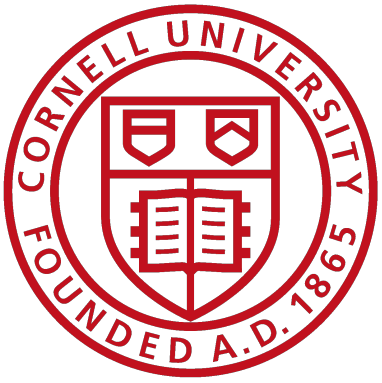
*eye*ROBOT - HETEROGENEOUS ROBOTIC PLATFORM FOR REAL-TIME COMPUTER VISION



A Design Project Report
Presented to the School of Electrical and Computer Engineering of Cornell University
In Partial Fulfillment of the Requirements for the Degree of Master of Engineering, Electrical and Computer Engineering

Submitted by
Mohit Yogesh Modi (mm2675)
Sravya Chinthalapati (sc2655)

MEng Field Advisor: Prof. Zhiru Zhang

Degree date: May, 2015

Abstract

Master of Engineering Program
School of Electrical and Computer Engineering
Cornell University
Design Project Report

Project Title: *eye*ROBOT - Heterogeneous Robotic Platform for Real-time Computer Vision

Authors: Mohit Yogesh Modi (mm2675), Sravya Chinthalapati (sc2655)

Abstract:
Developing FPGA prototypes using hardware description languages (HDL) is time consuming and limits the outreach of the designers. Instead, if the high level programming languages like C and C++ is used to develop hardware, then it can be developed much faster and can be adopted by wide range of programmers. The goal of this project is to develop a heterogeneous design which can drive robot based on the gesture recognition algorithm on real-time HD video stream via FPGA. The entire project is done using one of the most popular high level programming language C++, which opens a new opportunity for software engineers who are interested in developing hardware prototypes without learning HDL languages. The purpose of the project is to learn the challenges and possibilities that high-level digital synthesis brings during development of the hardware.

iRobot Create is a mobile robotic platform which can be driven based on the input commands provided over serial interface. In this project, iRobot is programmed to drive in designated direction based on the movement of the object perceived from an HD video connected with FPGA. This involves many aspect of the hardware-software co-design including FPGA prototyping, High-Level Digital Synthesis (HLS), Embedded System Development, Software Programming, Networking and Computer Vision.

Executive Summary

High-level Synthesis (HLS) is an automated design process that interprets the algorithmic behavior of high level programming languages and creates digital hardware that implements that behavior. The goal of eyeROBOT project is to create a heterogeneous computing platform on FPGA which allows controlling of robot based on the gesture recognition algorithm. Entire FPGA implementation is done in C++ using HLS design flow. The heterogeneous aspect of the project involves Dual core ARM Cortex-9 as a host processor which controls the main execution flow of the project and FPGA fabric as a video accelerator which performs gesture recognition on real-time HD video stream depending on host's instruction. The reason for using ARM as a host processor is because ARM has access to all peripheral ports like HDMI and ethernet. The reason for using FPGA as a video acceleration is because real-time HD video processing on host processor is not feasible.

All the work was accomplished for the eyeROBOT project, towards achieving the original expectation. Towards the end of project, Heterogeneous core was successfully able to recognize gesture from HD video and was able to send commands over network. The iRobot (a robotic platform) was also successfully able to receive the commands sent by heterogeneous core and was able to drive in desired direction based on the implied gesture. In addition, the same version of the project was developed in pure software (without any FPGA element involved) for comparison and benchmarking purposes. The eyeROBOT project has won 2nd prize in Computer Systems category in MEng Poster competition held in Cornell University in May 2015 [18].

There are challenges and difficulties involved in the project. The project deals with many aspects of the hardware-software co-design. So interaction between them is a key component. It requires rigorous design definition and design abstraction to develop all components individually and then merge them together to build the whole computing system. Rapid prototyping was also the key step for accomplishing the project. The risk items in the development stage were prototyped on the earlier stage to estimate its difficulty and feasibility.

Table of Contents

1 Introduction

1.1 Motivation

The motivation of this project is to explore the field of heterogeneous computing using High-Level Digital Synthesis (HLS). We wanted to build a system which can use FPGA as an accelerator along with the main processor and communicate with the outside world. As a specific application, we have programmed the FPGA to detect the video gesture from HDMI camera and send the commands to iRobot to drive in desired direction. As video processing on real-time HD video stream is slow on normal processor, we used FPGA as an accelerator to offload some of the work from processor. This project involves many aspects of hardware-software co-designing like HLS, Embedded Programming, Computer Vision and FPGA prototyping, making it an ideal project for students in Electrical and Computer Engineering field.

1.2 Overview

In our design, camera input is fed to the processing unit which can either be a computer, running software or an FPGA running heterogenous programs. The processing unit interprets the commands from the input video and sends it to the wireless access point. The wireless access point is used to broadcast the commands to Intel Galileo. Intel Galileo has an wireless receptor named Intel N135, which is responsible to receive the commands sent by access point. Intel Galileo receives the command from the wireless receptor and sends it to the iRobot over UART serial link operating at 57600 baud rate. The overall high level design of the project is shown in Figure 1.

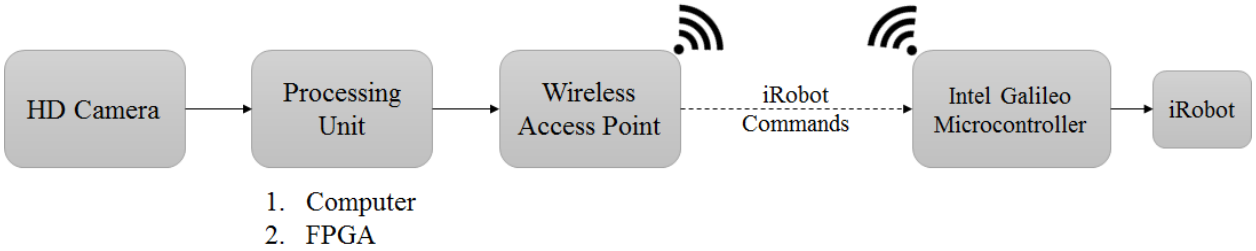


Figure 1: Abstract design of Project Objective

1.3 System of Requirements

Heterogeneous programming on FPGA using HLS is the priority of the project. Following were the system of requirements which we targeted throughout the project.

- All hardware programming should be done in high level programming language like C++ without verilog coding involved
- FPGA fabric should be used as an accelerator along with the main processor

- Significant workload has to be given to FPGA so that the computation supersedes the communication overhead between processor and FPGA.
- Vision-based gesture recognition algorithm should be implemented on Xilinx Zynq FPGA.
- FPGA should be able to process HDMI video frames in real-time.
- The heterogeneous core on FPGA should be able to communicate with iRobot over wireless channel.
- Performance of the heterogeneous system should be faster than that of software implementation.
- iRobot should be controlled by Intel Galileo microcontroller to take in the commands
- iRobot should be controlled smoothly and intuitively based on the gesture provided by user

2 Development Approach

We followed hierarchical and modular development approach. As the project involved fair amount of complexity, we divided the project into 2 major categories: (1) Software Design and (2) Hardware Design. Software Design was meant to develop software version of entire project. This includes camera capture, gesture recognition and command sending on Intel i3 processor, programming Intel Galileo to receive the command and send it to iRobot for driving. The hardware design was meant to develop heterogeneous core which replaces software version of video capture, gesture recognition and command sending units. We managed to finish software design in Fall semester and hardware development in Spring semester. As the software version is relatively easier than developing project on hardware, our approach helped us unblock major issues of the project earlier in the design cycle.

The incremental development approach for software and hardware design is mentioned in following sub sections.

2.1 Software Design

- **Acquired Reference Platform** from a previous project [1]. It gave us the starting point to begin the software design. The reference project implemented Gesture controlled iRobot control using software platform named *Processing*. The programming in *Processing* was done in Java programming language. Now, given the high-level design, we had a challenge to control the iRobot smoothly.
- **Smooth controlling of the iRobot using keyboard** was done to verify whether iRobot can be controlled smoothly or not. We developed a quick prototype on Processing software to receive keys from keyboard (instead of video gestures) and send the respective commands to iRobot. After successfully determining the feasibility of smooth robot control, we decided to migrate from Java programming language to C++, because C++ has better performance and also because we had planneed to migrate to HLS which uses C++ language to generate synthesizable netlist.
- **Wrote C code for video capture and command sending using Visual Studio, OpenCV and winsock.** We used Visual studio to rewrite the code of gesture recognition using OpenCV libraries. OpenCV libraries provide rich set of video processing APIs which can be leveraged to detect the gestures. After the gesture gets detected from laptop's web camera, OpenCV libraries calculates the inclination of the object and determines the angle of the object. Based on the

angle and distance of the detected red colored object from camera, code generates driving command for iRobot. To send the packet over the network, we used windows winsock APIs to send the packets over TCP network. For transmitting the data wirelessly, we hosted an virtual access point in our laptop with private SSID and password. At the receiving end, we programmed Intel Galileo to receive the command via N-135 and write it to iRobot's serial interface.

- **Implemented Gesture recognition using OpenCV libraries.** Gesture recognition essentially needs computer vision algorithms like object detection, thresholding, median filtering, corner detection etc. So we used OpenCV, a library of programming functions that are mainly aimed at real-time computer vision. Using OpenCV libraries in Visual Studio, we developed the gesture recognition algorithm that takes in a video input through the webcam and does image processing to output a set of corners of the object that will be used to generate commands that can be understood by the iRobot. We used a red colored object, the gestures of which will allow us to control the movement of the robot. The input red colored object is first converted from RGB to YCbCr domain and by setting the values of Y,Cb,Cr for our particular red object, we could classify the object distinctly from the background. After object detection, the program performs thresholding, a 5x5 median filtering followed by corner detection on the image for every frame of the input video. The corners are now used to track the orientation of the object. The angle of rotation and the distance of the object to the webcam are mapped to left, right and forward, backward motion respectively.

This way, we were able to develop the software design of the project.

2.2 Hardware Design

To implement the same functionalities on hardware was quite challenging. We wanted to replace the software processing unit with the hardware processing unit with minimal changes into the existing infrastructure. So, we decided to keep the iRobot, Galileo microcontroller code unchanged and work on FPGA to implement video capturing, gesture recognition and wireless communication. We followed the following steps to achieve this objective.

- **Conducted literature survey on how to communicate between FPGA and iRobot wirelessly.** We found a wireless chip which can be attached to FPGA via SD card port and work as a communicator between FPGA and Galileo/ iRobot. But the problem with that chip was that it was using SD card slot, which we were planning to use for loading linux operating system. So we couldn't use that chip for wireless communication. Hence we decided to take the alternative approach in which the command generated by Xilinx Zynq would be send via Ethernet port to a wireless access point and that access point broadcasts the command to Galileo/iRobot. This way we decided to use Ethernet for communication.
- **Used Xilinx SDSoC design suite to develop FPGA implementation**. SDSoC has capability to write synthesizable code using C++ and also allows developer to do heterogeneous programming. We took one of the sample HD video processing application provided by Xilinx and decided to build our project based of that. As HD video processing application requires high performance, we decided to use FPGA fabric (from now on, it's called PL - Programmable Logic) as an video accelerator (which does gesture recognition) and ARM

processor (from now on, it's called PS - Processing System) to do all other work. This involved acquiring video from camera, invoking FPGA to start processing the video frames, receiving the output video data from FPGA and sending commands via ethernet to iRobot. At this stage, we had complete design architecture that we wanted to implement. Then we started with code development to achieve gesture control on FPGA and command sending on ARM processor.

- **Wrote HLS code for corner detection of red object** to interpret gesture from the input video. As tracking red object's orientation to determine driving direction of iRobot is both easy and more interactive, we decided to choose this approach. We wrote the FPGA modules in C++ programming language to segregate the red object, denoise the image and determine the corners of this object. The main challenge in this development step was to denoise the image, so that we don't detect false corners of the object. The majority of the time spent on this was to determine the optimum solution for noise filtering, so that surrounding noise doesn't interfere with actual red color. For denoising, we implemented median filters over space (within a single frame) as well as over time (across frames) [2]. After doing corner detection, we went to the next step which was to interpret the corners sent by FPGA in PS.
- **Determined object inclination and sent command over ethernet using PS.** After development on PL fabric, the next step was to pass the result of PL to PS. We used compiler directives to handle the communication back and forth between PL and PS. This way, we received the corner information and output video from PL to PS. In PS, we developed a program which can calculate the angle of inclination based on the corner information. After calculating the angle, we used *arpa* C++ library to send packets over the Ethernet. As per our decision, we already had Galileo code at receiving end which we developed in the previous semester. This way modular design approach helped us save time in rewriting receiver end.
- **Benchmarked timing for FPGA and software implementation.** Finally, after meeting all the expectations of the project, we benchmarked the timing of different components to do analysis.

3 HLS Implementation

3.1 High Level Design

Our heterogeneous platform uses ARM Cortex 9 (@667 MHz) as the host processor and FPGA fabric as a video accelerator. The input frame from HDMI camera is first stored into video buffer hosted in main memory of FPGA. After the frame buffer is filled, PS invokes the FPGA fabric by passing the starting address of the frame. FPGA behaves as an accelerator and the moment PS gives a signal to FPGA, FPGA start processing the frame data in pixel-by-pixel manner. Once PL is done processing the whole frame, it raises a ready signal to PS informing that it's done with the processing and PS can take the data from PL. (Note that in our design, PL is taking video frame as an input and gives detected corners as well as video frame as an output.) The PS senses the flag and starts receiving corners and video out information from PL. The video out of PL is written into the video frame buffer, which will be read by Video DMA to output into HDMI screen. The corners sent by the PL is consumed by the PS and it generates command out of it.

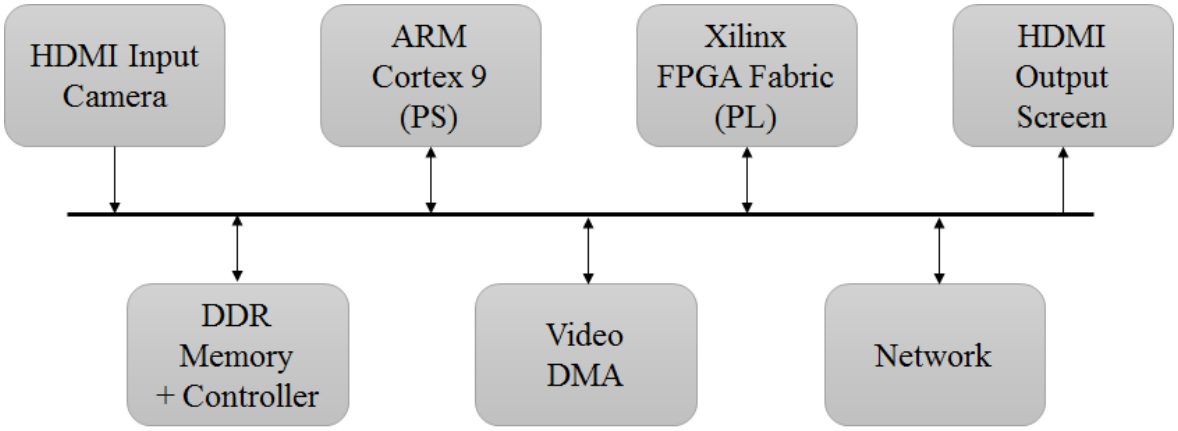


Figure 2: High level design of Heterogeneous core on Xilinx Zynq 702 FPGA

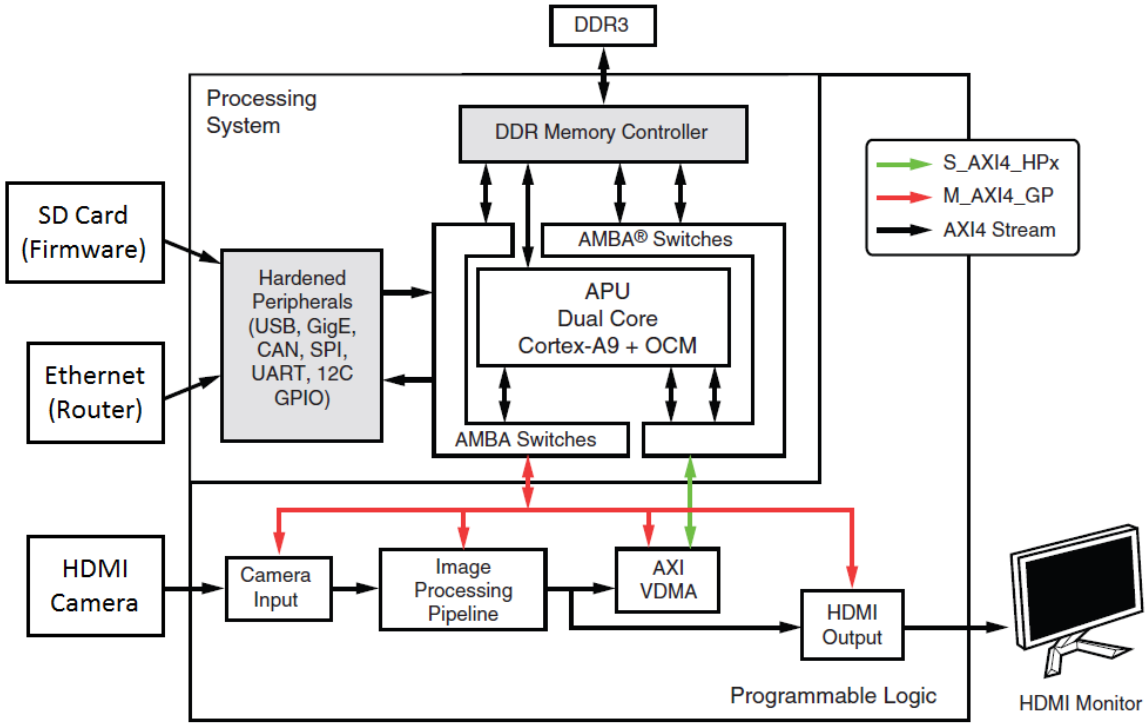


Figure 3: System Architecture of PS and PL in Xilinx Zynq FPGA. Adopted from Ref [15].

Figure 2 shows the high level design which are connected with FPGA. In Figure 3, the detail structure of the Xilinx FPGA is mentioned [15]. The processing system consists of DDR Processor, Memory controller and IO including Gigabit Ethernet. AMBA Switches are used for high speed interconnect between ARM Processor, PL and DDR Memory. Functionality of each of the components of Figure 2 is explained in following sub topics.

3.2 HDMI Input Camera

Our implementation supports HDMI 1920 x 1080 resolution with 60 frames per seconds. As an HDMI input, any source which can support that resolution can be considered. We have considered the HDMI output of laptop screen share to behave as a source of video, but it can be HDMI camera as well.

3.3 ARM Cortex 9 (PS)

Xilinx Zynq 702 evaluation board supports dual ARM Cortex 9 as an embedded processor which runs at 667 MHz clock [3]. Processor is capable of running PetaLinux operating system [4]. The high level flow of running PetaLinux on ARM is as follows:

- Store *.elf executable file along with the bootloader onto SD card [5].
- Insert SD card into Zynq board, configure Zynq board to load image from SD card [6] and power on the board.
- After power up, first stage boot loader will load the PetaLinux from SD card to RAM of FPGA.
- Bootloader will signal PS to boot the PetaLinux OS which is hosted purely on ARM.

In our project, bootloader is automatically created by SDSoC when we build the project. This flow indicates that we can essentially run most of all the program which are supported by normal linux OS. With this information, we have decided to implement video frame management, ethernet programming and trigonometric arithmetic calculation on PS using ARM core. When we execute our .elf file from PetaLinux, the application starts acquiring HDMI frame from input and starts storing it into one of the three video frame buffers of main memory in circular fashion (explained in next subsection). After storing a frame, it invokes PL to start doing processing. Once PL is done processing the video and has written output video into one of the frame buffer, PS acquires the corner of the detected object from PL. PS then calculates the inclination of the detected object using basic trigonometric equation shown in Equation (1). The visualization of the θ value can be interpreted from Figure 4. This value of θ is used to decide the turning radius of iRobot.

$$\theta = \tan^{-1} \frac{y_1 - y_2}{x_1 - x_2} \tag{1}$$

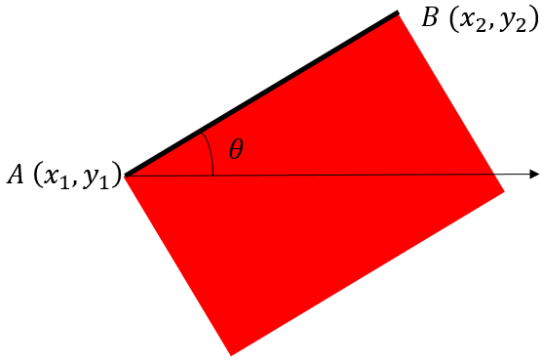


Figure 4: Object inclination Calculation using θ. Red object is the object which controls the iRobot.

The calculated θ value ranges is between -90 to 90 degree. This range is mapped into the actual turning radius required to control the Robot's direction (for further information about iRobot command structure, refer to Appendix X). The conversion from (-90, 90) degrees to (MAX_RIGHT_RADIUS, MAX_LEFT_RADIUS) is done via Equation (2) [7].

$$\beta = \beta_{min} + \frac{(\beta_{max} - \beta_{min})(L - L_{min})}{(L_{max} - L_{min})} \quad (2)$$

Where,

β_{min} = MAX_RIGHT_RADIUS

β_{max} = MAX_RIGHT_RADIUS

L_{min} = -90

L_{max} = +90

Similarly, to calculate the iRobot's front or backward speed, we are considering the area of the red object. If the area increases over time, then we increase the speed of iRobot in frontal direction. If the area decreases over time, then we decrease the speed of iRobot in frontal direction and eventually, we start issuing backward commands (iRobot commands are explained in Appendix X). The visual representation of this situation is represented in Figure 5 where the object is coming closer to the camera and hence we are increasing the speed of the iRobot. Here also, we are using Equation (2), to map max and min area of object to the maximum speed of iRobot in front to back direction.

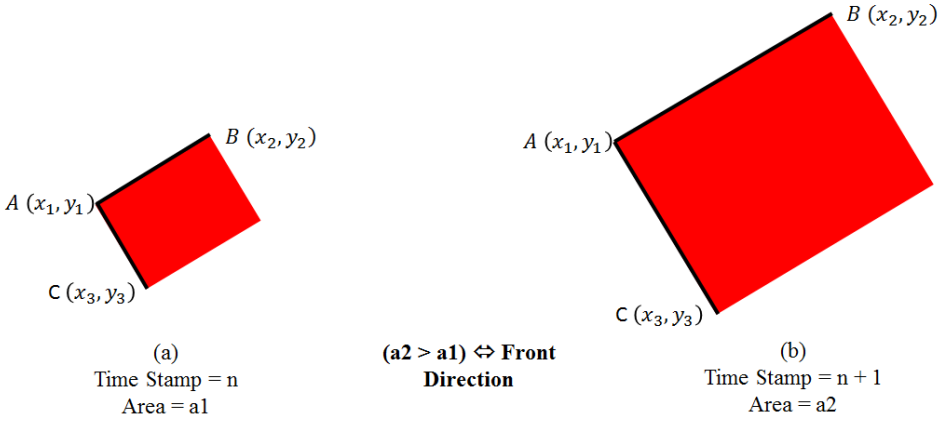


*Figure 5: Object movement in front and back direction corresponds to iRobot's speed in front and back direction. (a) Area a1 = AB*AC of red object on any timestamp n, (b) Area a2 = AB*AC of red object on the next timestamp n+1. If a2 > a1 then object is coming closer to the camera and hence it should issue front driving direction command to iRobot*

3.4 Xilinx FPGA fabric (PL)

Programmable logic is developed to work as an video accelerator. The video with resolution 1920*1080 is fed into the PL module. PL logic is fully pipelined and it processes 1 pixel per cycle with

Initiation Interval (II) = 1.The hardware modules for video processing is shown in Figure 6. Detailed explanation of each of the module is described in following sub sections.

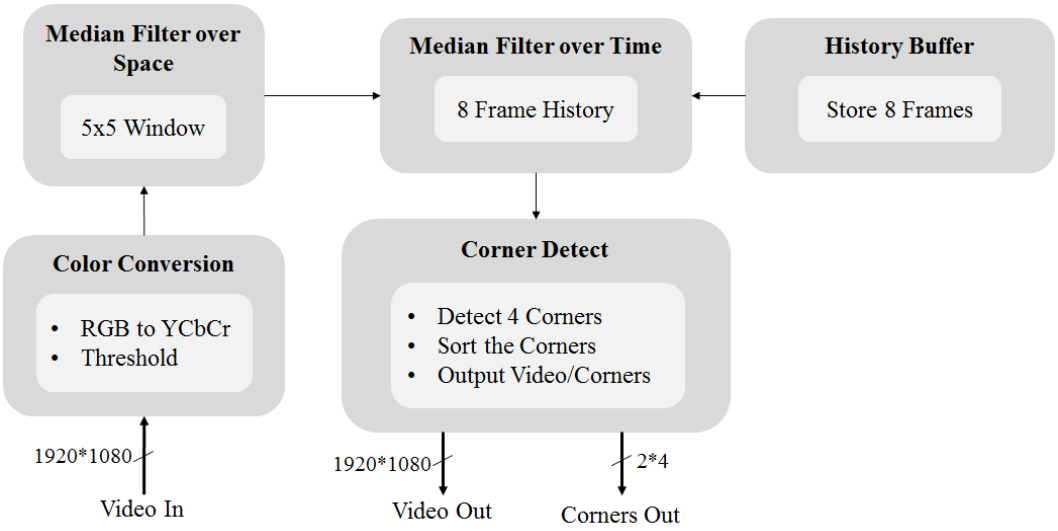


Figure 6: Corner Detection Algorithm on PL

Color Conversion: Color conversion is the first module in the pipeline chain. It's function is to convert the RGB color space into the YCbCr color space. YCbCr is a color space that separates the luminance (ie. light intensity) component of a frame from the hue components. Working in this color space is advantageous over the RGB color space because colors in YCbCr are represented independently from their relative brightnesses.The next step of this module is to threshold the red color and reject any color other than Red.

Median Filter over Space: In any video processing application, noise is the major component which we have to deal with. As we are dealing with HD videos with 60 frames per second, noisy red pixels in the video drastically deteriorates the corner detection algorithm's accuracy. To overcome this issue, we implemented median filter over space and median filter over time (next sub topic). Median filter over space uses 5x5 median filter window to reject the spurious red pixel on the screen. As median filter requires information of nearby pixels within 5x5 window, we had to come up with a way to store the history. Naive way of doing this would be to store all the content of the input video frame and then iterate over entire frame to do the median filtering. But as it slows down the performance significantly, we didn't use this approach. Instead, we used a concept of line buffering [12] [13] and windowing [14] to implement median filter in efficient manner, which not only saves the space for frame but also maintains II = 1. We kept the size of line buffer as 4 i.e. we store only last 4 rows of the frame from the current pixel. By doing that, we only require 4 * 1920 pixel line buffer and 5*5 pixel window buffer. The visual representation of how line buffering and windowing works for median filter is shown in Figure 7. Here, we keep on shifting line buffer down and window to the right side as and when the new input pixel keep on appending.

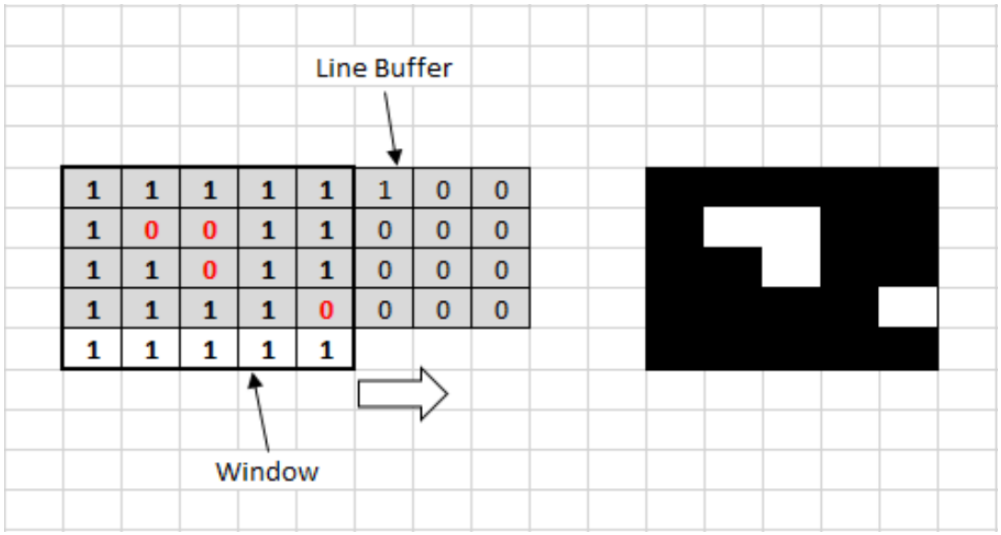


Figure 7: Line Buffering and 5x5 Windowing to process real-time video stream

Median Filter over Time: Median filter over space was not enough to get rid of most of the noise. So, we implemented median filter over time. In this filtering, we store the previous 8 frames of thresholded video on BRAM. While giving output from this module, we check the pixel value of past 8 frames at pixel (x, y) location. If majority of the pixels of last 8 frames is 1 then we output 1 otherwise we output 0. As an illustration, Figure 8 displays median filtering over 4 frames. Current pixel under consideration is (x, y) and the past4 values of pixel (x, y) was 1, 1, 1, 0 respectively. So the output of the median filter over time module will be 1 for pixel (x, y), as there are majority of 1's in the history.

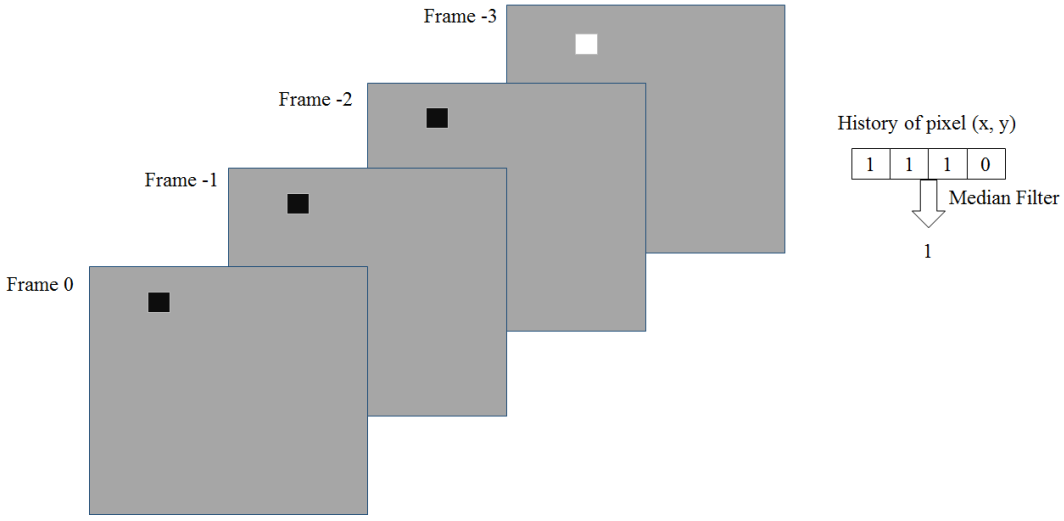


Figure 8: Median Filtering over 4 frames for pixel (x, y). If the pattern is 1110 for last 4 frames, then output will be 1.

History Buffer: History buffer is responsible for storing previous 8 frames on BRAM. The frames are stored in FIFO order.

Corner Detection: Corner detection module detects the extreme corners of the frame by finding X_min, X_max, Y_min and Y_max. It is also responsible to send output video back to PS along with detected corners. To reject the outlier corners within the frame, we are maintaining 8 frames' corner history and doing bubble sorting on those 8 frames' corners. The mid point of all sorted corners (which is the median filter of corners) is given as an output corner along with video output.

3.5 DDR Memory and Controller

The purpose of DDR memory is to (1) load the PetaLinux boot image after booting up from SD card and (2) maintain the video frame buffer which stores video frames coming from camera and going to monitor. In our design, we are using 3 video frame buffers to circulate the read/write access [15]. 3 frame buffers are used in such a way that while the PS is writing video frame into one buffer, PL will read the video frame from the previous buffer. Frame buffers, implemented in this way assures that there is not read/write contention between FPGA and ARM processor. To manage the DDR memory access HLS uses Memory controller which arbitrates reads/writes to the physical DDR memory when asked for data.

3.6 Video DMA

Xilinx LogiCORE IP AXI DMA is a soft IP provided by Xilinx to manage high-bandwidth direct memory access between memory and AXI4-stream video specific target peripherals [16]. The high level block diagram of AXI VDMA is shown in Figure 9. The Memory mapped DMA access gets initialized by ARM processor over AXI4-Lite bus. After ARM has registered all necessary data for transfer, Control and Status Module will start following orders by doing the data transfer to/from AXI4 Memory map to AXI-4 Stream.

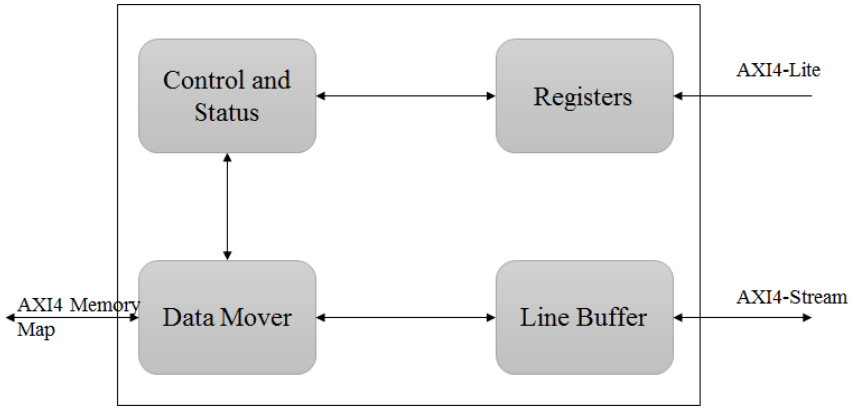


Figure 9: AXI VDMA Block Diagram. Adopted from Ref [16].

In the Write path, the AXI VDMA accepts frames on the AXI4-Stream Slave interface and writes it to system memory using the AXI4 Master interface.In the Read path, the AXI VDMA uses the AXI4 Master interface for reading frames from system memory and outputs it on the AXI4-Stream Master interface. Both write and read paths operate independently. The AXI VDMA also provides an option to synchronize the incoming/outgoing frames with an external synchronization signal.

3.7 Video Output Screen

Once PL has written back the processed frame onto the video frame buffer, PS instructs VDMA to stream video frame to the output. As a result, the frame is transferred to the video output screen via HDMI port.

3.8 Network

Once the PS is done calculating the orientation and speed of the iRobot, we use *arpa* [17] library and sys/socket.h header file to send a packet to the network. This code is executed on PetaLinux hosted on PS, so it's similar to writing C++ code for sending network packets on linux. The network packets are sent to the TP-Link wireless access point using Ethernet connection. We hosted a private network for our communication on wireless access point. (See Appendix E.4 for further instruction on setting up network on TP-Link).

3.9 Design, Compilation and System Linking Flow

We used Xilinx SDSoC Design suite to develop the application which can exploit both ARM core and FPGA. The design flow for developing heterogeneous system on SDSoC is summarized in Figure 10. The step wise instruction for development is mentioned below.

- Write normal C/C++ code which implements the functionality.
- Offload some of the work load to FPGA by providing appropriate compilation flags in the makefile. It would be prudent to offload only the work which is more efficient on FPGA and not on ARM.
- Compile the code using make script. On compilation, SDSoC invokes Xilinx Vivado HLS to cross compile the FPGA code. It compiles ARM code in native gcc compiler. If any error is found in the code, compilation exits with an error.
- Once all the errors(if any arise) are fixed, compilation succeeds and then SDSoC does the code analysis and determines the communication and data pattern between FPGA and main memory.
- After code analysis, it implements proprietary stub function to establish communication.
- After stub implementation, SDSoC calls system linker which links appropriate IP modules and invokes Xilinx Vivado for synthesis.
- After the successful synthesis, SDSoC places the final package including bootloader, .elf executable file on the sd_card folder.
- We can copy the content of this folder onto SD card and boot Xilinx Zynq from SD card to run the application.
- Based on the performance of the application, we can do further HLS optimization (more details in Appendix A) as well as function distribution between ARM and FPGA.

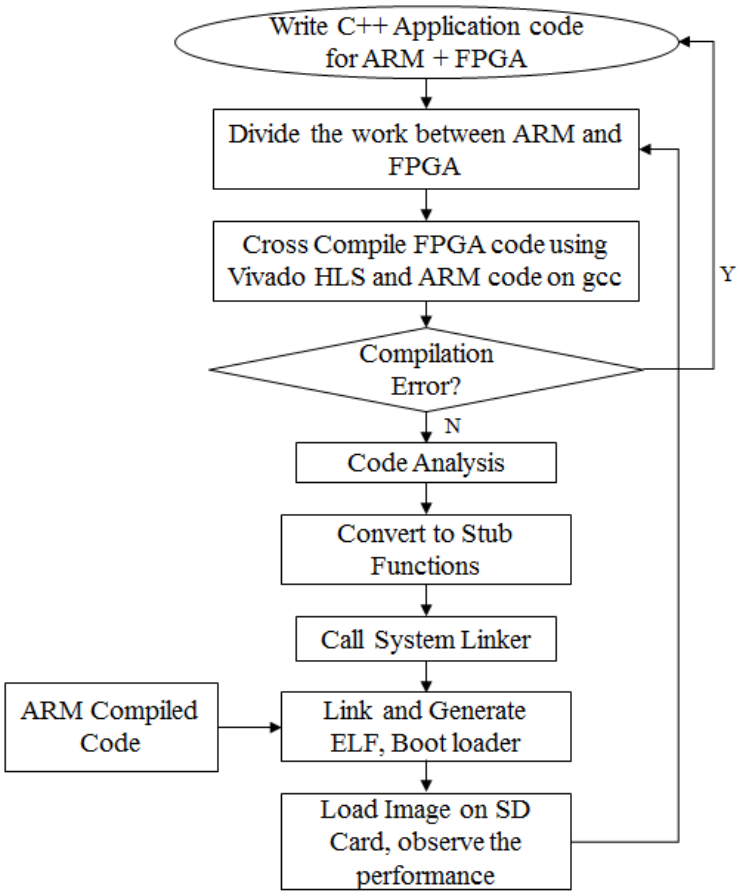


Figure 10: Development Flow of Heterogeneous core using SDSoC.

4 Results

As per the design, we are able to drive the iRobot successfully based on the object's orientation in different directions. FPGA is successfully able to process HD frames in real-time at 60 FPS and is able to send commands to the iRobot.

Comparing the HLS design with the software version which we implemented previously, we found that HLS performs far better than its software counterpart. We ran software version of the project on Intel i3 Dual core processor running at 1.90 GHz frequency. The benchmark results of software vs HLS implementation are shown in Table 1.

Table 1: Video Processing Unit's Specification and Performance matrix

Design Type	**Resolution**	**FPGA CLK**	**Processor CLK**	**Single Frame Processing Time (ms)**	**FPS**
HLS	1920*1080 (HD)	168 MHz	667 MHz	17.99	60
Software	640 * 480 (SD)	-	1.90 GHz	77.00	13

With the software implementation, we were only able to process 13 frames of Standard Definition (SD) video even at 1.90 GHz clock. The reason being the general purpose processor architectures are not designed to be highly optimized for video applications, but designed to perform all the tasks in reasonably good speed. Whereas using HLS, we are able to process exactly 60 frames per seconds for HDMI video and time for processing single frame is just 18 ms. This timing includes the communication overhead from transferring data from ARM to FPGA and back. This is almost 4.3X of performance boost as compared to General purpose processor. The result was exactly matching our expectation and specification which says that Heterogeneous core should be able to perform faster than the software version of the video processing.
The resource utilization of the Zynq was not too high, which also shows an advantage of using HLS. The module wise and overall resource utilization of different FPGA components is shown in Table 2.

Table 2: FPGA Resource Utilization in High Level Digital Synthesis

FPGA Video Accelerator Modules	**Resource Utilization**		
	BRAM (%)	**FF (%)**	**LUT (%)**
RGB to YCbCr	0	0	0
Median Filter (space)	0	0	1
Median Filter (time)	20	0	1
Corner Detect	0	3	15
Total	20	3	17

Finally, the effort estimation for developing the software version of the project vs hardware version of the project is shown in graph of Figure 11. The development time of the HLS is more than that of software development, partly because software development is at very high level abstraction without considering any of the hardware implication, whereas HLS implementation does require hardware knowledge in order to implement efficient version of the hardware. One interesting thing to note here is that testing and debugging effort in HLS was significantly higher in our case, because we didn't have well developed test bench which tests the HLS code before synthesis. As a result, we ended us spending more time on debugging the application for most of the time. For testing, we were using Visual studio to create some sample application code and verify on the visual studio itself. Once we know the stability of the function, we copied those codes to HLS for synthesis. To debug the application, we were inserting some visual representation on the output screen. This way we were able to know whether something is wrong and if so, in which condition the issue is occurring.

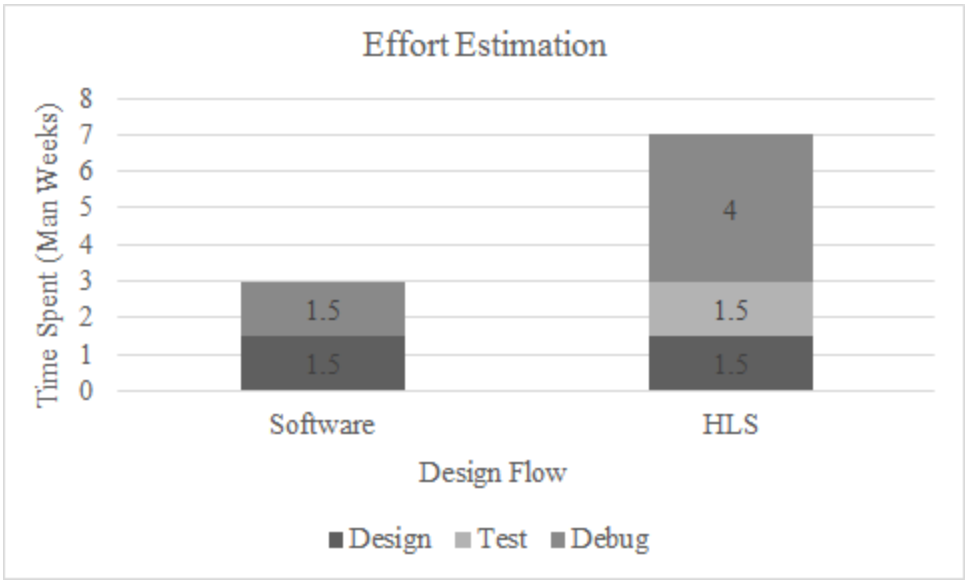


Figure 11: Efforts required to develop software version vs HLS version of the project.

5 Conclusion and Future Scope

In our project, the performance of HLS version of the video processing is 4.3X faster than the software version of the video processing, but at the cost of increasing development time of 2.3X. With rich testbench framework, the development time of the HLS can be significantly brought down close to that of the software implementation. In such scenarios, HLS provides benefits of both the worlds, one world where performance is of utmost importance and the other is where time to market is of great importance.

Future direction of the project would be to develop a testing framework which verifies the hardware design before synthesis. This project can also be used as a platform to develop different computer vision algorithms targeting various robotic applications.
Our current project has one issue and possible extensions which are listed below:

- FPGA sometimes crashes in one corner case scenario (which we don't know) and due to that iRobot stops taking any further commands from FPGA and retains previous velocity and radius. To prevent that, we are rebooting the FPGA.
- Currently, we are outputting only detected red object into white color and everything else into black color. One extension to the project can be to overlay the input video with the detected object.
- The other extension can be to use wifi dongle on FPGA instead of ethernet. For further information, refer to link [23].

6 Acknowledgements

We would like to thank our advisor Prof. Zhiru Zhang for his constant guidance and encouragement. His insights and technical guidance helped us resolve many issues. We are grateful to Steve Dai for his support throughout the project and Julie Wang for her detailed documentation explaining corner detection algorithm on FPGA. We also appreciate Xilinx Inc. for providing SDK tools and numerous guidelines that have helped us in our project implementation.

7 Reference

[1] Gesture Controlled iRobot with Intel Galileo, *Qiukai Lin*.

[2] Bruce in the Box. *Julie Wang*. Link: http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/s2014/jsw267/html/html/

[3] ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC. Link: http://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf

[4] PetaLinux Operating System. Link: http://www.wiki.xilinx.com/PetaLinux

[5] PetaLinux Getting Started. http://www.wiki.xilinx.com/PetaLinux+Getting+Started

[6] *Boot Pre-Built Xilinx ZC-702 image.* Link: http://www.wiki.xilinx.com/Boot+Pre-Built+Xilinx+ZC-702+image

[7] *Bit Rate Throttling Algorithm on Video over RTP*. M. Y. Modi, S. Kasula. Link: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6780120&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6780120

[8] OpenCV tutorial. Link: http://opencv-srf.blogspot.com/2013/05/installing-configuring-opencv-with-vs.html

[9] Configuring OpenCv and Visual Studio. Link: https://www.ibm.com/developerworks/community/blogs/theTechTrek/entry/installing_and_using_opencv_with_visual_studio_2010_express1?lang=en

[10] Intel Galileo. Link: http://www.arduino.cc/en/ArduinoCertified/IntelGalileo

[11] iRobot Create Manual. Link: http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf

[12] Implementing Memory Structures for Vivado Processing in the Vivado HLS Tool. Link: http://www.xilinx.com/support/documentation/application_notes/xapp793-memory-structures-video-vivado-hls.pdf

[13] HLS Line Buffer. Link: http://www.wiki.xilinx.com/HLS+LineBuffer

[14] HLS Window. Link: http://www.wiki.xilinx.com/HLS+Window

[15] 1080p60 Camera Image Processing Reference Design. *Mario Bergeron (Avnet, Inc.), Steve Elzinga, Gabor Szedo, Greg Jewett, and Tom Hill (Xilinx, Inc.)*. Link: http://www.xilinx.com/support/documentation/application_notes/xapp794-1080p60-camera.pdf
[16] AXI Video Direct Memory Access v6.2. Link: http://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf
[17] ARPA library. Link: http://pubs.opengroup.org/onlinepubs/7908799/xns/arpainet.h.html
[18] MEng Poster Session 2015. Link: http://www.ece.cornell.edu/ece/academics/graduate/meng/meng-poster-session-2015.cfm
[19] C-Based Synthesis. Course ECE 5775, High Level Digital Design Automation. Prof. Zhiru Zhang. Link: http://www.csl.cornell.edu/courses/ece5775/pdf/hls.2up.pdf
[20] Vivado High Level Synthesis. Link: http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html
[21] SDSoC development environment - Xilinx. Link: http://www.xilinx.com/publications/prod_mktg/sdnet/sdsoc-development-environment-backgrounder.pdf
[22] Configuring TP-link router. Link: http://www.tp-link.com/en/FAQ-417.html
[23] Xilinx Wiki - Zynq SDIO WiFi. Link: http://www.wiki.xilinx.com/Zynq+SDIO+WiFi?responseToken=04346356e336d23fc5197047c17374fe1

8 Appendix

8.1 Appendix A: High Level Synthesis

High-Level Synthesis is a C specification of Register Transfer Level implementation. It takes C program as an input, passes through the High-Level Synthesis flow and gives synthesizable RTL as an output. HLS bridges the software and hardware domains.

- It allows hardware designers who implement designs in an FPGA to take advantage of productivity benefits of working at a higher level of abstraction, while creating high-performance hardware.
- It provides software develops with an easy way to accelerate the computationally intensive parts of their algorithms on a new compilation target, the FPGA provides a massively paralleled architecture with benefits in performance, cost and power over traditional processors.

The high level design flow for HLS is shown in Figure 12.

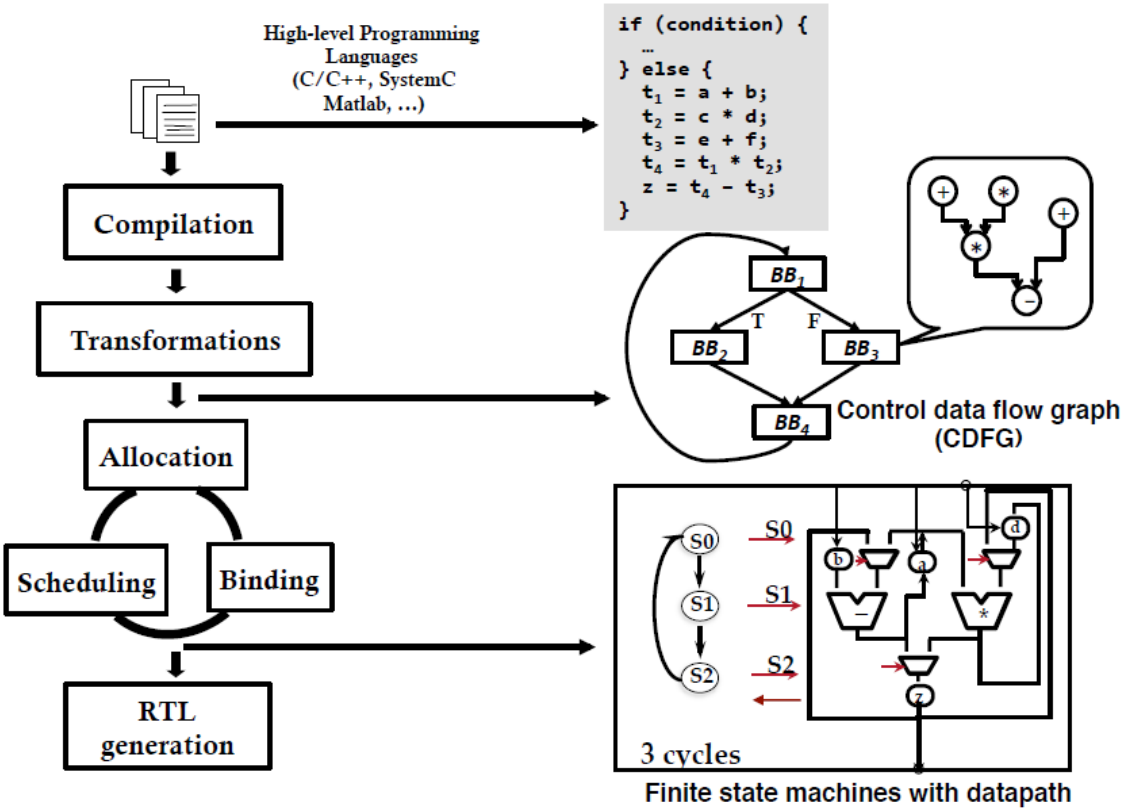


Figure 12: High Level Synthesis Flow. Adopted from Ref [19]

The entry point to the HLS flow is the high level programming code. Based on the functional specification provided by the code, HLS tools first compile the code to see whether there is any compilation errors or not. If there are no compilation errors, then Compiled code passes through the transformation phase. Transformation phase converts the functional level code into control flow graph (CFG) and data flow graph (DFG). The transformed code is then passed to Allocation stage where HLS tools allocates physical resources required for CDFG. The scheduling stage after Allocation determines when to schedule and operation (which cycle execute which operation). Binding binds operations, variable or data-transfers to the available resources. Finally RTL netlist is generated which can be uploaded onto FPGA to execute the expected behavior.

8.2 Appendix B: Xilinx Design Suite

8.2.1 Xilinx Vivado Design Suite

Vivado Design Suite, Vivado High-Level Synthesis accelerates IP creation by enabling C, C++ and SystemC specifications to be directly targeted into Xilinx All Programmable devices without the need to manually create RTL [20]. Vivado HLS provides system and design architects alike with a faster path to IP creation by:

- Abstraction of algorithmic description, data type specification (integer, fixed-point or floating-point) and interfaces (FIFO, AXI4, AXI4-Lite, AXI4-Stream)
- Directives driven architecture-aware synthesis that delivers the best possible QoR
- Fast time to QoR that rivals hand-coded RTL

- Accelerated verification using C/C++ test bench simulation, automatic VHDL or Verilog simulation and testbench generation
- Multi-language support and the broadest language coverage in the industry
- Automatic use of Xilinx on-chip memories, DSP elements and floating-point library.

8.2.2 Xilinx SDSoC Design Suite:

Xilinx SDSoC is a wrapper to the Xilinx Vivado design suite and allows heterogeneous computing to exploit hardware-software co-design. It uses Vivado to generate bitstream for FPGA and uses GCC compiler to generate executable for ARM processor. The high level design flow for SDSoC is shown in Figure 13.

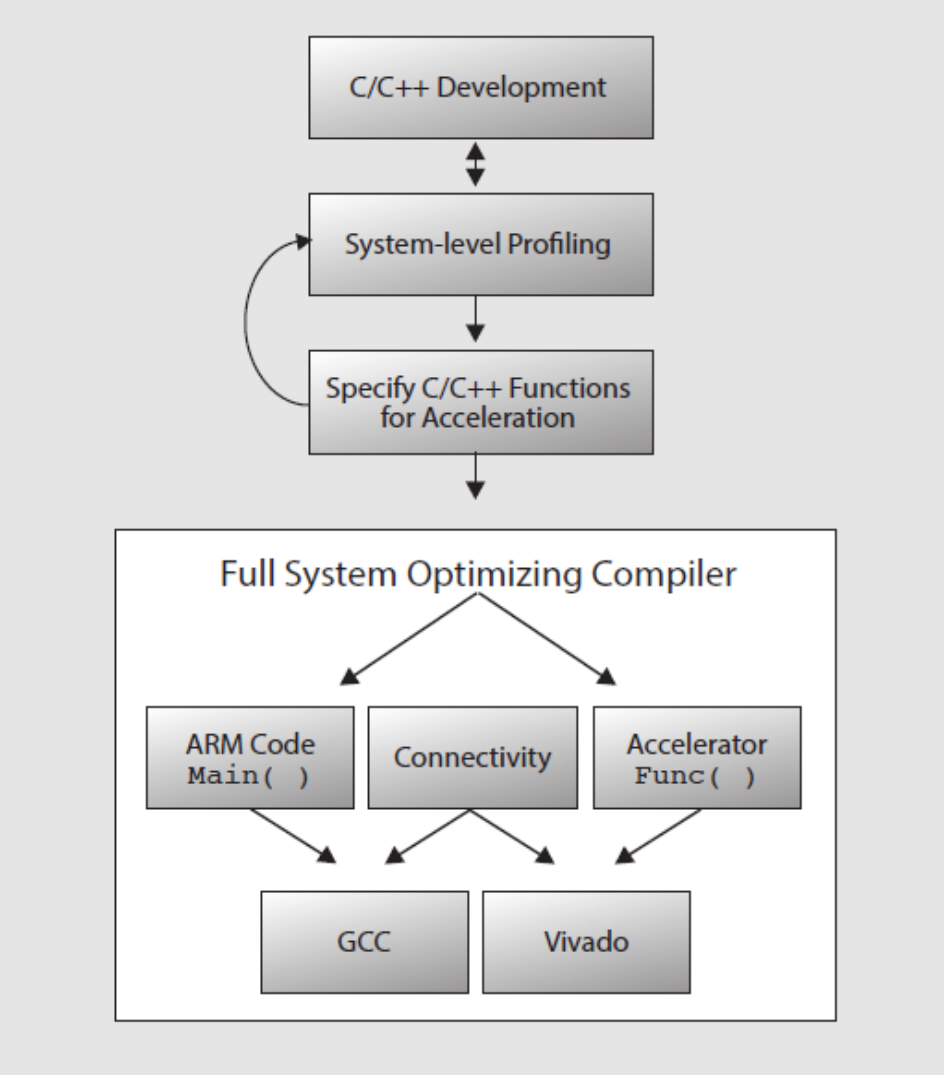


Figure 13: Design flow Xilinx SDSoC. Adopted from [21]

8.3 Appendix C: iRobot Create

iRobot Create is an affordable, preassembled mobile robot platform that provides an out-of-the-box opportunity for educators, students and developers to program behaviors, sounds, movements and add

additional electronics. From beginner developers to advanced robotics students, Create allows for a variety of programming methods and is totally compatible with any computer or microcontroller. The Create also has a 7-pin Mini-DIN serial port through which sensor data can be read and motor commands can be issued using the iRobot Create Open Interface. The software interface lets us manipulate Create's behavior and read its sensors through a series of commands including mode commands, actuator commands, song commands, demo commands, and sensor commands that you send to Create's serial port by way of a PC or microcontroller that is connected to the Mini-DIN connector or Cargo Bay Connector. The detailed description of iRobot Create along with the Open Interface commands that help in driving the robot are provided in the manual [12].

In order to drive the robot in the designated direction with a particular speed, we have generated commands in the Create Open Interface format and these commands are sent over the network from the processing unit. Each command starts with an opcode that determines what instruction the robot has to execute and is followed by relevant data bytes that will enable smooth controlling of the robot. For example, the opcode 128 starts the Open Interface and this is the first instruction sent to the iRobot. Apart from the commands that are used to start the OI and get it ready for use, there are also other commands that control the iRobot Create's actuators like wheels, LEDs, digital outputs etc. For example, the opcode 137 is the drive command that controls Create's drive wheels. This drive command takes four data bytes, interpreted as two 16-bit signed values using two's complement. The first two bytes specify the average velocity of the drive wheels in millimeters per second (mm/s), with the high byte being sent first. The next two bytes specify the radius in millimeters at which Create will turn. The longer radii make Create drive straighter, while the shorter radii make Create turn more. The radius is measured from the center of the turning circle to the center of Create. A Drive command with a positive velocity and a positive radius makes Create drive forward while turning toward the left. A negative radius makes Create turn toward the right. A negative velocity makes Create drive backward. The drive instruction is the most important OI command that we have used to control the motion of the robot. The actual notation of the drive command is as follows and is also shown in the figure below.

Drive: [137] [Velocity high byte] [Velocity low byte] [Radius high byte] [Radius low byte]

8.4 Appendix D: Intel Galileo

Galileo is a microcontroller board based on the Intel® Quark SoC X1000 Application Processor, a 32-bit Intel Pentium-class system on a chip. This platform provides the ease of Intel architecture development through support for the Microsoft Windows*, Mac OS*, and Linux* host operating systems. The Intel Galileo board is also software-compatible with the Arduino software development environment has several PC industry standard I/O ports and features to expand native usage and capabilities beyond the Arduino shield ecosystem. A full-sized mini-PCI Express* slot, 100 Mb Ethernet port, Micro-SD slot, 6-pin 3.3V USB TTL UART header, USB host port, USB client port, and 8 Mbyte NOR Flash* come standard on the board. Galileo has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. Galileo provides UART TTL (5V/3.3V) serial communication, which is available on digital pin 0 (RX) and 1 (TX). In addition, a second UART provides RS-232 support and is connected via a 3.5mm jack. The USB Device ports allows for serial (CDC) communications over USB. This provides a serial connection to the Serial Monitor or other applications on your computer. Ref. [11] gives further details about Intel Galileo

board and how it can be used for various applications. The figure below shows how the commands generated by the processing unit are received over the wireless network using the Intel N-135 wireless receptor that is connected to the Galileo board (Ref. to appendix E.3) and these commands are then transmitted to the iRobot via serial communication.

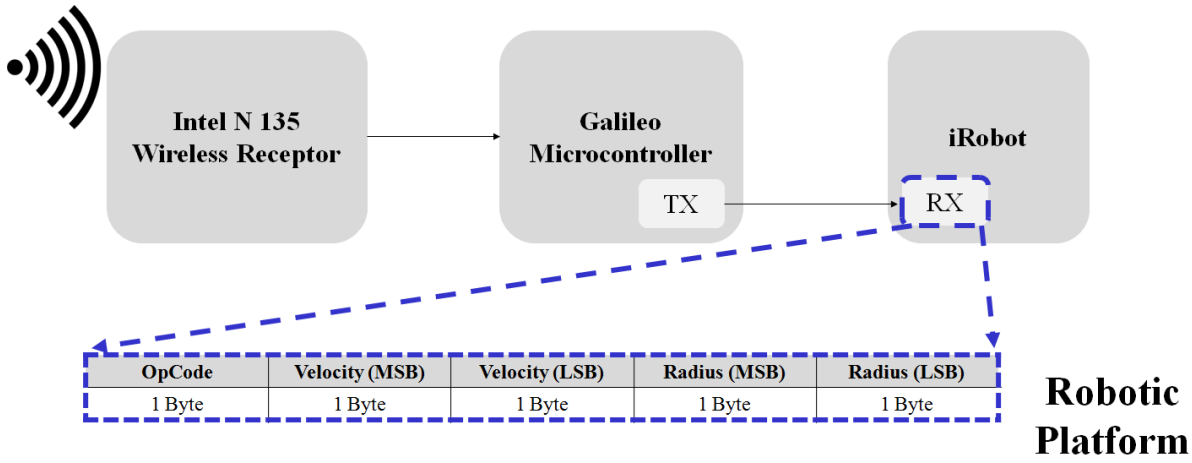


Figure: Command propagation from the wireless network to the iRobot via the Intel N 135 wireless receptor connected to Galileo board

8.5 Appendix E: Software Setup Instructions

8.5.1 Setting up Xilinx SDSoC

Step 1: To setup Xilinx SDSoC. Copy following content into .bashrc file in zhang-01 research group server.

```
export XILINXD_LICENSE_FILE=2100@flex.ece.cornell.edu

################################################################################
# Xilinx ISE
################################################################################
source /work/zhang/common/tools/xilinx/14.7/ISE_DS/settings64.sh

################################################################################
# Xilinx Vivado
################################################################################
source /work/zhang/common/tools/xilinx/Vivado/2014.1/settings64.sh
# 5775
export PATH=$PATH:/work/zhang/users/dai/llvm_src/build/Debug+Asserts/bin
export PATH=$PATH:.
export PATH=$PATH:/work/zhang/common/tools/CPLEX_Studio_Full/cplex/bin/x86-64_sles10_4.1

################################################################################
# SDIntegrator
################################################################################
```

source /work/zhang/common/tools/xilinx/SDIntegrator/2014.2/settings64.sh
export XILINXD_LICENSE_FILE=$XILINXD_LICENSE_FILE:/work/zhang/common/tools/xilinx/xilinx-apf-sdi.lic

Step 2: Go to <current_project_dictionary>/ Release folder. Type *make all*. It invokes the HLS tool chain from SDSoC.

Step 3: To compile newly added files along add that file to <current_project_dictionary>/ Release/ makefile.

Step 4: To move any function from ARM to FPGA, modify <current_project_directory>/ Release/ hw/ subdir.mk file and add -sdi -hw directive along with the function name.

8.5.2 Configuring OpenCV with Visual Studio

Step 1: Download and install Visual Studio Express (free) version to be used as the IDE.
Download OpenCV from http://opencv.org/ and extract the files to C:\ directory.

Step 2: In order to use OpenCV libraries the environment variables have to be setup properly.To do so, follow the instructions in [8] .

Step 3: Once the OpenCV environment is setup, we now need to make necessary changes in Visual Studio so that we can incorporate OpenCV libraries in our programs.

- Go to File>New>Project>Visual C++>Win32>Win32 Console Application. Enter the name of a sample program(eg. first) and click finish.
- Include the any relevant OpenCV code into first.cpp
- Include the following 3 directories under: *Note: Use the appropriate version of OpenCV in the below paths as per the version you downloaded*
    - Project->first properties->VC++ Directories->Include Directories. Click Edit
        - add 'C:\OpenCV2.1\include\opencv'; click Apply and OK

      Project->first properties->VC++ Directories->Library Directories. Click Edit
        - add 'C:\OpenCV2.1\lib' ; click Apply and OK

      Project->first properties->VC++ Directories->Source Directories. Click Edit.

        - add 'C:\OpenCV2.1\src\cv; C:\OpenCV2.1\src\cvaux; C:\OpenCV2.1\src\cxcore; C:\OpenCV2.1\src\highgui; C:\OpenCV2.1\src\ml'

    - The next step is use include all the OpenCV libraries at the linking phase

      Project->first properties->Linker->Input->Additional Dependencies

        - Cut and paste the .lib file names mentioned in [9] by clicking on Edit
        - Once all the library files are added, click Apply and Ok
- Now you can build the program by choosing Debug->Build Solution

Use the references [8] and [9] to get click-by-click help for the above instructions. Once the program builds successfully and gives the correct output, we can proceed writing programs for our specific application.

8.5.3 Setting up Intel Galileo

There are three essential components of setting up the Galileo in our design. First is the communication between Arduino software and Galileo board; Second is the connection between Galileo and the iRobot; Third is the configuration of Wi-Fi module on the Galileo. The detailed instructions for each of these arrangements are mentioned in [1]. They are however described briefly below:

Communication between Arduino IDE and Galileo:

The following instructions are based on Windows. Basic steps are:

1. Download and install Arduino IDE and Linux image (*link is mentioned in Ref. [1]*)
2. Connect power and USB Client port (closer next to Ethernet) to your computer through serial cable, as shown below. Remember to wait until USB LED lights up to connect the cable, otherwise you may damage the board!
3. Install Drivers and software
   - In your computer, follow **Start>Control Panel>System>DeviceManager**
   - Go down and find **Port**. If not, go to **Other Devices**.
   - You should see **Gadget Serial V2.4**. Right click and select **Update Driver Software**.
   - Choose **Browse my computer for Driver software**
   - Navigate to **hardware/arduino/x86/tools** directory
4. Update firmware
   - Launch Arduino IDE. It is under folder **Arduino-1.5.3**, double click **arduino.exe**
   - Under **Help** tab, select **Firmware Update**.

   Note: You should have power connected all the time. Do not have SD card plugged in at this time.
5. Upload programs
   - You can select **Board** and **Serial Port** under **Tools**.
   - You can go to **Files>Examples** for numerous sample codes already written with helpful comments. You can also upload them to Intel Galileo in order to see how things work

In this way, we can write programs relevant to our application in the Arduino software and upload them onto the Galileo board.

Connection between Galileo and iRobot

1. Connect Pin 1 (RXD) on Galileo to Pin 2 (TXD) on iRobot and Pin 0 on Galileo receives data, while Pin 2 on iRobot sends data
2. Connect Pin 2 (TXD) on Galileo to Pin 1 (RXD) on iRobot and Pin 1 on Galileo sends data, while Pin 2 on iRobot receives data
3. Connect GND pin on Galileo to Pin 14 (GND) on iRobot

   Note: Pins on iRobot start with Pin 1 until Pin 25, while digital pins on Galileo start with Pin 0. There are more than 1 GND pins on iRobot. Both of them work.

Configure Wi-Fi module on Galileo board

1. Attach the N-135 Wi-Fi card to Galileo board as described in [1].
2. Unzip Linux image and transfer everything to SD card
   - Plug in the SD card into your computer. Unzip the files and upload to the SD card directly. Preserve the folder structure.
3. After booting SD card, we can plug it into Intel Galileo, and then we can power on the board through 5V power cable.
4. To connect the Galileo to a wireless network, there are two ways depending on what is considered as our access point. In case the access point is a wireless router then Appendix section E.4 has to be followed and the Galileo should be connected to the same SSID as that of the router. If the access point is a laptop, then the instructions given in [1] have to be followed to host a WPA on the laptop and connect the Galileo wirelessly to it.

8.5.4 Setting up the network router

Ref. [22] will give a detail description of how to configure the TP-link router to act like an access point. The steps are briefly mentioned below.

Step 1:

Connect your computer to the router through the LAN port using Ethernet cable, then login to TP-LINK web interface through the IP address on the bottom label

As of now the router is set at a static IP of 192.168.137.1. By entering this http://192.168.137.1 you will be able to connect to the router interface

Step 2:

Go to Network ->LAN on the left side menu, you can change the LAN IP address of TP-LINK router here as per your requirement.

Step 3:

Configure the wireless

Go to Wireless->Wireless Settings page, configure the SSID (Network name) and Channel. (If it is a dual band device, please don't forget to select the Band.) Click Save button.

Go to Wireless->Wireless Security page, configure the wireless security. Here WPA-PSK/AES is recommended. PSK Password is the key of your wireless.

Step 4:

Go to DHCP on the left side menu, disable the DHCP Server and click Save button.

Step 5:

Go to System Tools->Reboot page, click Reboot button to reboot the device.

Step 6:

Connect the main router to the LAN port on TP-Link router through Ethernet cable.

After all of the tips above, your computer would have internet through any LAN ports of TP-LINK by an Ethernet cable. At the same time your wireless client would have access to the TP-LINK wirelessly through the SSID and Password

As of now the router is set with a SSID as 'irobot' and password as 'irobotirobot'. Only with these credentials any device will be able to wirelessly connect to the irobot private network.