times until both j_{ref} and j_{echo} are calculated.

The first is precalculated as mentioned above. Both of these distances have fractional numbers that is represented in fixed point with 5 bit precision. 5 bits were large enough that we would get the necessary precision for the 10 cm resolution that we need to display to the user, but at the same time not unnecessary large and take up space in the FPGA. As the speed of light and the sampling frequency in the last calculation is constant we precalculated this and implemented it as a constant inside the block. The result is then sent out through the FSL-link.

IV. Hardware Implementation

In complete hardware implementation phase a lot of things were already done, but it needed to be tied together with a controller and a menu system that could interface with all the other parts in VHDL. Modeling of speed were implemented, the keypad and LCD routines were ported to VHDL. The Fig. 12 shows the datapath and the different components of complete hardware implementation.

A. Distance Chain Implementation

The only changes here from what were used in mixed implementation, was the implementation of speed inside the Admodel. It was decided that the easiest way to do this would be to try to shift the echo pulse in each test vector, and when it is close to the end or reference, shift it the other way implying that the object were moving towards and away from the user. The speed of the simulation should be able to be set so we included a signal that was sent from the main program.

B. Main Block Implementation

The main program holds all the outside ports, all other components and the menu system. It has to listen for the key presses, start the distance chain, calculate speed and update variables when inside the menu. The main basically has two major components to keep track of, the distance calculation and the menu-system. When in distance calculation mode it starts the distance chain waits for it to finish and then depending on how long since it updated the LCD it might calculate speed and then update the LCD using DisplayText routine. If its in distance calculation mode, it can go into the menu by pressing ENTER.

C. DisplayText Routine Implementation

The main has to update variables and show these updated variables to the user together with a string that shows what the variable is i.e. (Testvector=variable), it was decided that sending whole strings to the LCD unit would take too much space so a system where a variable together with the screen (function) that the main want to write is sent. DisplayText routine takes the variable sent and converts it to a string using the component IntToString. When this is done a case statement chooses what to print out depending on the function. In each of these statements a custom string is built using some characters and the converted variable. This string is then fed one character at a time to the LCD and then printed. In the case of printing "D=distance S=speed" one extra variable is converted, for the integer value of the speed that comes from the speed block.

D. IntToString Routine Implementation

This block takes an integer between 0 and 999 and converts it to a three character string. If the number isn't 3 characters long it will be padded with spaces. The algorithm does checks against the size of the integer and then sends it to one of three different states: Ones, Tens and Hundreds.

E. Main Menu Implementation

The menu has to know what variable to update when PLUS or MINUS is pressed, and where to go whenever ENTER or BACK is pressed it also has to display this variable on the LCD. All variables is kept in an array and the whenever you move in the menu the index is updated and therefore the active variable that you can change is updated. Whenever a keypress happens the appropriate action is taken and then the LCD is updated by sending the variable and a function to DisplayText.

F. LCD Implementation

Hardware implementation of the LCD is almost a complete port from the C-code used previous two phases. When the device is set ON, the code initializes the LCD and then waits for DisplayText to send its first character. The limitations of the LCD regulate how long we have to set and hold the signals for it to register the change. This value has been set to 5ms using iterative methods.

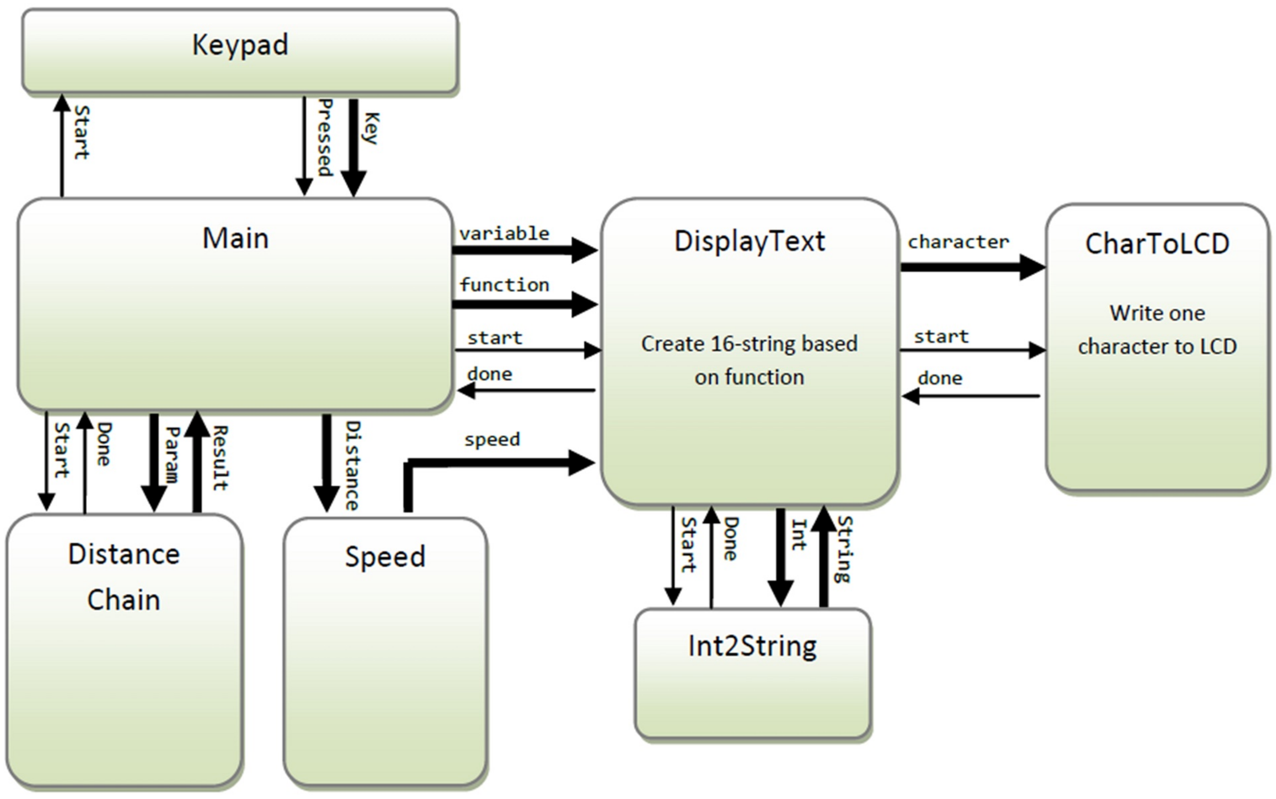

Figure 12: Overview of the datapath of distance chain for complete hardware implementation.

V. ASIC Implementation

The specification for the ASIC implementation was changed, following changes were made in the specification:

- Change in number of channels to one
- Change in data rate (200Mhz)
- Addition of SPI port
- Signed Coefficients
- Taps input Serially during initialization
- Result is in decimeter
- The data is input from pattern generator instead of AD model
- The output will be fed to logic analyzer

Fig. 13 is the Timing Diagram showing signals for all the major blocks of the system. An alternate design with a FIFO at the front end was also developed, distance chain itself was unchanged. The idea with this FIFO implementation was to allow the distance chain to run at a lower frequency and thus have lower power dissipation. Fig. 14 shows the block diagram of final ASIC design

A. FIR Filter Implementation

The filter is a 5 word buffer, on which the filter calculations are performed. To be able to accomplish all multiplications and additions required without increasing the performance load too much, a three stage pipeline has been introduced. The big changes in the filter block itself were the switch from two channels of data to one, and the use of signed coefficients. Where as the old filter block effectively had two filters, the new block only has one. The main addition to the filter block is the UART process which is used for reading coefficient data and feeding this to the filter.

B. Find Max Block Implementation

The next stage in the chain was to find the position of the reference and echo pulse. As these operations are identical the block was implemented to run for 20 clock cycles send out the maximum $value(i)$, $value(i{+}1)$, $value(i{-}1)$ and the index, to the distance block. When doing this it would start the distance and interpolation unit so it could work in parallel to findmax and do some pre-calculations as it was waiting for the value and position of the echo pulse. Findmax would then restart and try to find the echo pulse and then signal the distance unit again. Findmax is implemented as a simple comparator with a one word buffer. It checks the present value against the maximum value, if the present is higher it will store the present value and the value in the buffer, which is the previous value. It will also set a flag to store the next value. When the first 20 values have been checked the stored values will be sent to the distance unit, and the process will repeat for the echo values.

C. Distance and Interpolation

For both the reference and the echo, division has to be done to calculate the interpolation. The division is handled by

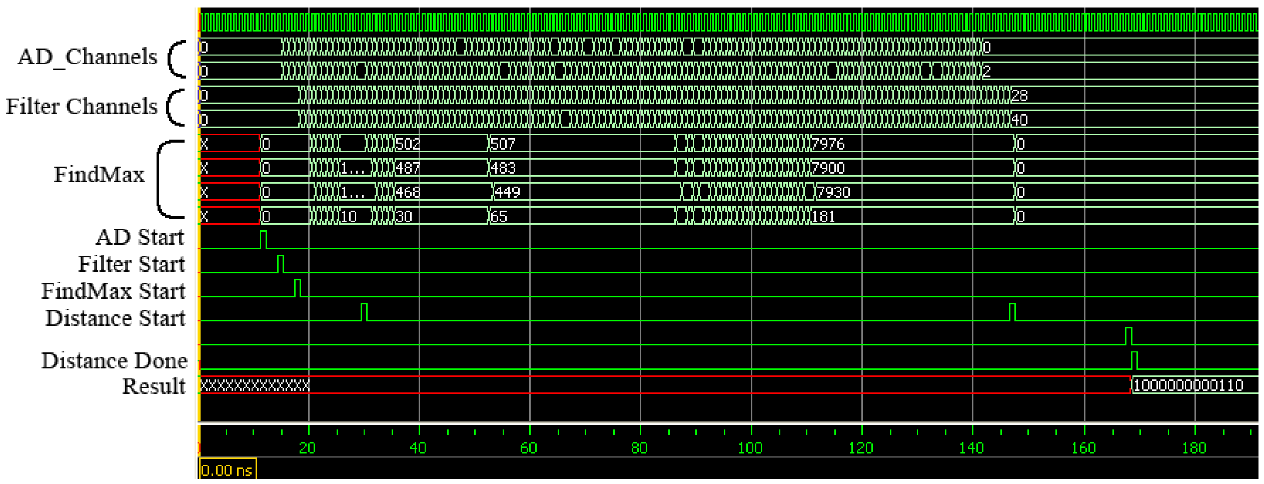

Figure 13: Timing Diagram showing signals for all the major blocks of the system

a separate block that implements fast non-restoring division. The division can't handle negative numbers so the sign of the value b has to be checked and made to its two-complement and then the result from the division has to be made to its two complement to make the correct calculation when calculating j. This is done two times until both j_{ref} and j_{echo} is calculated. The first is pre-calculated as mentioned above. As the speed of light and the sampling frequency in the last calculation is predefined we pre-calculated this and implemented it as a constant inside the block. The result is then sent out through the SPI.

D. Speed Calculation

The speed is calculated by storing a distance value, waiting a given time, then calculating the difference in distance compared to the stored value. This value is then multiplied with the waiting time divided by 1 second. For example the waiting time is by default set to 0.2s, to calculate the speed in dm/s the difference in distance is therefore multiplied by 5.

E. Serial Peripheral Interface (SPI) Implementation

An SPI port was implemented for outputting the result of measurements. The design uses this SPI master/slave interface, where the ASIC acts as master, for outputting of results. The SPI is designed according to the standard one-slave configuration, using mode 0. The SPI clock (SCLK) generated from the ASIC runs at 3.125 MHz. The SPI (in the ASIC) initiates a data transfer each 10 us, that is, at a frequency of 100 kHz. The data transfer is initiated by lowering the SS signal, according to the SPI standard. The output data is specified to be 12 bits unsigned values. The total number of bits for outputting over SPI is 16, for compliance with standard SPI components. The first bit is an indication of whether the data sent is a distance measurement or a speed measurement. A zero in this bit indicates distance, a one indicates speed.

F. Evaluation

There were two architectures which were considered. One was using FIFO so that the whole processing time can be spread over the whole spectrum of time. The FIFO was running on high speed clock i.e. 200 MHz where as rest of the design was working on slow clock i.e. 3.125 MHz. The other was running on the high speed clock. Synthesis of both the design was carried out at 4ns constraint to get a rough picture of area, timing, and power, table II shows the results.

Table II: FIFO and No FIFO

	FIFO	Without FIFO
Area [mm^2]	0.15	0.04
Timing [ns]	1.48	3.70
Power [mW]	18.54	6.68

Architecture without FIFO is taking 3 times less area then with FIFO. Both the designs fulfill the timing constraint of 4 ns but the design with FIFO is more efficient in terms of timing. The design with FIFO is 2.7 times expensive in term of power. The design without FIFO was chosen as it was less expensive in terms of power and area.

After synthesis power was simulated using VCD-files generated by the test bench, the table III show the results.

Table III: CLK Power

Net	Power(mW)	Cap(nF)
CLK	0.495	1.719

Using the expression $(P = f * Vdd^2 * C)$ clock power at 1.2V was 495072 nW, which verifies the result we got from the RTL compiler. The table IV shows power consumption in major blocks.

Table IV: Power consumption in major blocks

Block Name	Power (mW)
Filter	1.61
Distance	1.03
Findmax	0.88
SPI	0.09
Clkdiv	0.08

The results show that filter is the most power hungry

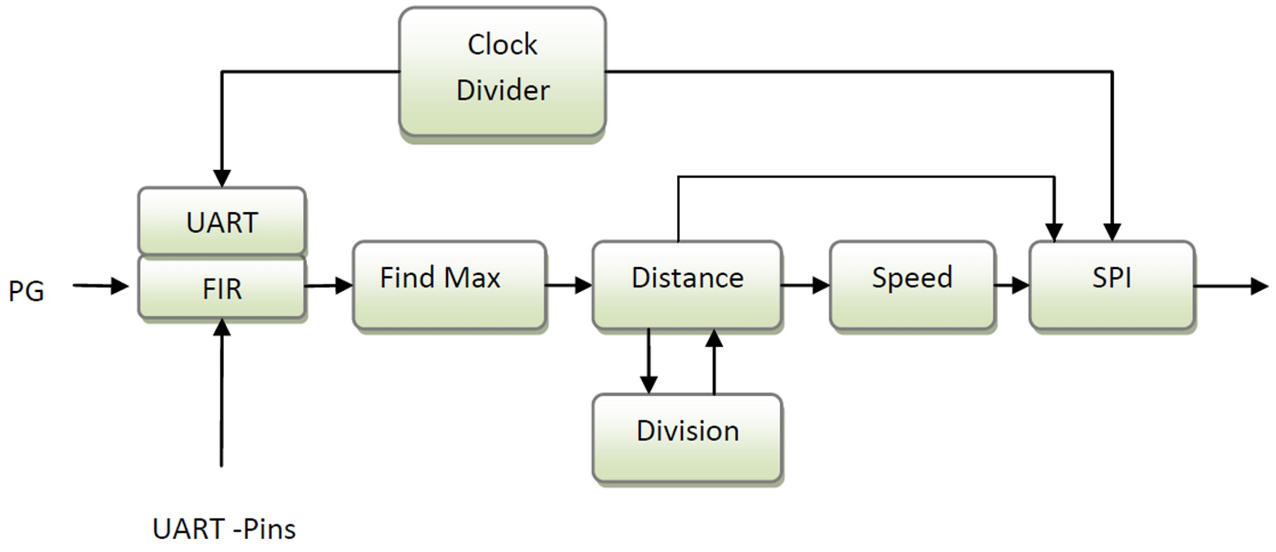

Figure 14: Overview of ASIC Design

block. The filter contains number of multipliers therefore it is consuming large part of power. The table V shows total power consumption.

Table V: Total Power

Leakage Power(mW)	Dynamic Power(mW)	Total Power(mW)
1.217	3.254	4.472

The table VI shows total area and timing results.

Table VI: Total Area and Timing

Area (um^2)	Timing (ns)
0.046	4.251

The timing constraint given was 5ns. The initial synthesis was carried out at medium effort and we got a slack of 749ps. The critical path was found to be between filter and findmax. The filter and findmax were consuming the major core area. Fig. 15 shows the layout of the chip.

Post layout the design was pad limited, using 24 pads, of which 3 were unused. The core area was $0.11mm^2$, with a core utilization of 43%. This gave a die area of $0.55mm^2$.

G. Verification

The complete design was verified with the help of test bench. This test bench would emulate AD converter functionality and send input to the design, and then it would receive distance and speed from the design and verify those results. The test bench is a comprehensive test bench designed to verify that the chip conformed to the specification with regard to functionality. The test bench is comprised of 14 test cases. Each test case is basically a component in the test bench, and they all share the component which holds the design. While a case is running it prints results to a corresponding text file, when it has finished it relinquishes control of the

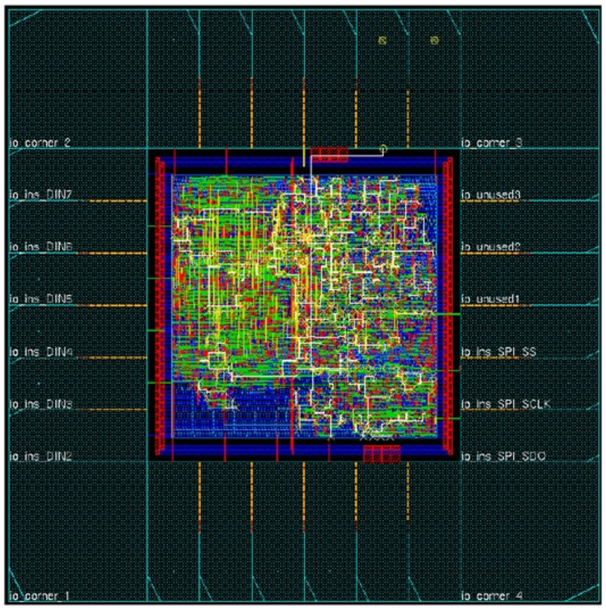

Figure 15: Layout of the chip

design to the next case. Test cases 1 through 8 tested distance measurements by moving the echo through the entire range of the specification, with different parameters. Test case 5 for example moves backwards through the range, and case 7 has negative coefficients. The remaining test cases were mainly for verification of speed measurements.

H. Physical Testing of Chip

The Fig. 16 shows pinout of the chip. For physical testing, power up the chip using 1.2V as VDD and Gnd as VSS. The interface has UART where you can feed co-efficient after reset, the coefficients are 8 bit wide and three coefficients needs to be entered. Put the DAV line high and feed 255 bytes to the chip. Put the DAV line low and you will get the result at SPI. The data out from the chip is 16 bit wide word and last three bits are redundant. The result of the speed calculation is after every 20,000 distance calculations. If the LSB is set the resulting word is speed and if it is clear the resulting word is distance.

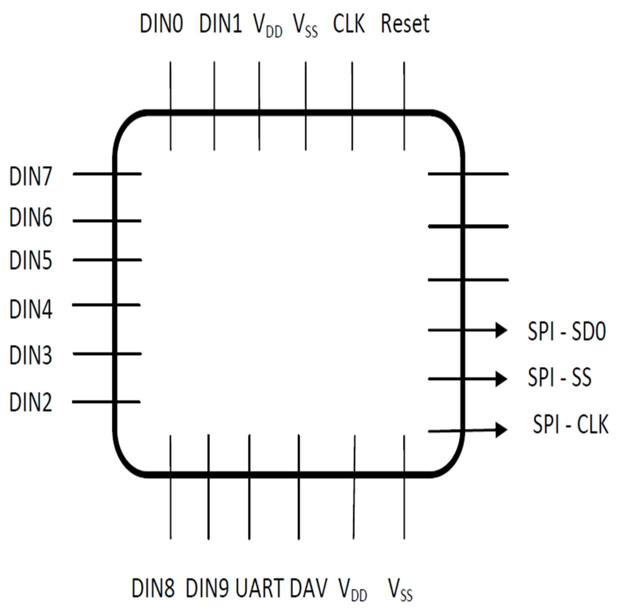

Figure 16: Pinout of the chip

VI. Conclusion

This paper presented a very practical problem and allowed the project group to approach it from several different viewpoints. This proved very useful as it showed the weaknesses and strengths of the platforms used in the project, or rather it showed how much impact these attributes have on real performance. For example it is quite expected that a hardware implementation should be faster than a software implementation. However, it was surprising to a roughly 400x increase in performance, which is the improvement from software phase to complete hardware implementation.

Acknowledgment

This work is partially supported by the Chinese Academic of Sciences (CAS) and The World Academy of Sciences (TWAS).

References

[1] T. Schlegl , T. Bretterklieber , M. Neumayer and H. Zangl “A novel sensor fusion concept for distance measurement in automotive applications”, IEEE Sensors, pp.775 -778 2010

[2] W. J. Fleming “New automotive sensors A review”, IEEE Sensors J., vol. 8, no. 11, pp.1900 -1921 2008

[3] T. Gandhi and M. M. Trivedi “Pedestrian protection systems: Issues, survey, and challenges”, IEEE Trans. Intell. Transp. Syst., vol. 8, no. 3, pp.413 -430 2007

[4] D. Marioli , C. Narduzzi , C. Offelli , D. Petri , E. Sardini and A. Taroni “Digital time-of-flight measurement for ultrasonic sensors”, IEEE Trans. Instrum. Meas., vol. 41, no. 1, pp.93 -97 1992

[5] C. Cai and P. P. L. Regtien “Accurate digital time-of-flight measurement using self-interference”, IEEE Trans. Instrum. Meas., vol. 42, no. 6, pp.990 -994 1993

[6] F. E. Gueuning , M. Varlan , C. E. Eugene and P. Dupuis “Accurate distance measurement by an autonomous ultrasonic system combining time-of-flight and phase-shift methods”, IEEE Trans. Instrum. Meas., vol. 46, no. 6, pp.1236 -1240 1997

[7] C. C. Tong , J. F. Figueroa and E. Barbieri “A method for short or long range time-of-flight measurements using phase-detection with an analog circuit”, IEEE Trans. Instrum. Meas., vol. 50, no. 5, pp.1324 -1328 2001

[8] S. S. Huang , C. F. Huang , K. N. Huang and M. S. Young “A high accuracy ultrasonic distance measurement system using binary frequency shift-keyed signal and phase detection”, Rev. Sci. Instrum., vol. 73, no. 10, pp.3671 -3677 2002

[9] L. Angrisani , A. Baccigalupi and R. S. L. Moriello “A measurement method based on Kalman filtering for ultrasonic time-of-flight estimation”, IEEE Trans. Instrum. Meas., vol. 55, no. 2, pp.442 -448 2006

[10] C. F. Huang , M. S. Young and Y. C. Li “Multiple-frequency continuous wave ultrasonic system for accurate distance measurement”, Rev. Sci. Instrum., vol. 70, no. 2, pp.1452 -1458 1999

[11] Y. S. Didosyan, H. Hauser, H. Wolfmayr, J. Nicolics and P. Fulmek “Magneto-optical rotational speed sensor”, Sens. Actuators A, Phys., vol. 106, no. 3, pp.168 -171 2003

[12] L. Wang, Y. Yan, Y. Hu and X. Qian “Rotational speed measurement through electrostatic sensing and correlation signal processing”, IEEE Trans. Instrum. Meas., vol. 63, no. 5, pp.1190 -1199 2014

[13] Y. Yan, B. Byrne, S. Woodhead and J. Coulthard “Velocity measurement of pneumatically conveyed solids using electrodynamic sensors”, Meas. Sci. Technol., vol. 6, no. 5, pp.515 -537 1995

[14] J. Ma and Y. Yan “Design and evaluation of electrostatic sensors for the measurement of velocity of pneumatically conveyed solids”, Flow Meas. Instrum., vol. 11, no. 3, pp.195-204 2000

[15] Lijuan Wang, Yong Yan , Yonghui Hu , Xiangchen Qian “Rotational Speed Measurement Using Single and Dual Electrostatic Sensors”, IEEE Sensors, pp. 1784-1793 2014

[16] Xilinx Inc. FPGA Design Tools. Silicon Devices. www.xilinx.com

[17] Xilinx. MicroBlaze. www.xilinx.com/tools/microblaze.htm

[18] Xilinx Virtex-II Board. www.xilinx.com/univ/xupv2p.html

[19] Xilinx FSL. www.xilinx.com/products/intellectual-property/fsl.html

[20] STMicroelectronics. www.st.com/web/en/home.html

[21] Cadence EDA Tools. www.cadence.com/en/default.aspx

[22] M. Imai, “Embedded tutorial: hardware/software codesign”, IEEE Asia and South Pacific Design Automation Conference (ASP-DAC), Jan. 1999.

[23] J. Noguera, R.M. Badia, “HW/SW codesign techniques for dynamically reconfigurable architectures” IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol. 10, no. 4, pp. 399-415, Aug. 2002.

[24] M. D. Edwards, et al., “Acceleration of software algorithms using hardware/software co-design techniques”, J. Syst. Architecture, vol. 42, no. 9/10, pp.1997.

[25] W. Wolf, “A Decade of Hardware/Software Codesign,” IEEE Computer, vol. 36, pp. 38-43, April 2003

[26] R. Ernst, J. Henkel, and T. Benner, “Hardware-Software Cosynthesis for Microcontrollers,” IEEE Transaction on Design and Test of Computers, vol. 10, pp. 64-75, December 1993

[27] W. Wolf, “Hardware/software Co-design of Embedded Systems,” IEEE Proceeding, vol. 82, pp. 967-989, July 1994

[28] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood, “Hardware-Software Co-Design of Embedded Reconfigurable Architectures,” in Proceeding of 37th Design Automation Conference, pp. 507-512, June 2000

[29] Vermeulen, L. Nachtergaele, F. Catthoor, D. Verkest, and H. De Man, “Flexible Hardware Acceleration for Multimedia Oriented Microprocessors,” IEEE Transactions on Very Large Scale Integration Systems, pp. 171-177, December 2000

[30] M. Boden, J. Schneider, K. Feske, and S. Rulke, “Enhanced Reusability for SoC-based HW/SW Co-design,” in Proceeding of Euromicro Symposium on Digital System Design 2002, pp. 94-99, September 2002.

[31] Kai-Yuan Jan, Chih-Bin Fan, An-Chao Kuo, Wen-Chi Yen, and Youn-Long Lin, "A Platform-based SOC Design Methodology and Its Application in Image Compression," Special Issue on HW-SW Codesign for SoC, International Journal of Embedded Systems, Inderscience Publishers, USA. Vol. 1, Issue 1/2, pp. 23-32, 2005.

[32] Chiodo, M. and et al. Hardware-software codesign of embedded systems. In IEEE Micro, 1994.

[33] Ernst, R. and et al. Codesgin of embedded systems: status and trends. In Proceedings of IEEE Design and Test of Computers, 1998.

[34] Gallery, R. and et al. Hardware/software partitioning and simulation with SystemC. In Proceedings of the 2nd WSEAS ICECSP, 2003.

[35] Hurk, J. and et al. System Level Hardware/Software Co-Design: An Industrial Approach, 1997.

[36] De Micheli, G. and et al. Hardware/Software Co-design. In Proceedings of the IEEE, 1997.