A Study on Real-time Image Processing using Field Programmable Gate Arrays with Random Sampling Techniques

December, 2017

Nagasaki University, Graduate School of Engineering

Theint Theint Thu

Abstract

With the rapid development of image processing technology and the popularity of computer applications, real-time image processing systems become widely applied in the field of electronic technology. According to Moore's law, the energy efficiency plays a key role in embedded computing systems. Field Programmable Gate Arrays (FPGAs) are getting popular in the name of high energy efficiency, that is to say, a good energy performance ratio. The main theme of this dissertation is to find the best architecture on FPGAs using random sampling techniques based on stream architecture with many tradeoff analyses. At the level of abstraction where hardware logic is coded in a Hardware Description Language (HDL), design constraints and optimization become critical issues for meeting the design requirements. While random sampling is a critical issue in numerous applications such as cryptography, robotics, computer vision, machine learning, etc., the implementation of FPGA-based random sampling techniques has not been addressed very much so far, since it is not so straightforward to be pipelined. Hence, this dissertation concentrates on high-performance and compact architecture for real-time image processing on FPGAs with random sampling techniques. This dissertation also focuses stream-oriented processing which brings benefits to fast and low power FPGA-based parallel processing for real-time applications without using any external memory.

Chapter 1 introduces the conceptual basis for reconfigurable systems and parallel processing. Chapter 2 provides background of the study. In Chapter 3, overview of the research methodologies are presented. Chapter 4 discusses the real-time FPGA implementation of equation solvers for RANdom SAmple Consensus (RANSAC). In FPGA implementation of figural model estimations and fitting from video images, a well-known and effective algorithm called RANSAC employs various computational approaches to solve simultaneous equations for each model. In order to successfully estimate the reasonable parameters for model fittings, the hypothesis generation process with the least square method and compared four types of solver architectures were implemented. In contrast to large matrix solvers, few attention has been paid so far to small matrix manipulation on FPGA, and it is a question what kind of approach is appropriate for FPGA.

The several kinds of FPGA implementation of equation solvers are compared to reveal the tradeoff relationship among different methods when it comes to the matrix manipulation on an FPGA. The four solver modules are Cramer's rule (CRAMER) with long-integer arithmetic, CRAMER with double precision floating point (FP) arithmetic, Gauss-Jordan elimination (GAUSS) with single precision FP arithmetic and GAUSS with double precision FP arithmetic. While the Gauss-Jordan method is one of the most popular algorithms for this purpose, the Cramer's rule is generally never considered due to a large number of multiplication operations to be made in

the process of calculation of the determinant. However, especially for a small matrix size, execution performance of an algorithm becomes more sensitive to architectures and does not necessarily reflect computational complexity.

The advantage of the Cramer's method in FPGA implementation is that it has a simple and regular control flow. In addition, most calculations can be performed with integer arithmetic rather than floating-point arithmetic. On the other hand, a wide dynamic range is required to calculate the determinants of a given matrix. For the sake of revealing how the arithmetic type impacts the performance and hardware costs, two arithmetic implementation for the Cramer's rule: double precision floating-point and long integer are compared by changing the size of matrices.

In the evaluation experiments, each type of hardware solved the simultaneous equations repeatedly for synthetic benchmark models for circles, aligned ellipses, and ellipses with angles. The four solvers were compared the comparison criteria in terms of estimation accuracy, performance and resource usage. Depending on the maximum bit width for each multiplication step, the number of bits is changed to optimize the resource usages of long integer arithmetics. Experimental evaluation showed a Cramer's rule approach coupled with long integer arithmetic can reduce most hardware resources without unacceptable degradation of estimation accuracy compared to other versions.

Chapter 5 deals with the real-time FPGA implementation of the posterior system state estimation in dynamic state-space models using a particle filter. Although various researchers have attempted to apply the particle filter for real-time FPGA implementation, fully parallelizable particle filter implementation for high performance real-time object tracking is still an open problem in many settings, including security camera, endoscopic instruments, sports, and so on. Parallel resampling is of critical importance in stream image processing. Difficulty of parallel resampling comes from it needs information of all particles and thus computational dependency arises.

To cope with this problem, the dissertation proposes an architecture which combines FPGA optimized resampling (FO-resampling) method and an object tracking pipeline for FPGAs. The resampling step is performed in a synchronized area of an input image frame to achieve the performance of more than 60 FPS for VGA images without using any external memory. The results revealed that a full cycle of particle filtering including prediction, likelihood calculation and resampling can be fully parallelizable on FPGA. Therefore, the particle filter design can be fitted in a smaller FPGA chip, by reusing hardware resources in a time-sharing manner with a higher clock frequency. With changing the number of particles in multinomial resampling and FPGA optimized resampling, the resource utilization, performance and accuracy of real-time object tracking system were evaluated. The number of real particles (M=100) and virtual particles (B=50) is promised a better solution for the tradeoff analysis of hardware resources and required criteria. According to the performance comparison between two resampling methods, the evaluation result shows that FO-resampling is far superior to multinomial resampling in terms of accuracy and performance. Moreover, the estimated power consumption on FPGA was compared for the design alternatives with an accelerated clock frequency (135 MHz) and the original designs with a slow clock frequency (27 MHz).

Chapter 6 is dedicated to conclusions and to the possible future development directions. Through the evaluation with several applications, FPGA-based implementation proves that it can overcome the limitation of normal computer architectures when it comes to the computational performance and energy consumption. When the hardware random sampling is applied to the particle filter, the throughput of higher than 60 FPS was achieved with 0.898 W power consumption, which is 3.42 times (135 MHz) and 17 times (27 MHz) faster than the corresponding software version of the algorithm. As regards the tradeoff analysis of circuit sizes, speed performance and estimation accuracy, real-time robust estimation with random sampling performs the object fitting on an FPGA platform with 281.6 MHz. Accordingly, making best use of a deep-pipelined stream processing approach can outperform data processing tasks.

Acknowledgements

I would like to express my heartfelt thanks first of all for JICA who kindly let me win the scholarship to Nagasaki University for a doctorate in computer and information system engineering. Besides, I am deeply grateful to the Faculty Council of Nagasaki University for enabling me to do this illustrious course. My supervisor Associate Professor Yuichiro Shibata really merits a special mention for his unfailing and painstaking supervision throughout my academic research. Without his precious support, it would not be possible to conduct this research.

My thanks are also due to Professor Kiyoshi Oguri, my co-supervisor, who has never failed to give me a number of clear and valuable instructions whenever the situations call for. My sincere thanks are expressed to Professor Senya Kiyasu, Associate Professor Makoto Fujimura, Associate Professor Masao Moriyama, Associate Professor Tomoya Sakai and Professor Yasuhiro Shimizu for sparking my interest in programming languages. Furthermore, I would like to give special thanks to my dissertation committee. Personally, I would like to thank my family and friends for supporting me throughout the years, practically and with moral support, especially my parents. To my husband, you should know that your continued support and encouragement was worth more than I can express on paper.

Particular thanks goes to Keisuke Dohi, Jimpei Hamamura, Yoshiki Hayashida, Masahito Oishi and Fumihiko Iwasaki who responded to my questions and queries so promptly. Completing this work would have been all the more difficult were it not for the support and friendship provided by the other members of slab. They include the following individuals: Akane Tahara, Rie Soejima, Yudai Shirakura, Ryo Fujita, Aiko Iwasaki, Kota Aoki, Kota Fukumoto, Shun Kashiwagi, Koji Okina, Kaoru Hamasaki, Taisei Segawa, Alperen Mustafa Colak, Taito Manabe, Hiroki Nagayama, Kazuya Uetsuhara, Ryo Kamasaka, Shohei Yano, Yuka Mizoguchi, Hiroki Egawa, Kouya Iwamura, Ryouhei Tsugami, Yuichi Kawamata, Takanori Itagawa and Tomohiro Kida. My time at Nagasaki was made enjoyable in large part due to many friends.

Last but not least, I owe my gratitude a great deal to each and every one of the teachers who have helped me move several rungs up my career ladder until I have reached my current status.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When it comes to the performance and computing speed, parallel processing through multi-core processors has been deeply involved in many computer-intensive devices including mobile devices for decades. Historically, the strategy of scaling down the gate size of Integrated Circuits (ICs), reducing the supply voltage and increasing the clock rate, was successful and resulted in faster single-core processors during the 1990s and into the 21^{st} century [1]. However, single-core processor frequency scaling happened to hit the *power wall* in 2004. In terms of increasing the operating frequency, the gap between processor performance and main memory latency is large and growing. This trend is referred to as the *processor-memory gap* or *memory wall*. Multiple instructions are executed simultaneously using multiple instructions in a single clock (Superscalar design) or pipelined manner to reduce the latency of accessing memory. The available instruction-level parallelism has reached the limit, known as *Instruction-Level Parallelism* (ILP) *wall*. The interface between hardware (physical parts of a computer system) and software (a set of instructions on a microprocessor) becomes a promising solution to overcome the above constraints. Hence, with the help of software tools like Electronic Design Automation (EDA), the design of Very Large Scale Integration (VLSI) circuits can improve the design quality and save the design time.

VLSI circuit can be divided into Application Specific Integrated Circuits (ASICs) and Programmable Logic Devices (PLDs). ASIC is designed for specific applications. Although ASIC has many advantages including high-performance and low-power consumption, it needs speeding up time-to-market, reducing development cost and more design flexibility. On the other hand, PLDs can be reprogrammed multiple times. The first Programmable Logic Devices (PLDs) were introduced in the 1970s and which is a general-purpose chip for implementing logic circuits [2]. It looks like a black box that contains logic gates and programmable switches. The types of PLD can be classified into Simple Programmable Logic Devices (SPLDs), Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays

(FPGAs). They are configurable through the software.

SPLD is comprised of many macrocells. A macrocell consists of sum-of-products (SOP) combinational logic expressions and an optional flip-flops [3]. Various SPLD devices are Programmable Logic Array (PLA), Programmable Array Logic (PAL) and Generic Array Logic (GAL). Although both AND and OR planes are programmable in PLA, it is difficult for correctly fabrication, high speed and high performance, that led to the development of a similar device, known as PAL. A PAL consists of a programmable array of AND gates that connects to a fixed array of OR gates and is implemented with fuse process technology. PAL is one-time programmable (OTP) and a GAL is a type of PAL which is reprogrammable, such as EEPROM (E2CMOS), instead of fuses.

A more sophisticated type of chip, called a complex programmable logic device (CPLD), is widely used for larger circuit design with multiple I/O blocks due to the limitations of SPLD with a small number of I/O pins. CPLD is a programmable device which has macrocell, a set of interconnection wires, functional blocks and its own chip memory. The interconnection wiring contains programmable switches that are used to connect the PAL-like blocks. Once a CPLD is programmed, it retains the programmed state permanently, even when the power supply for the chip is turned off. This property is called non-volatile programming technology such as EPROM, EEPROM, flash, etc. With the advancement of technology, it has become possible to produce devices with more flexible architecture than CPLD has. FPGA is a RAM-based digital logic chip whereas CPLD is ROM-based design and it has less resources than FPGA. Since FPGA is the SRAM-based device, it can infinitely reprogrammable. Consequently, development costs are greatly reduced compared to ASIC.

Flexibility of FPGAs has enabled a new computing paradigm called reconfigurable computing, where each application algorithm is directly mapped on FPGAs to be executed as custom hardware. Especially, it has been known that deep pipelined structures configured on FPGAs are advantageous in real-time image processing with regular arithmetic flows. It is also known that for more advanced image processing applications such as object fitting and object tracking, random sampling techniques are effective. However, the random sampling approaches require large amount of computation. so that they are difficult for real-time applications. Although FPGA implementation is promising approach, stream-based pipeline architectures are not straightforward, since random sampling algorithms generally have irregular structure of computation.

With the random sampling techniques for FPGA implementation, object fitting with RANSAC algorithm and object tracking with particle filter algorithm are proposed for this dissertation in terms of power, speed, area and cost. RANSAC is an iterative algorithm to estimate parameters of a model from a data set of points including outliers. Particle filter is a recursive filter that can apply for the object movement estimation with the help of particle movements and it can approximate to the non-linear and non-Gaussian posterior.

Nowadays, Computer Aided Design (CAD) tools are widely used in the rapid development of FPGA technology. With the aim of performance improvement and reducing the hardware resources, architectural redesign becomes a critical issue. However, it is difficult to design the larger and complex circuits targeting FPGA devices for this purpose. To cope with this problem, HDL plays a critical role in FPGA design entry phase with several advantages. Two leading HDLs under Institute of Electrical and Electronics Engineers (IEEE) standards are VHDL and Verilog to describe, simulate and create the hardware design. VHDL stands for Very High speed integrated circuit Description Languages. VHDL was launched in 1981 by the U.S. Department of Defense [4] and became an IEEE standard (1076) in 1987. Since that time, it has been revised many times until the current standard IEEE 1076-2008 called VHDL-2008.

The second type of HDLs is Verilog HDL and Verilog means Verifying Logic. The original version of Verilog was introduced in 1985 by Gateway Design Automation, which was later acquired by Cadence Design Systems. Verilog was adopted as an official IEEE standard, called 1364-1995, in 1995. With a number of features, two subsequent versions of Verilog are Verilog-2001 with major enhancements and Verilog-2005 with minor changes. Verilog is easier to learn than VHDL because of its syntax based on C programming language whereas VHDL syntax is based on Ada programming language.

Describing the hardware design by HDL requires simulation and synthesis. Hardware simulation helps the design verification. Synthesis tool is generally similar to a compiler and is used to translate the hardware design written in HDL into an efficient netlist. Synthesis provides to translate HDL design into gate-level netlist and optimize the design constraints to improve speed and area. Hence, Register-Transfer Level (RTL) design is proposed for this study to ensure the design logical correctness. Xilinx ISE and Vivado are applied for synthesis and analysis of HDL designs, and performing timing analysis in this study.

1.1 Aims of This Study

This study is aimed at:

- **To reveal how the arithmetic processes and matrix sizes impact the performance and hardware costs on FPGA for real-time robust model fitting with RANSAC algorithm.**

Three arithmetic types: long integer, single precision FP and double precision FP with two methods are proposed. Three matrix sizes come from the circle, ellipse, and ellipse with angles. Two types of algorithms are implemented on FPGA with the comparison criteria in terms of the estimation accuracy, performance and resource utilization.

- **To overcome the constraint of parallel resampling on FPGAs for real-time object tracking with non-linear and non-Gaussian system.**

  To cope with this problem, the dissertation proposes an architecture which combines FPGA optimized resampling (FO-resampling) method and an object tracking pipeline for FPGAs.

- **To provide the comparative discussion on implementation alternatives with improved area efficiency in terms of the hardware amount and power consumption.**

  Four alternative designs are introduced, aiming at reducing the hardware amount. With an accelerated clock frequency and a slow clock frequency, three types of power consumption are compared for each design using Xilinx Vivado tool.

- **To select a reasonable number of particles for FO-resampling method.**

  The preliminary evaluation is performed to evaluate the required number of real and virtual particles for resampling step in software before implementing on FPGA. The tradeoff between the accuracy and resource utilization is proposed to achieve the high performance. When it comes to the estimation accuracy, three metrics: Tracker Detection Rate (TDR), Average Tracking Error (ATE) and Maximum Tracking Error (MTE) are evaluated.

- **To find the better accuracy between the multinomial resampling and FO-resampling method for object tracking approach.**

  Tracking quality of a real-time object tracking system based on the proposed architecture is evaluated and compared with a conventional multinomial resampling approach using an object tracking benchmark video.

1.2 Configuration of the Dissertation

This dissertation is structured as follows. Chapter 2 discusses the background of the study including the reconfigurable computing systems in which FPGAs. Chapter 3 gives an overview of the research methodologies. Moreover, the challenges and purposes of this study are also presented. The implementation of solving equations in detail with random sampling techniques using an FPGA platform is described in Chapter 4. Chapter 5 shows how to implement the posterior system state estimation in dynamic state-space model using random sampling. Finally, some conclusions, discussions about major achievements and the scope of the future work are described in Chapter 6.

Chapter 2

Background

This chapter covers the fundamental concepts of FPGAs and different aspects related to FPGAs. The overview of FPGA architecture is described in Section 2.1. The FPGA design flow is presented in Section 2.2. The FPGA CAD tools makes the design quality improved in terms of area-efficiency, power consumption and speed. Section 2.3 discusses the relation to previous works on FPGA-based real-time robust model estimations with RANSAC algorithm. Section 2.4 surveys related work on the efficient implementation of particle filter using random sampling techniques on FPGAs.

2.1 Field Programmable Gate Arrays

A brief timeline of the steps leading to FPGA development is Metal Oxide Semiconductor Field Effect Transistors (MOSFETs) in 1960, Integrated Circuit (IC) in 1961, Transistor-Transistor Logic (TTL) in 1962, Complementary Metal Oxide Semiconductor (CMOS) in 1963, Moore's law in 1965, Programmable Read-Only Memory (PROM) in 1970, Erasable Programmable Read-Only Memory (EPROM) in 1971, Depleted Substrate Transistor (DST) in 1972, Programmable Logic Array (PLA) in 1975, Programmable Array Logic (PAL) in 1978, Electrically Erasable Programmable Read-Only Memory (EEPROM) in 1983, Generic Array Logic in 1983 and flash memory in 1984 [5]. Finally, the first commercial FPGA with 2000 gates, named XC2064, was debuted by Xilinx in 1985.

The main architectural components of FPGA are the number of configurable logic blocks, embedded block RAM, programmable interconnections and other hard macros inside the core area as shown in Fig. 2.1. A configurable logic block is made up of slices which contain a group of logic cells. A logic cell consists of look-up-tables (LUTs), flip-flops (FFs), a network of carry logic and multiplexers. A LUT can implement any Boolean function as a truth table, consisting of a number of memory cells equal to 2^n SRAM bits inputs for n-number of outputs. Fig. 2.2 shows one of

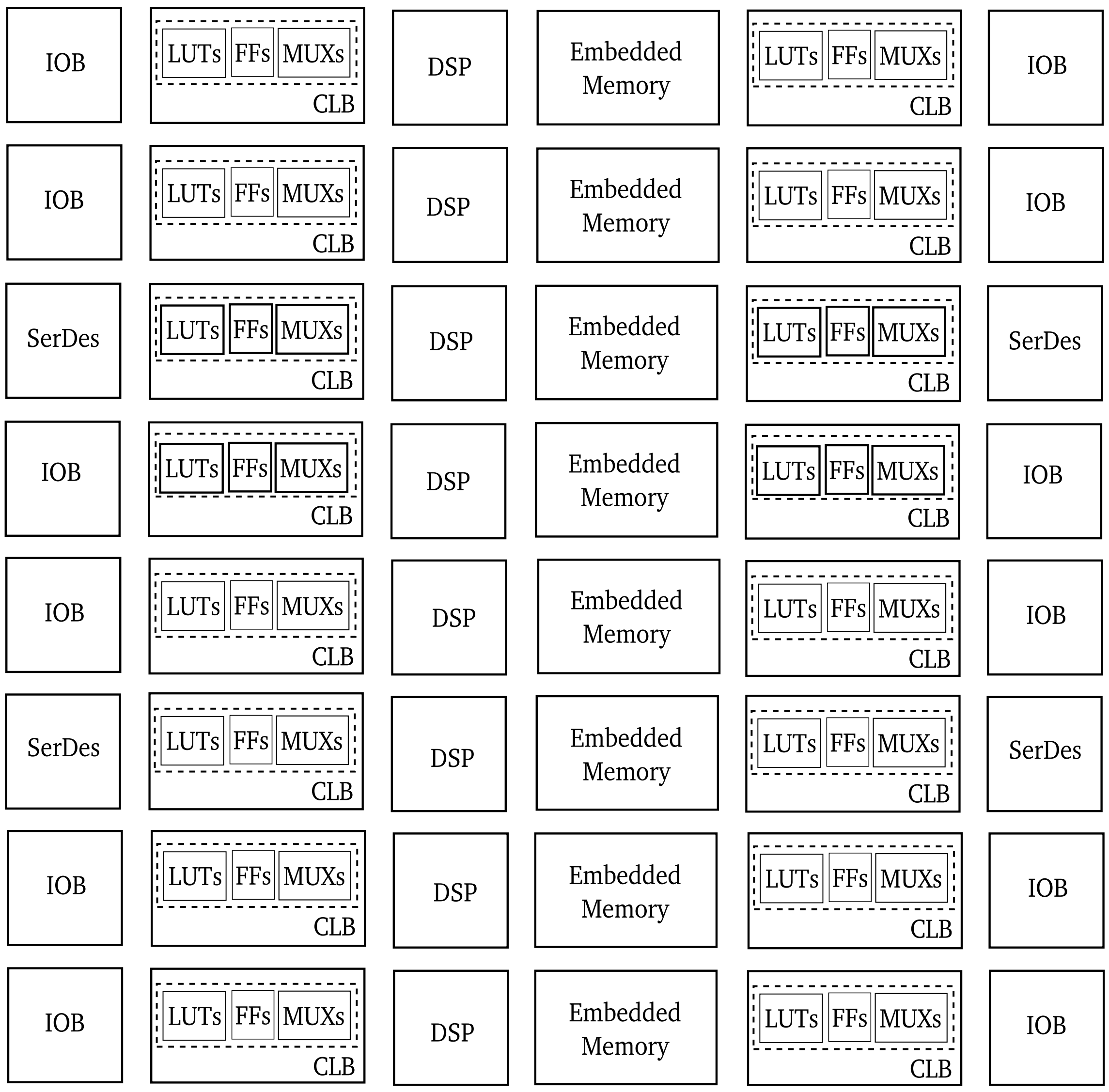


Figure 2.1: The basic architecture of an FPGA

the basic example of LUT function as a combinational circuit. After performing the logic operations, the result of LUT is stored in the register elements called Flip-Flop (FF) which is a very fast memory for sequential logic operations. LUTs provide FPGA architecture with great flexibility and utility [6]. The programmable wire plays a role to connect the elements in FPGA depending on the arbitrary circuit. The input and output block (IOB) supports a programmable unidirectional or bidirectional interface between the core of the FPGA and external devices.

Clocking resources provide clock deskewing, frequency synthesis and phase shift. The main types of FPGA clocking resources are clock buffers, dedicated clock routing and clock management tile (CMT) [7]. Xilinx provides several clock buffers including global clock buffer, Input/Output (I/O) clock buffer, regional clock buffer, single-ended input buffer and differential input buffer. One of the single-ended input clock buffer, IBUFG, provides dedicated connections from a top level port to MMCM,

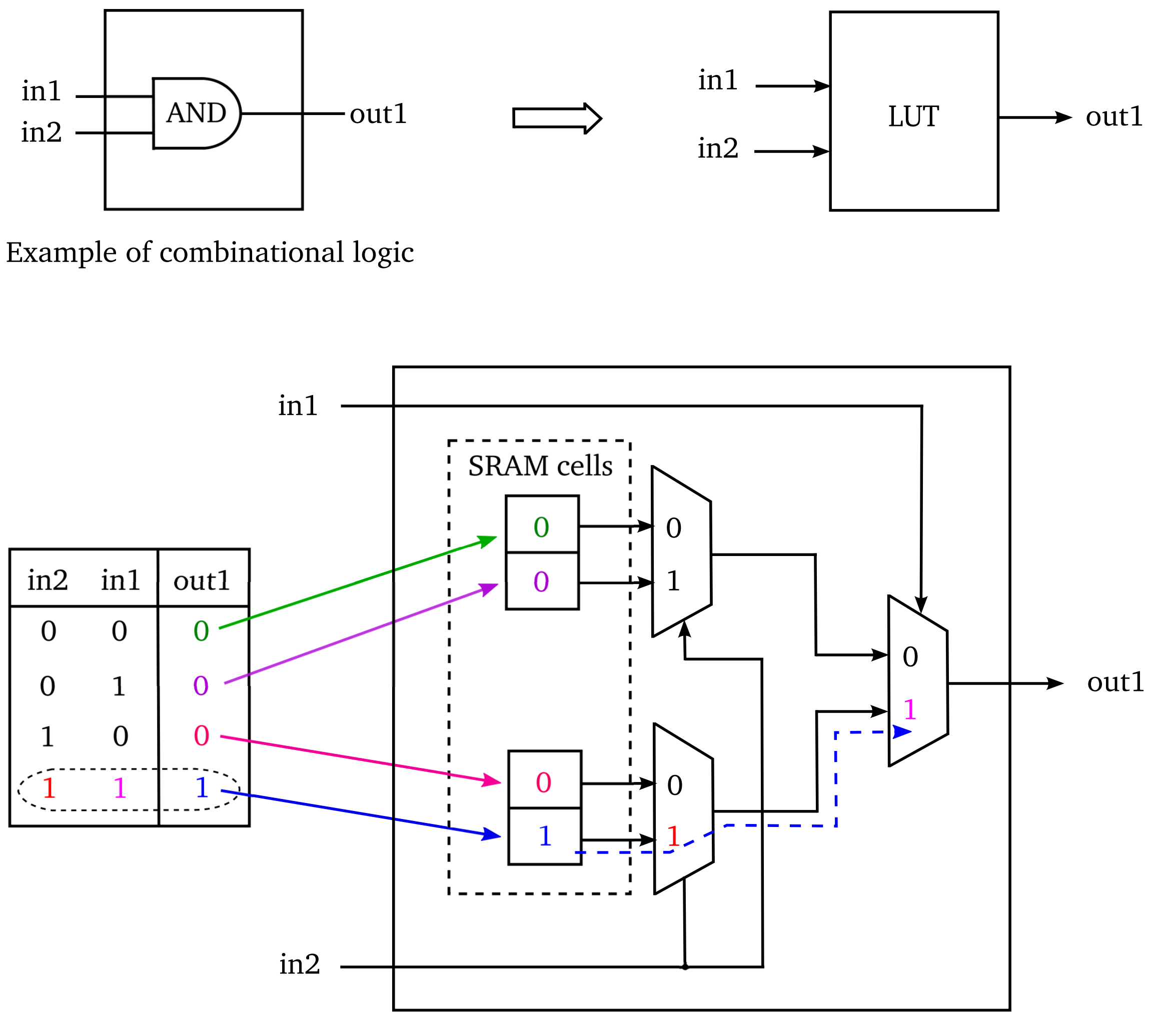


Figure 2.2: Example of two-input LUT implementation

PLL, BUFG, etc [8]. Global buffers, BUFGs, can minimize the clock skew and give precise phase. Some of differential signaling dedicated clocking resources are IBUFDS, OBUFDS and IBUFGDS. Differential clocks give better noise immunity and they are suitable for high frequencies which are above 100 MHz. Moreover, dedicated clock routing can reduce the delay time compared with the general clock routing.

CMT consists of Digital Clock Manager (DCM), Phase-Locked Loop (PLL) and Mixed-Mode Clock Manager (MMCM). DCM primitive includes a Delay-Locked Loop (DLL) to create a customized clock. Only DCM is mainly used in Spartan-3 and earlier Xilinx FPGA families. The basic idea of PLL is to reduce the clock jitter and to generate multiple different output clock frequencies simultaneously using a Voltage Controlled Oscillator (VCO). PLL is an analog clock manager. PLL together with DCM was introduced in Virtex-5 and Spartan-6. MMCM leverages features from PLL with phase control. MMCM and PLL primitivies in 7 series

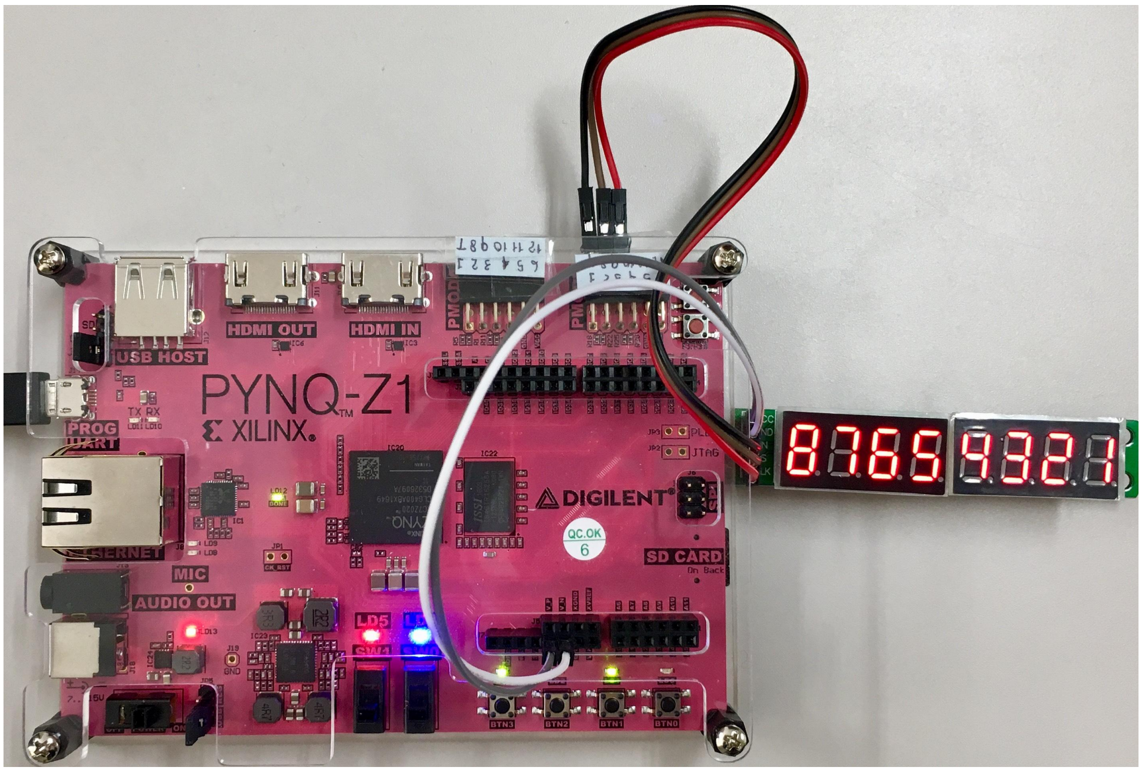


Figure 2.3: Example of seven-segment display on Pynq-Z1 FPGA

FPGAs are MMCME2_BASE, MMCME2_ADV, PLLE2_BASE and PLLE2_ADV. FPGA provides Clocking Wizard and manual instantiation of clock manager to create output clock frequencies [9]. The desired output frequency varies with input frequency (F_{CLKIN}) as shown in Eq. (2.1) and Eq. (2.2). M, D and O represent CLKFBOUT_MULT_F setting which is fraction counter, DIVCLK_DIVIDE and CLKOUT_DIVIDE. The number of "O" counters can be independently programmed ranging from 1 to 128 with six for PLL (O_0 to O_5) or seven for MMCM (O_0 to O_6).

$$F_{VCO} = F_{CLKIN} \times \frac{M}{D} \tag{2.1}$$

$$F_{OUT} = F_{CLKIN} \times \frac{M}{D \times O} \tag{2.2}$$

For example, arduino MAX7219 seven-segment digital LED tube display is connected to Pynq-Z1 board which owns Artix-7 family programmable logic [10]. In that case, serial clock input in MAX7219 limits 10 MHz maximum rate [11] and Pynq-Z1 system clock is 125 MHz. Hence, generating serial clock from 125 MHz becomes a necessary step. When it comes to the avoiding dependency on CAD tools, MMCME2_BASE manual instantiation with global clock network and no hold time violation was chosen to implement the seven-segment display as shown in Fig. 2.3.

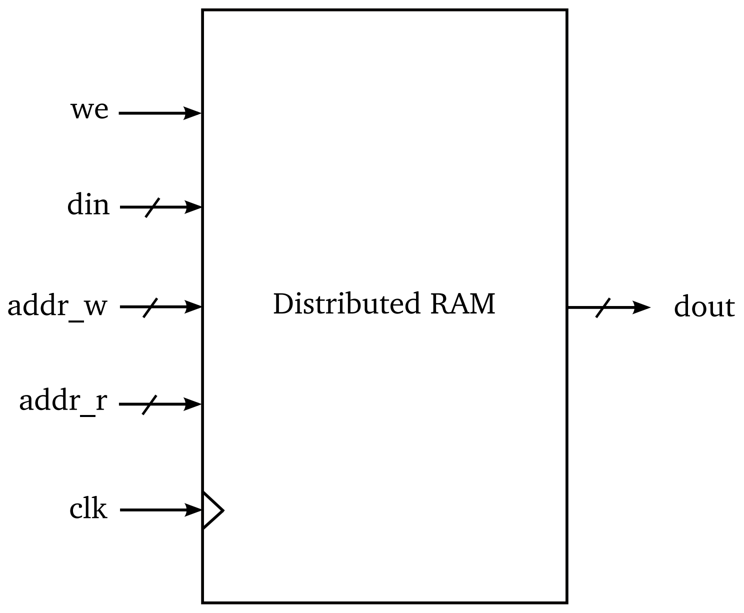


Figure 2.4: Single-port distributed RAM

The serial clock output from FPGA is given by Eq. (2.3).

$$\begin{aligned} F_{MAX7219\_OUT} &= 125 \times \frac{8}{1 \times 100} \\ &= 10MHz \end{aligned} \tag{2.3}$$

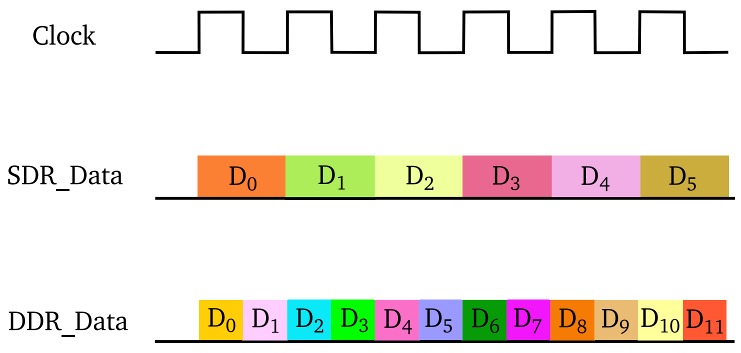
Xilinx also provides four options for using the design element [12]. The four options are (1) instantiation, (2) inference, (3) CORE Generator or other Wizards and (4) macro support. Memory is one of the dominant factors in high performance data computing platforms. The main function of memory in FPGA is to store and retrieve the data with embedded memories: a dedicated Block RAM (BRAM) or LUT-based distributed RAM [13]. Depending on the circuit designs, data can access via single-port RAM, dual-port RAM, single-port ROM, dual-port ROM, etc., using a special memory coefficient file (.coe) or manual instantiation. BRAM can implement for large sized memories whereas cascaded several distributed RAM are required for deeper and wider memory implementation. Hence, distributed RAM is more suitable for small sized memories to avoid extra wiring delays.

In single-port distributed RAM given by Fig. 2.4, if write enable control signal (we) is asserted at the rising edge of clock (clk), a write operation is performed with write address (addr_w) and input data (din) is stored into memory simultaneously. The read address (addr_r) can retrieve the output data (dout) using asynchronous or synchronous method. Aspect ratio of memory varies with data width and address depth. Moreover, shift registers can apply for time delay using data storage and data movement. First-In First-Out (FIFO) is also popular for data queue and interfacing two systems of differing data rates.

FPGA can interface with external devices via I/O modules such as Universal Serial Bus (USB) port, Video Graphic Array (VGA) port, High-Definition Multimedia Interface (HDMI), RS-232 port, Universal Asynchronous Receiver/Transmitter (UART), connectors, etc. Moreover, Xilinx has adopted the Advanced eXtensible Interface (AXI) protocol from ARM Advanced Microcontroller Bus Architecture (AMBA) for Intellectual Property (IP) cores [14]. AXI design begins with the Spartan-6 and Virtex-6 devices and continues with 7 series and Zynq-7000 devices. Xilinx introduced three types of AXI4 interfaces: AXI4, AXI4-Lite and AXI4-Stream in ISE and Vivado design suites. AXI4 is for high-performance memory-mapped requirements and AXI4-Stream is for high-speed streaming data. AXI4-Lite can apply for simple, low-throughput memory-mapped communication. On the other hand, Altera provides Avalon bus protocols with two variants: Avalon Memory Mapper (Avalon-MM) and Avalon Streaming (Avalon-ST) in the classic tool SOPC Builder and the new tool Qsys [15]. Alternative on-chip internal buses are CoreConnect from IBM which is used by Xilinx Embedded Development Kit (EDK), Open Core Protocol (OCP) from OCP International Partnership (OCP-IP) and Wishbone from OpenCores.

Basically, general-purpose I/O interfaces can be divided into two types, namely: single-ended interfaces and differential interfaces [16]. A signal's assertion (whether it is High or Low) of single-ended interface is based on its voltage level relative to a fixed voltage threshold that is referenced to GND. Depending on the voltage of the signal and its threshold voltage, the state is considered High or Low. With the higher-performance interfaces and power saving, differential interfaces assert a signal based on the relative voltage levels of the two complementary signals. The state is considered High or Low according to the voltage difference between Positive signal and Negative signal. Common example of a single-ended I/O standard is Transistor-Transistor Logic (TTL) and Low-Voltage Differential Signaling (LVDS) for a differential I/O standard. Supported I/O standards in 7 series are LVTTL, LVCMOS, TMDS, LVDS, etc.

Single Data Rate (SDR) and Double Data Rate (DDR) are available for both single-ended and differential interfaces. In essence, SDR can transfer one data per one clock cycle whereas DDR allows two data per one clock cycle as shown in Fig. 2.5. Hence, data bandwidth in DDR becomes two times of clock frequency. Xilinx FPGAs, such as Spartan-6 and 7 series, contain Input SerDes (ISERDES) and Output SerDes (OSERDES) blocks. These primitives make the design of serializer and deserializer circuits very straightforward with higher operational speeds [17]. To serialize and deserialize the data in the correct bit order, a Bitslip submodule is an essential for reordering the parallel data in ISERDESE2 block and the serial data in OS-



Clock
SDR_Data
D_0 D_1 D_2 D_3 D_4 D_5
DDR_Data
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8 D_9 D_{10} D_{11}

Figure 2.5: Sample timing diagram for SDR and DDR

ERDESE2 block [18]. The Bitslip operation can shift the data left by one bit in SDR mode. Moreover, it can shift the output pattern right by one and left by three alternatively in DDR mode.

Xilinx provides example verilog programs for HDMI and Digital Visual Interface (DVI) to implement Transition Minimized Differential Signaling (TMDS) audio/video interface using the master and slave ISERDES2 and OSERDES2 modules [19]. The DVI/HDMI transmitter design includes TMDS encoding and 10-bit parallel-to-serial conversion. Similarly, the receiver design involves TMDS decoding and 1:10 deserialization. The reference of TMDS decoding and encoding designs is given by [20]. To perform the required 10:1 serialization or 1:10 deserialization for both DVI and HDMI, ISERDES2 cascading and OSERDES2 cascading are implemented on FPGA for DVI/HDMI pass-through design. In particular, visible area, the whole frame area, pixel frequency and screen refresh rate can be different depending on modes such as VGA mode 640×480 @ 60 Hz for industry standard timing or HDMI mode 1920×1080 @ 60 Hz. Video camera interfacing with FPGA is also widely used in real-time image processing. For example, CMOS OV7670 camera provides 640×480 VGA resolution with a $25-$MHz pixel rate and Serial Camera Control Bus (SCCB) interface compatible with I2C interface [21]. Real-time video data coming from camera to FPGA is compatible with SubMiniature version A (SMA) connectors or Pmods. The interpolation from bayer pattern to true color (red, green and blue) or some image quality control is done by Digital Signal Processor (DSP). It is efficient for handling pixel values on FPGA due to dedicated DSP slice resources and fully hardware parallelism. To sum up, FPGA becomes one of the solution for real world applications which offer low power and high performace.

2.2 FPGA CAD Flow

The design flow of FPGA consists of many steps from design specification to implementing a digital logic design on FPGA chip as shown in Fig. 2.6. The first step of the design process is the design specification. Software simulation is an example of design specification. Then, the circuit design must be entered into the design application software called design entry. FPGA design entry can be divided into two methods: (1) a schematic editor and (2) an HDL. The next step is functional simulation to simulate the entered and compiled design by software for logical correctness without timing information. Waveform editor and HDL testbench are main tools for this purpose. Some simulators for FPGA design simulation are ISIM and Vivado by Xilinx, ModelSim by Mentor Graphics, VCS by Synopsys, NCSim by Cadence, Qsim by Altera, Icarus Verilog and Verilator, etc.

Moreover, hardware simulation can be divided into three stages: analysis, elaboration and simulation. *Analysis* is the first step of simulation and checks for the syntax and semantics errors of HDL. *Elaboration* is the process of expanding the HDL description to instantiate all modules in Verilog or entities in VHDL into a hierarchically described circuit for simulation. Finally, event-driven simulation is carried out to generate the timing information or a cycle-based simulation is used to reduce the number of calculation for functional verification.

Synthesis is an essential step for synthesizing the design. It generates a netlist file which connects a list of nets such as logic gates, flip-flops, wire, etc. Design implementation is an important step with three procedural steps: (1) translation, (2) mapping and (3) place and routing step. All netlists from synthesis step are translated into one large flat netlist with no hierarchy. Then, logic blocks from design netlist file are mapped into FPGA resources depending on the target FPGA device. Finally, predefined FPGA resources are placed onto the locations of physical device and are routed the interconnect according to the predefined path in netlist. Place and route design offers the area and speed factor of FPGA device. After implementing process, configuration file is generated called a bistream, which is a device dependent and it can be directly downloaded onto FPGA chip via JTAG with iMPACT software or other configuration ports.

2.2.1 Schematic capture

A schematic capture tool is a CAD tool to draw a logic circuit using logic gates and interconnection wires [2]. A *library*, the collection of graphical symbols, facilitates many logic gates with different numbers of input pins to draw a schematic diagram. Moreover, *hierarchical design* provides to create a large circuit with other subcircuits. Although the schematic-capture method is simple for small circuits, it is not suitable for the larger and more complex circuit and can be cumbersome. Hence, a better method for dealing with larger circuits becomes HDL because it is easy to write source code.

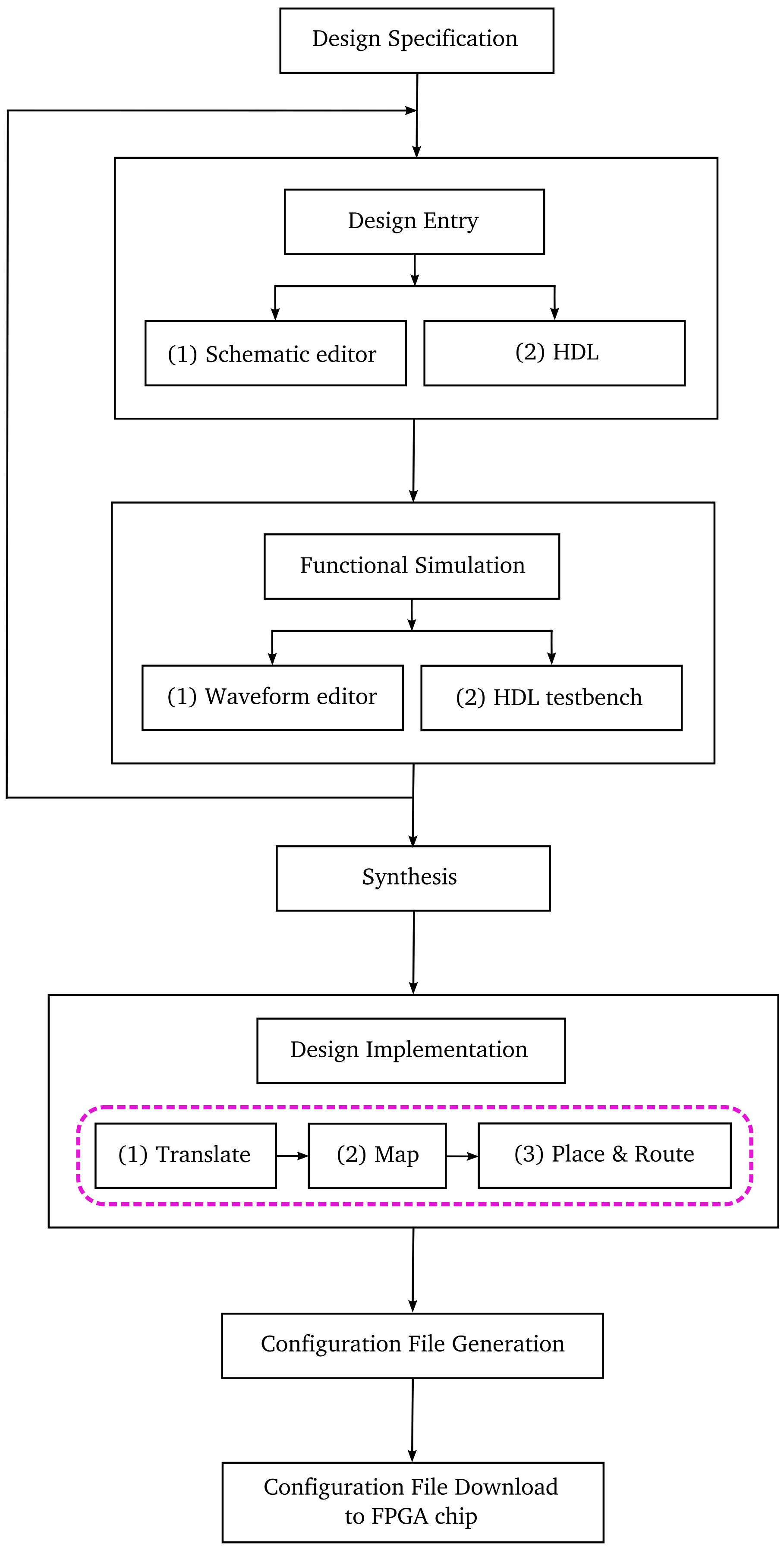


Figure 2.6: Design flow of FPGA

2.2.2 Hardware Description Language

A Hardware Description Language (HDL) describes hardware to express an electronic circuit. More specifically, it is a form of computer language rather than a program to be executed on a computer. The main difference between hardware description and software is that HDL offers concurrent processing whereas software for microprocessor performs the sequential processing. Although many commercial HDLs including proprietary are available from many vendors, standard HDLs supported by IEEE are VHDL, Verilog HDL, SystemC and SystemVerilog.

2.3 FPGA-based Object Fitting using RANSAC

Effective parameter estimation has been deeply involved in computer vision for a long time. One of the most popular robust estimator for fitting a model to experimental data is RANSAC [22]. Moreover, FPGAs have recently become the focus of considerable interest in embedded imaging application in terms of the inherent parallelism results in better performance [23–25]. Many RANSAC-based systems can employ various computational approaches using FPGA. For example, Dellaert and Tariq [26] introduced a multi-camera pose tracker using a RANSAC-based method to estimate the true pose of the rig by finding 2D to 3D point correspondences between the images captured from the rig and survey features in the environment. They developed an FPGA-based miniature camera rig to detect affine invariant features in real time for up to 4 cameras in parallel. Many feature extraction methods required by RANSAC systems [27] such as Harris [28], SURF [29–31], SIFT [32–34] and BRIEF [35] have also shown to be effective on FPGAs.

As regards real-time robust estimation, RANSAC performs the road model fitting of a Lane Departure Warning (LDW) system on an FPGA platform [36]. In that case, the lane marking candidates from the input images are extracted using Gaussian noise reduction, histogram stretching, intensity gradients, edge thinning and extraction method. More specifically, the three main modules of the system: (1) extracting, (2) fitting and (3) tracking runs in real-time at 30 FPS with a resolution of 752×480. [36] reported that the FPGA-based system was about 30 times faster than the corresponding software version of the algorithm. Moreover, FPGA implemention of the robust essential matrix estimation with RANSAC can accelerate the processing speed in [37].

Fijany and Hosseini [38] have studied model estimation of a homography transformation using RANSAC on FPGA for Harris Corner Detector [39] and Sum of Squared Difference computations. Experimental results given in [40] describes a high-efficiency FPGA-based pipeline and a fast outlier rejection scheme using

RANSAC for an inertial-assisted visual odometry system. The cost-time-accuracy trade-offs analysis of five distinct HW/SW pipelines on a Virtex-6 FPGA and a 150 MIPS CPU are implemented for visual odometry [41]. In ellipse estimation for eye tracking [42], Starburst algorithm was performed to get the optimal hypothesis result based on RANSAC.

Chapter 4 presents the comparative evaluation of implementation alternatives for a real-time FPGA-based model estimation system based on RANSAC. Although the proposed system can apply for various geometric shapes, three different models are selected to easily evaluate the matrix size and performance. They are circle, ellipse and ellipse with angle. The robust estimation systems using various matrix sizes with different arithmetic approaches are implemented on an FPGA to compare the estimation accuracy, performance and resource usage.

2.4 FPGA-based Object Tracking using Random Sampling

Since the object tracking plays a part in image processing and computer vision systems, fast and robust real-time image processing of non-rigid objects becomes a performance bottleneck. On the other hand, Kalman filter is a popular and powerful tool in order to estimate the state of a system for numerous applications such as object tracking, autonomous navigation and many computer vision applications. However, it can only work for linear state transitions. Kalman filter uses the Gaussian (normal) assumptions given by Fig. 2.7 to keep track of means and variances. In the real world, many tracking problems deal with non-linear and non-Gaussian systems. The main advantage of particle filter is that can apply for non-Gaussian models as shown in Fig. 2.8. For this reason, image-based features for particle filters using color histogram were introduced by [43]. Particle filtering [44] approximates the density directly as a finite number of samples whereas the extended Kalman filter cannot approximate the probability density without a Gaussian.

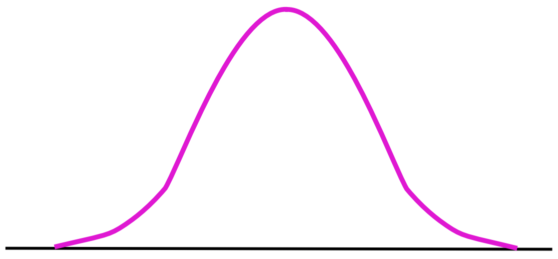


Figure 2.7: An example of Gaussian distribution in Kalman filter

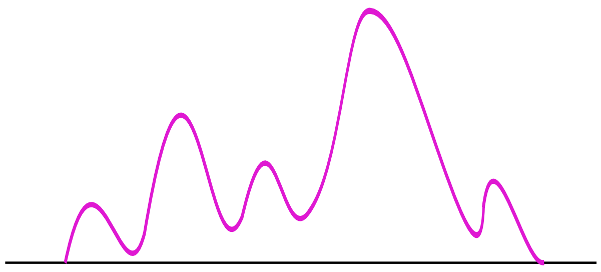


Figure 2.8: An example of non-Gaussian distribution in particle filter

The color-based particle filter enables an embedded implementation in [45]. However, the parallel implementation does not offer in their tracking system. Many parallel resampling methods have been applied to FPGA-based particle filter implementation. For example, the particles are resampled using Independent Metropolis-Hastings (IMH) resampling method and the root mean square error is used to measure the accuracy [46]. Although they could apply the parallel particle filter implementation successfully with the high speed and accurate estimation performance, they did not fully parallelize the particles due to lack of hardware resources.

Difficulty of parallel resampling comes from it needs information of all particles. Hence, FO-resampling method overcomes this constraint on the particle filter [47]. In their implementation, they compared FO-resampling with the multinominal resampling using root mean square error. The FO-resampling scheme is similar to Gibbs sampling but it can execute the resampling step without the complete particle set. In Chapter 5, a deep-pipelined fast and robust object tracking system with stream-based image processing on an FPGA are proposed. Image data obtained by the camera device is streamed in the system pixel by pixel [48]. The streamed processing approach achieves real-time object detection for input video frames without any external memory modules [49–51].

Chapter 3

Research Methodology

3.1 Real-time Image Processing

With the commercialization of camera devices gaining momentum, many real-world image processing applications rely on the real-time image and video processing systems. A pixel, a picture element, is the basic part of a digital image. Normally, a frame is composed of many pixels arranged in a 2-dimensional arrays. Real-time image processing system requires high resolution, low latency and high throughput. The important criteria for high resolution image is the number of pixels in an image. On the other hand, each pixel generally represents the eight bits of memory and color pixel requires 24-bit memory for true color. Therefore, memory management of real-time image processing becomes a performance bottleneck. When it comes to the performance and speed of image processing, two types of real-time image processing with different platforms are presented in Fig. 3.1 and Fig. 3.2, respectively.

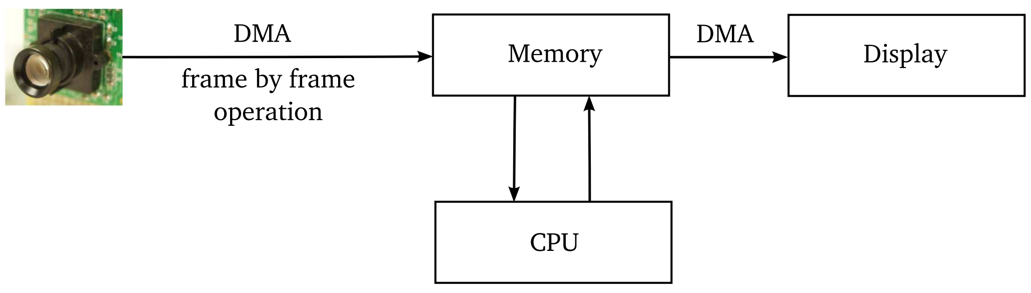


Figure 3.1: A typical system organization for real-time image processing with CPU

Since CPU performs the sequential operations and CPU cannot connect directly from camera, transferring data from camera to CPU via memory leads to high latency and low throughput. *Latency* is the time between the start and the completion of an event also referred to as execution time [52]. In particular, latency is measured in units of times: hours, minutes, seconds, nanoseconds or clock periods [53].

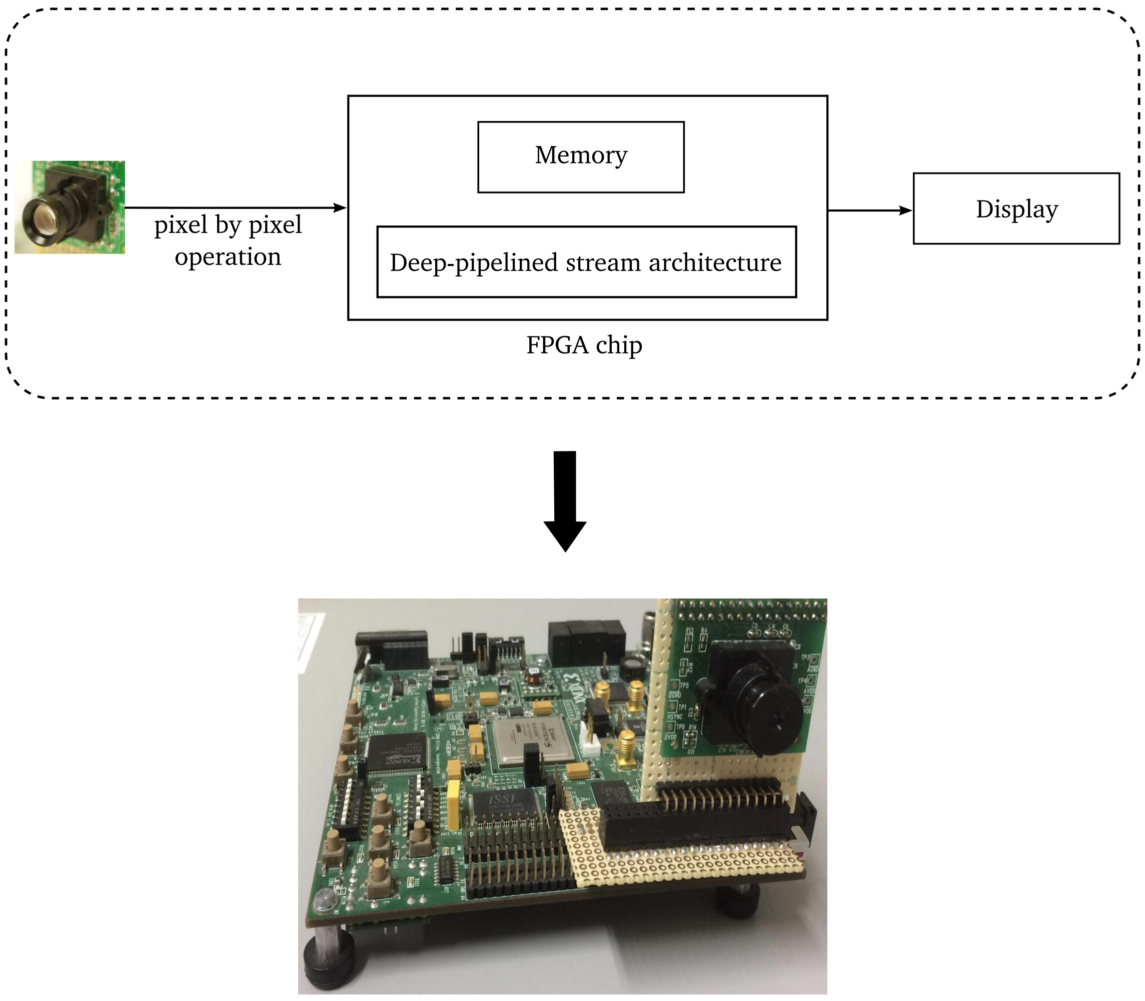


Figure 3.2: A typical system organization for real-time image processing with FPGA

Throughput is the total amount of work done in a given time. This is measured in units of whatever is being produced (car, motorcycles, I/O samples, memory words, iterations) per unit of times.

Direct Memory Access (DMA) can transfer the data faster than the Programmed I/O (PIO) does because it can directly read from and write to main memory without CPU whereas PIO requires CPU. However, CPU cannot execute without memory interfacing with peripheral devices, i.e., high latency and low throughput. In contrast, FPGA can directly connect with camera without external memory, i.e., low latency and high throughput with low power consumption. Hence, FPGA with pixel-by-pixel operation can outperform CPU with frame-by-frame operation on real-time image processing using deep-pipelined stream architecture.

Necessity of high performance in data processing, pipelining technique becomes a critical issue for this purpose. Nowadays, almost every processor utilizes pipelining process because it can make the throughput increased as shown in Fig. 3.3 and Fig. 3.4. Example latency uses $2ns$ to compare the rate at which example models arrived at their intended destination. A deep pipelining not only can enhance the throughput but also can save the energy. Therefore, this study aims to reveal the methodology which allows for real-time image processing using deep-pipelined stream processing on FPGA.

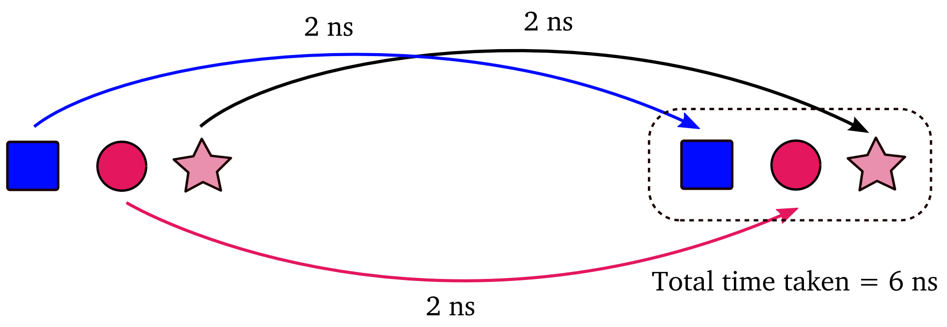


Figure 3.3: Example of non-pipelining

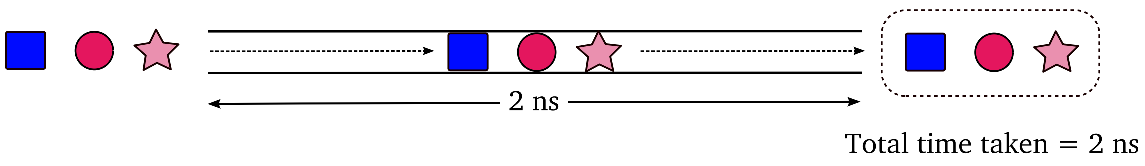


Figure 3.4: Example of pipelining

3.2 Stream-oriented Process

Since the effective memory utilization makes the data processing speeded up, it is suitable for many applications required the high speed computation. The stream-oriented process can be implemented with the streamed architecture as shown in Fig. 3.5. It can effectively apply to FPGA architecture [42,48,54–60]. The streamed architecture mainly consists of three parts: registers, First In, First Out (FIFO) and arithmetic pipelining. The input data are stored in registers as a part of pipeline stages. With the shift registers, FIFO is used to queue the data for specific applications although it is not a part of pipeline. The number of shift registers and FIFO width depend on the input data and applications. Only the pipeline part is used for computation of data processing. With this architecture, a large amount of data stream can process without any huge memory.

Fig. 3.6 shows the input example of stream processing with a grayscale image and its pixel values between 0 to 255. The intensity values of image represent a two-dimensional array matrix. For the image processing, good memory management is a key for the energy efficiency. To understand clearly about the stream processing,

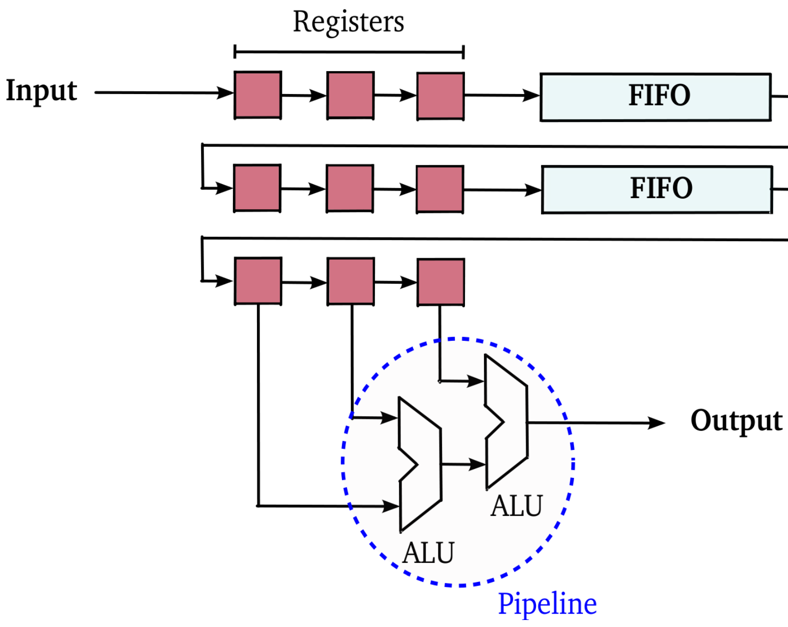


Figure 3.5: Streamed architecture

169	103	129	127	126	160	139	98
146	103	121	147	210	151	147	57
151	99	106	160	176	194	40	159
143	99	122	54	175	42	53	160
137	111	124	79	139	55	154	175
150	39	211	75	109	76	147	207
107	48	116	69	150	134	132	96
90	57	125	89	137	206	48	79

Figure 3.6: Input example of stream processing

let us consider a grayscale image and a Laplacian edge detection image as input and output examples of streamed architecture. The Laplacian kernel is selected as an example of pixelwise image arithmetic given by Fig. 3.7. The image size corresponds to the y number of rows and x number of columns, 8 by 8 elements for this example. Any image filtering process can perform with m-connected neighborhoods instead of convolution operation and (x_0, y_0), (x_0, y_1), (x_0, y_2), (x_1, y_0), (x_1, y_1), (x_1, y_2), (x_2, y_0), (x_2, y_1) and (x_2, y_2). Six shift registers store the pixel values of 3×3 matrix from image and 5-bit two FIFO for holding the pixel values. The data are fed from registers to ALU for doing the arithmetic operations using pipeline architectures. The main purpose of this architecture is for fast and low power FPGA implementation for real-time image processing without using any external memory.

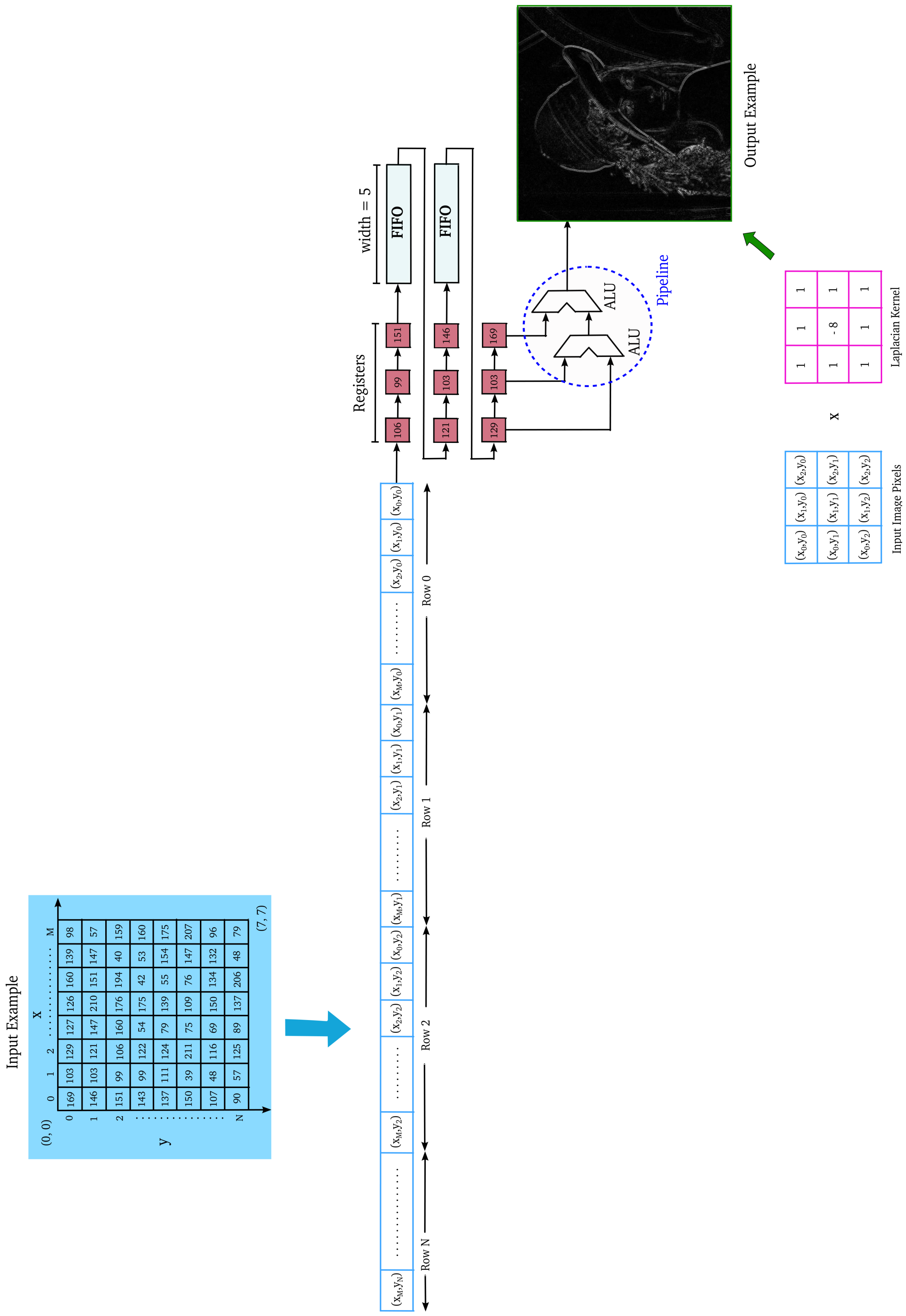


Figure 3.7: Example of image stream processing

3.3 Random Sampling in Image Processing

Since image processing plays a role in various fields such as computer vision, machine vision, artificial intelligence, machine learning, computer graphics, embedded systems, etc., various sampling algorithms are applied for object detection and tracking of moving objects. According to parameter estimation, least square parameter estimation is generally used for point correspondences. However, it is sensitive to outliers, so that a few outliers can greatly skew the model fitting [61]. Hence, estimation methods which are robust to outliers become essential. Generally, robust estimation executes two processes. The first process is to classify data points as outliers or inliers, i.e., points do fit the model or not. After that, the second process will begin to fit the model to inliers while discarding outliers.

Since the robust estimators can reduce the adversely effect of outliers, RANSAC algorithm becomes a powerful tool to cope with the model estimations which contains outliers. Selecting the number of samples and the number of feature points in a sample are initial stage to apply RANSAC algorithm to image processing. Initial estimation with least squares method over all inliers makes the fitting quality improved.

This dissertation addresses the efficient model estimation from input video image using RANSAC algorithm with three different arithmetic types: long integer (64-*bit*), single floating point (32-*bit*) and double floating point (64-*bit*). The basic format of IEEE Standard floating-point representation for binary floating-point numbers is shown in Fig. 3.8. The sign bit represents 1 bit: '0' for positive and '1' for negative. Although the sign bit is same for both floating-point representations, exponent and mantissa are different, i.e., 8 bits and 23 bits for single precision FP format, and 11 bits and 52 bits for double precision FP format, respectively.

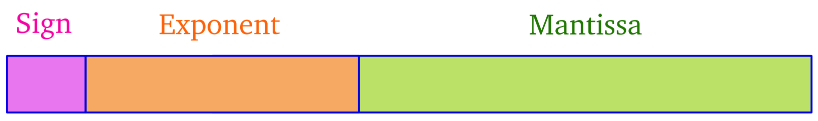


Figure 3.8: IEEE 754-standard floating-point format

According to object fitting, randomly selecting the center points (x, y) and radius (r) are required to estimate the circle model from random generated feature points as shown in Fig. 3.9 and Fig. 3.10. Depending on the minimum sample points of estimated models, the sizes of matrices are also different to solve the simultaneous equations. Fig. 3.11 and Fig. 3.12 illustrate the two types of ellipse models. Chapter 4 presents the detailed explanation of model estimations from feature points with two different algorithms to get a reasonable outcome.

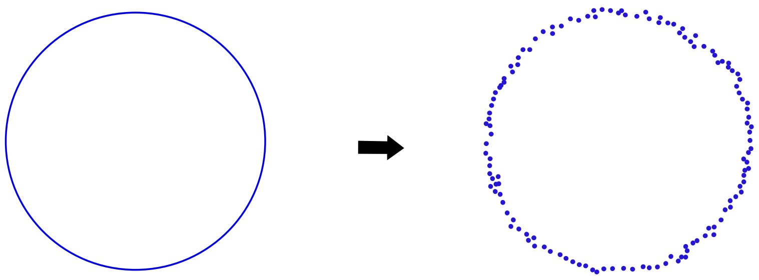


Figure 3.9: Example of circle model and feature points with observation errors

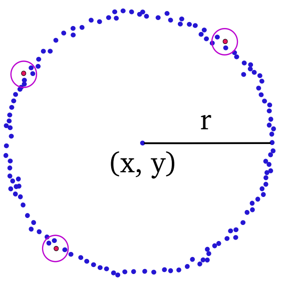


Figure 3.10: Example of random sample points for circle model

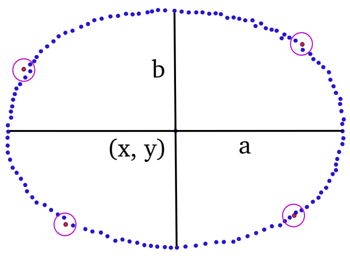


Figure 3.11: Example of random sample points for ellipse model

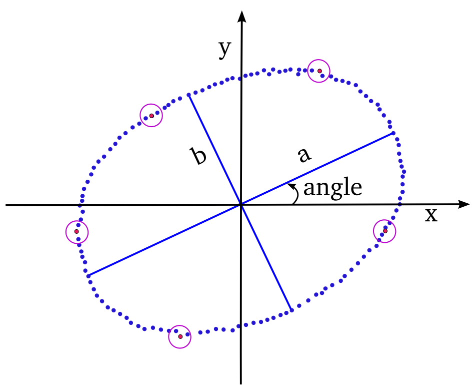


Figure 3.12: Example of random sample points for ellipse model with angle

In the state estimation of non-linear dynamic systems, random number generation and resampling are critical to meet the timing constraints and and its performance target. Parallel resampling becomes a challenging problem in object tracking when it comes to high speed and performance. This dissertation aims to perform the high performance object tracking with parallel resampling compared to multinomial resampling method. Moreover, alternative designs implemented on FPGA are compared with respect to performance and power consumption.

Chapter 4

FPGA-based Real-Time Robust Model Fitting

4.1 Overview

With the rapid development of FPGA technology, FPGA implementation of real-time circle and ellipse estimations from images is a promising solution in many applications such as robotics, computer vision, and so on. More specifically, FPGAs are getting popular in the name of the high energy efficiency, that is, a good energy-performance ratio. To fulfill the demands for robust estimations of objects such as circles and ellipses, a well-known and effective algorithm called RANdom SAmple Consensus (RANSAC) [22] has been widely used. Typical implementation of RANSAC essentially consists of three process steps: (1) randomly selecting a set of points for model parameters from feature points, (2) generating hypothesis from the selected points and (3) verifying the generated hypothesis. These three steps are repeated in order to get the best hypothesis as a result.

The real-time implementation of the RANSAC algorithm needs to generate as many hypotheses as possible for different point selection during a single camera frame, and Step (2) easily becomes a performance bottleneck [42]. The most time consuming task is the least square method to generate hypothesis from the selected feature points where a simultaneous linear equation is solved. The number of unknown variables of the simultaneous equations is different according to each model: 3 to 5 for circles and ellipses. In contrast to large matrix solvers, few attention has been paid so far to small matrix manipulation on FPGA, and it is a question what kind of approach is appropriate for FPGA.

In this chapter, several kinds of FPGA implementation of equation solvers are compared to reveal tradeoff relationship among different methods when it comes to the matrix manipulation on an FPGA. The evaluated solver algorithms are the

Gauss-Jordan method and Cramer's rule method. While the Gauss-Jordan method is one of the most popular algorithms for this purpose, the Cramer's rule is generally never considered due to a large number of multiplication operations to be made in the process of calculation of the determinant [62]. However, especially for a small data set, execution performance of an algorithm becomes more sensitive to architectures and does not necessarily reflect computational complexity.

The advantage of the Cramer's method in FPGA implementation is that it has a simple and regular control flow. In addition, most calculations can be performed with integer arithmetic rather than floating-point arithmetic. On the other hand, a wide dynamic range is required to calculate the determinants of a given matrix. For the sake of revealing how the arithmetic type impacts the performance and hardware costs, two arithmetic implementation for the Cramer's rule, double precision floating-point and long integer, are compared by changing the size of matrices. While several FPGA implementation of circle and ellipses estimations have been reported including in [42] [23], one of the major contributions of this chapter is comprehensive discussion on implementation alternatives including effects of matrix size and the use of long-integer arithmetic in a simultaneous equation solver.

The rest of this chapter is structured as follows. Section 4.2 gives a brief overview of the RANSAC algorithm. The implementation of the system in detail with both floating-point and long integer arithmetic processes are described in Section 4.3. Section 4.4 shows the evaluation results and discussion. Finally, summary of this chapter is described in Section 4.5.

4.2 RANSAC Algorithm

RANSAC is an iterative algorithm which fits the parameterized model from a data set of points including outliers. Initially, RANSAC randomly picks 3 points for a circle, 4 points for an ellipse and 5 points for an ellipse with angle, respectively from feature points. Subsequently, a hypothesis is generated from the selected points according to the parameters. Due to the circle and ellipses equations as shown in Eq. (4.1), Eq. (4.2) and Eq. (4.3), the number of estimated parameters is different. In addition, the least square method is used to estimate the circle and ellipses as shown in Eq. (4.4), Eq. (4.5) and Eq. (4.6). These equations are used to generate the hypotheses that are verified by substituting all the feature points and the obtained parameters are substituted in Eq. (4.1), Eq. (4.2) and Eq. (4.3) as follows:

$$x^2 + y^2 + Ax + By + C = 0 \tag{4.1}$$

$$x^2 + Ax + By^2 + Cy + D = 0 \tag{4.2}$$

$$x^2 + Axy + By^2 + Cx + Dy + E = 0 \tag{4.3}$$

where A, B, C, D and E are estimated parameters for a circle and ellipses.

$$\begin{pmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & \sum 1 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} -\sum x_i\left(x_i^2 + y_i^2\right) \\ -\sum y_i\left(x_i^2 + y_i^2\right) \\ -\sum x_i^2 + y_i^2 \end{pmatrix} \tag{4.4}$$

$$\begin{pmatrix} \sum x_i^2 & \sum x_i y_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i^2 & \sum y_i^4 & \sum y_i^3 & \sum y_i^2 \\ \sum x_i y_i & \sum y_i^3 & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i^2 & \sum y_i & \sum 1 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} -\sum x_i^3 \\ -\sum x_i^2 y_i^2 \\ -\sum x_i^2 y_i \\ -\sum x_i^2 \end{pmatrix} \tag{4.5}$$

$$\begin{pmatrix} \sum x_i^2 y_i^2 & \sum x_i y_i^3 & \sum x_i^2 y_i & \sum x_i y_i^2 & \sum x_i y_i \\ \sum x_i y_i^3 & \sum y_i^4 & \sum x_i y_i^2 & \sum y_i^3 & \sum y_i^2 \\ \sum x_i^2 y_i & \sum x_i y_i^2 & \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i^2 & \sum y_i^3 & \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i y_i & \sum y_i^2 & \sum x_i & \sum y_i & \sum 1 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \\ E \end{pmatrix} = \begin{pmatrix} -\sum x_i^3 y_i \\ -\sum x_i^2 y_i^2 \\ -\sum x_i^3 \\ -\sum x_i^2 y_i \\ -\sum x_i^2 \end{pmatrix} \tag{4.6}$$

The left-hand side of Eq. (4.1), Eq. (4.2) and Eq. (4.3) are considered as an error. The proportion of inliers among the data set is counted with an error threshold value. Such being the case, the three steps: random sampling, hypothesis generating and verifying are iterated until the termination criteria is met. The standard termination criteria for RANSAC means that a new data set of the next frame image arrives. Finally, the best hypothesis, which has the maximum number of inliers, is output as the estimated parameters.

4.3 Implementation

4.3.1 Design Overview

Fig. 4.1 shows an overview of the proposed system, which consists of the three clock domains. Feature points extracted from a given image are firstly stored into the feature point table. Then, the random selector module randomly selects the minimum sample points from the feature point table and stores them into FIFO1 with a 19-bit width and 16 depth. In hypothesis generation, four kinds of solvers are implemented for simultaneous equations. More specifically, two of them are a single precision FP solver and a double precision FP solver for Gauss-Jordan elimination and the rest are long integer and double precision FP solvers based on Cramer's rule. Cramer's rule with single FP cannot be used for solving the simultaneous equations due to numeric overflow in multiplication steps. Generated hypothesis parameters

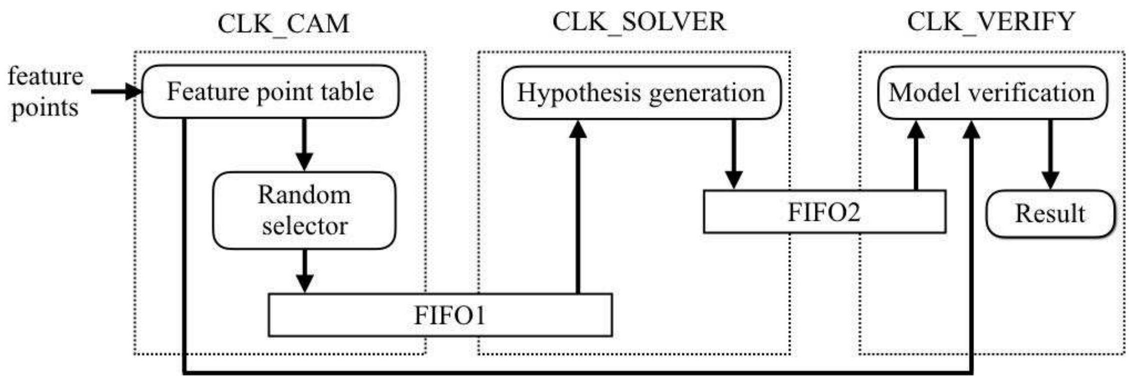


Figure 4.1: Overview of the proposed system in object fitting

are passed to the model verification module via FIFO2 with a 32-bit width and 16 depth. With the use of double buffering via dual-port RAMs, the Hypothesis generation and Model verification modules work concurrently. To the best hypothesis, the Model verification module produces a model with the maximum number of inliers.

RANSAC algorithm for fitting a model to data is as shown in Fig. 4.2. First, RANSAC selects a random subset from the extracted feature points. The minimum number of parameters needs changing according to the models, such as circles and ellipses. Then, the selected points estimate the model parameters using the least square method. S represents the absolute value of solving the left-hand side of Eq. (4.1), Eq. (4.2) and Eq. (4.3), respectively. S is compared with a predefined threshold value to split the inliers and outliers. If the value is less than threshold, data points are classified as inliers. Mismatched points are defined as outliers. The Model verification module counts the number of inliers and updates the temporal best parameters when the number of counted inliers are larger than the current maximum number of inliers. Finally, the hypothesis with the largest inlier set during a given period is selected as the best model.

4.3.2 Hypothesis Generation

4.3.2.1 Cramer's rule

In the case of $n \times n$ matrix, Cramer's rule can be generalized a system of n linear equations in n unknowns as follows:

$$A\boldsymbol{x} = \boldsymbol{b} \tag{4.7}$$

where $n \times n$ matrix, A, has a nonzero determinant, $\boldsymbol{x}$ and $\boldsymbol{b}$ denote column vectors. In that case, the unknown values are given by Cramer's formulas:

$$x_i = \frac{|A_i|}{|A|} \tag{4.8}$$

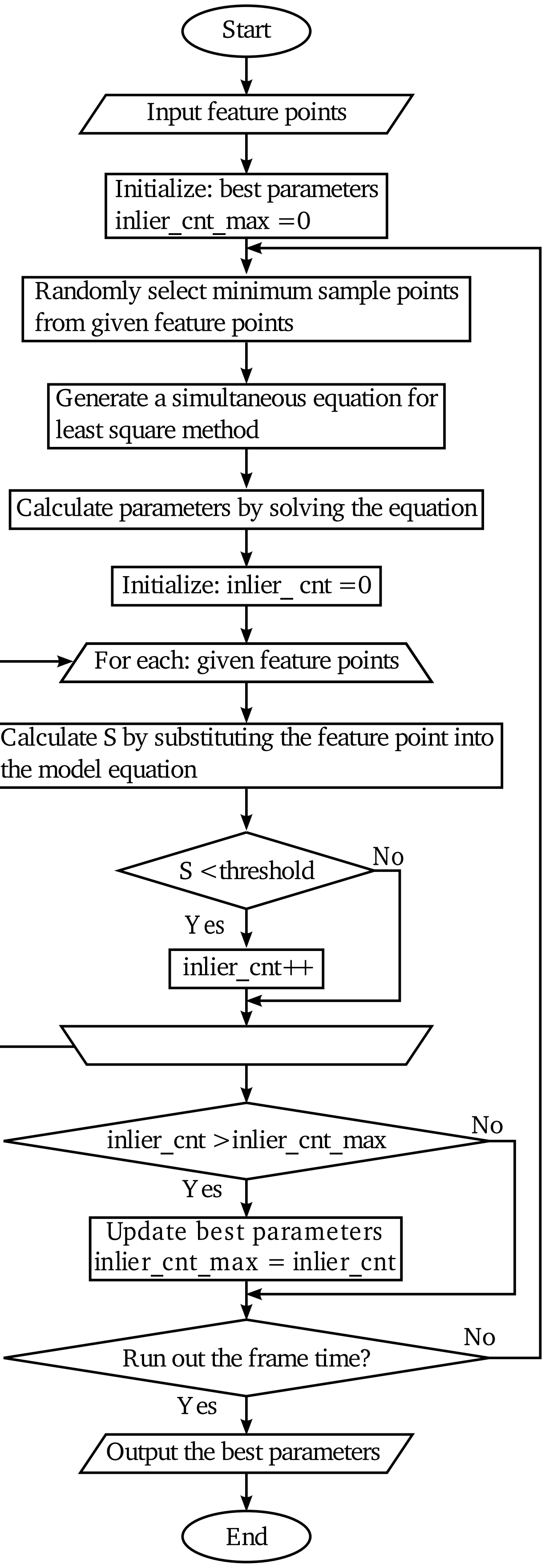

Start
Input feature points
Initialize: best parameters
inlier_cnt_max =0
Randomly select minimum sample points from given feature points
Generate a simultaneous equation for least square method
Calculate parameters by solving the equation
Initialize: inlier_ cnt =0
For each: given feature points
Calculate S by substituting the feature point into the model equation
S <threshold
No
Yes
inlier_cnt++
inlier_cnt >inlier_cnt_max
No
Yes
Update best parameters
inlier_cnt_max = inlier_cnt
Run out the frame time?
No
Yes
Output the best parameters
End

Figure 4.2: Flowchart of the approach based on RANSAC algorithm

where x_i represents the unknowns of the system and A_i refers a matrix obtained from A by replacing the i-th column by the column vector $\boldsymbol{b}$. The determinants are used to solve the systems according to Cramer's rule. The Leibniz formula for the determinant $|A|$ of order n is as follow:

$$|A| = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i} \tag{4.9}$$

where S_n denotes the set of all permutations of the integers $\{1, 2, ..., n\}$, σ_i denotes the value in the i-th number after the reordering σ. For each permutation σ, $\mathrm{sgn}(\sigma)$ denotes the signature of σ which represents $+1$ for even σ and -1 for odd σ. In general, Cramer's rule is much less efficient for large systems of equations with regard to its higher computational complexity compared with other methods such as Gauss-Jordan algorithm. Nevertheless, it could be efficient for small matrix operations because of its systematic and regular approach. Generally, the number of terms in the determinant of an $n \times n$ matrix is $n!$. Hence, the determinant of a 3×3 matrix contains six terms. For a 4-th order determinant, the sum of the terms is 24 as shown in Eq. (4.10). Similarly, an explicit formula for the determinant of a 5×5 matrix is a sum of 120 products.

A memory table is used as a *sign and address table*. In this table, the elements A_{i,σ_i} are the elements $A_{i,\sigma_{ji}}$ of the matrix A where the index j is defined as the j-th element of permutation σ_j for column vectors of the matrices (3×3 bits) of 3×3 matrix, (3×4 bits) of 4×4 matrix and (3×5 bits) of 5×5 matrix, respectively and the value of $\mathrm{sgn}(\sigma_j)$ (1 bit). For example, the first term of Eq. (4.10) describes $\{1'b_0, 3'd_0, 3'd_1, 3'd_2, 3'd_3\}$ in the table.

$$\begin{aligned} det(A) = {} & a_{11}a_{22}a_{33}a_{44} + a_{11}a_{24}a_{32}a_{43} + a_{11}a_{23}a_{34}a_{42} \\ & + a_{12}a_{21}a_{34}a_{43} + a_{13}a_{21}a_{32}a_{44} + a_{14}a_{21}a_{33}a_{42} \\ & + a_{12}a_{23}a_{31}a_{44} + a_{14}a_{22}a_{31}a_{43} + a_{13}a_{24}a_{31}a_{42} \\ & + a_{12}a_{24}a_{33}a_{41} + a_{13}a_{22}a_{34}a_{41} + a_{14}a_{23}a_{32}a_{41} \\ & - a_{11}a_{22}a_{34}a_{43} - a_{11}a_{23}a_{32}a_{44} - a_{11}a_{24}a_{33}a_{42} \\ & - a_{12}a_{21}a_{33}a_{44} - a_{14}a_{21}a_{32}a_{43} - a_{13}a_{21}a_{34}a_{42} \\ & - a_{12}a_{24}a_{31}a_{43} - a_{13}a_{22}a_{31}a_{44} - a_{14}a_{23}a_{31}a_{42} \\ & - a_{12}a_{23}a_{34}a_{41} - a_{14}a_{22}a_{33}a_{41} - a_{13}a_{24}a_{32}a_{41} \end{aligned} \tag{4.10}$$

A data fetch mechanism consisting of *column tables* and a *sign-and-address table* is used to calculate the determinants with a simple architecture. Fig. 4.3 shows the structure of this mechanism in the case of $n = 4$. Initially, each column vector of the matrix A is stored in the column tables, which consists of n banks. The addresses to

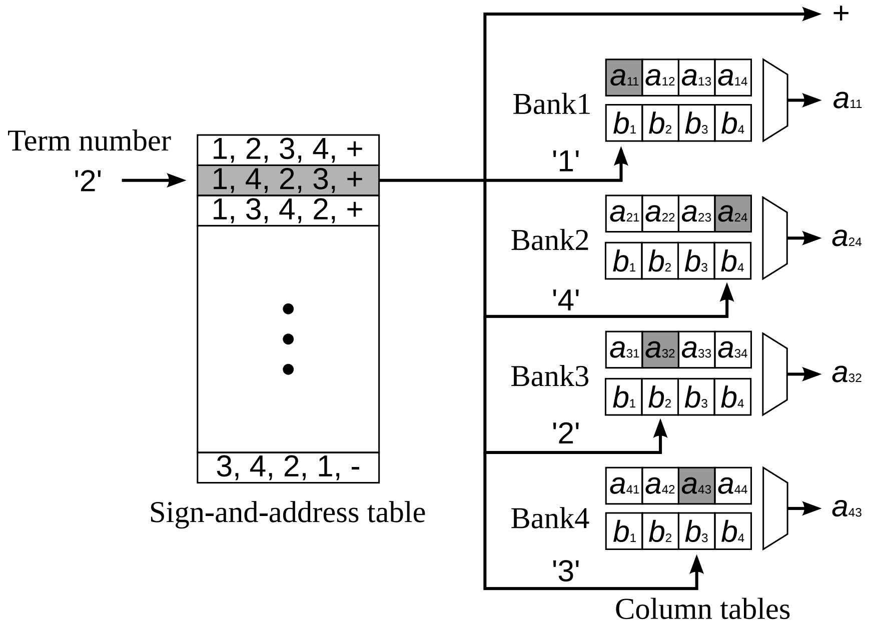


Figure 4.3: Data fetch mechanism for the Cramer module

the column tables are given by the sign-and-address table, whose i-th entry contains i-th permutation σ_i and the value of $\mathrm{sgn}(\sigma_i)$. For example, the second entry of the sign-and-address table contains '1', '4', '2', '3', and '+', corresponding to the second term Eq. (4.10). To support calculation of $|A_i|$, each bank of the column table can also be switched to output the column vector $\boldsymbol{b}$.

Fig. 4.9 illustrates the entire structure of the proposed Cramer module in the case of $n = 5$. A simple counter generates an address to the sign-and-address table every clock cycle. Next step is to fetch the corresponding elements of the matrix from the five banks of the column tables. After multiplying the five elements and the sign, the result is accumulated. This process repeats 120 times to get the determinant of the matrix. While multiplication steps can be fully pipelined, pipeline stalls occur in the final accumulation step due to the latency of the adder. For the sake of eliminating pipeline stalls, the calculation of determinants involves interleaving the elements of six matrices $(A, A_1, \ldots, A_5)$ using an adder with 6-cycle latency. Hence, six determinants are obtained every clock cycle after computing 720 sets of multiplication. Finally, the last five determinants are divided by the first determinant to get the ellipse parameters.

According to Cramer's rule, the determinants of matrices can be mostly calculated by integer arithmetic. However, calculation of the determinants needs a wide dynamic range. For this reason, double precision FP arithmetic or long integer arithmetic is required. Cramer modules with double precision FP arithmetics are illustrated in Fig. 4.5, Fig. 4.7 and Fig. 4.9. Fig. 4.4, Fig. 4.6 and Fig. 4.8 show the overview of the Cramer module with long integer arithmetics on an FPGA. Each label on the arrows shows the bit width required to process 640×480-pixel images.

After multiplying the five elements and the sign as shown in Fig. 4.8, the bit size extends 205 bits. Then, the addition of corresponding values repeats 120 times to get the determinant of the matrix. After the addition process had executed, the long integer values are reduced to 64 bits without changing the values to avoid the numeric overflow for integers to single precision FP conversion. In that case, the conversion process entails large barrel shifters. According to the column vector $\boldsymbol{b}$, $|A_i|$ is calculated by using $b_{\sigma_{ji}}$ instead of $a_{i,\sigma_{ji}}$ from the i-th column table at the j-iteration, where σ_{ji} means i-th number of the permutation σ_j.

In accordance with the arithmetic processes, the hypothesis generator for ellipse estimation consists of two 41-bit integer multipliers, one 82-bit integer multiplier, one (164×41 bits) integer multiplier, one 205-bit integer adder and one single (FP) divider for integer version. With respect to this estimation, double FP version includes four double precision FP multipliers, one double precision FP adder and one double precision FP divider as shown in Fig. 4.9. In contrast to long integer method, the input matrices are double precision FP format and the values of six determinants are converted into single precision FP representation after the final accumulation step.

In 4×4 matrix, the hypothesis generation comprises two 40-bit integer multipliers, one 80-bit integer multiplier, one 160-bit integer adder and one single precision FP divider for integer version. In that case, the hypothesis generator for double FP contains three double precision FP multipliers, one double precision FP adder and one double precision FP divider. For circle estimation, one 32-bit integer multiplier, one (64×32 bits) integer multiplier, one 96-bit integer adder and one single precision FP divider require for integer version. The double FP version includes two double precision FP multipliers, one double precision FP adder and one double precision FP divider. For the long integer arithmetic, the integer size needs to reduce 1 bit or 9 bits in accordance with the size of the matrix. All estimated parameters are outputs as single precision FP values. Hence, some format converters involve in both arithmetic processes.

4.3.2.2 Gauss-Jordan elimination

Gauss-Jordan elimination works with the augmented matrix in order to solve a system of simultaneous equations. This method is a systematic way for solving very large systems of equations. Moreover, Gauss-Jordan algorithm does not need the backward substitution which is highly sequential process compared with Gaussian elimination. In contrast, RANSAC needs solving simultaneous equations many times because of changing the feature points selection. The best estimated parameters are adopted from this process. The accuracy for the best estimated parameters is the

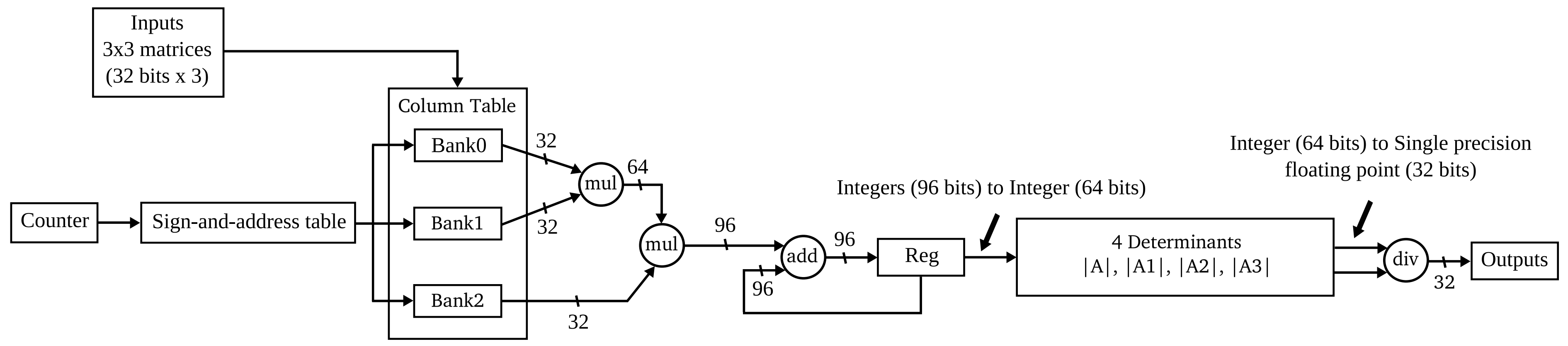


Figure 4.4: Overview of Cramer module for long integer (circle)

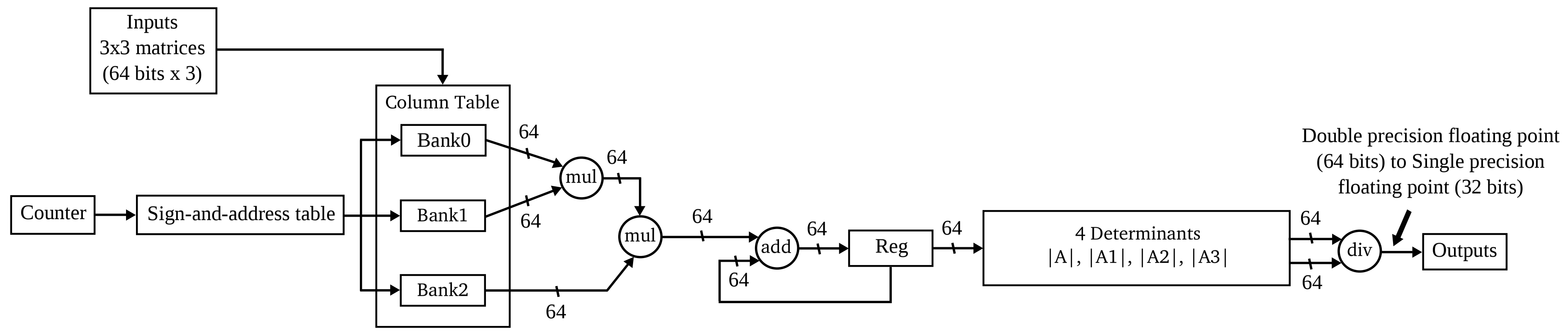


Figure 4.5: Overview of Cramer module for double precision FP (circle)

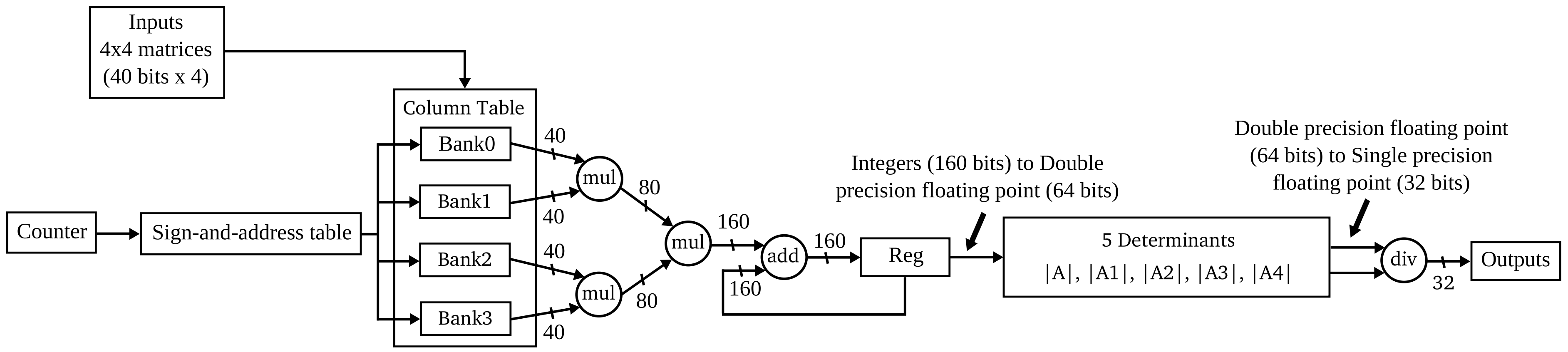

Inputs
4x4 matrices
(40 bits x 4)
Counter
Sign-and-address table
Column Table
Bank0
Bank1
Bank2
Bank3
40
40
40
40
mul
mul
80
80
mul
160
add
160
160
Reg
Integers (160 bits) to Double precision floating point (64 bits)
5 Determinants
|A|, |A1|, |A2|, |A3|, |A4|
Double precision floating point (64 bits) to Single precision floating point (32 bits)
div
32
Outputs

Figure 4.6: Overview of Cramer module for long integer (ellipse 4 points)

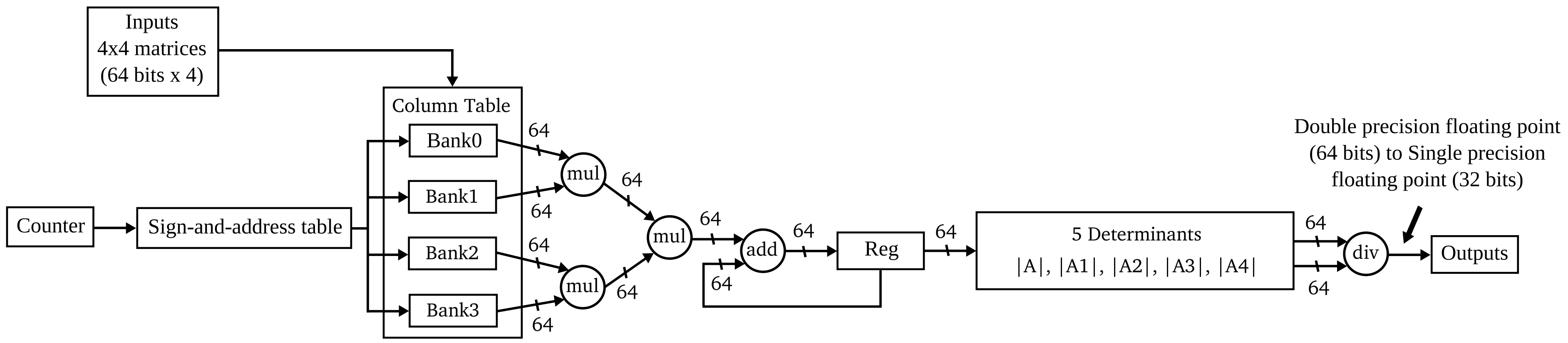


Figure 4.7: Overview of Cramer module for double precision FP (ellipse 4 points)

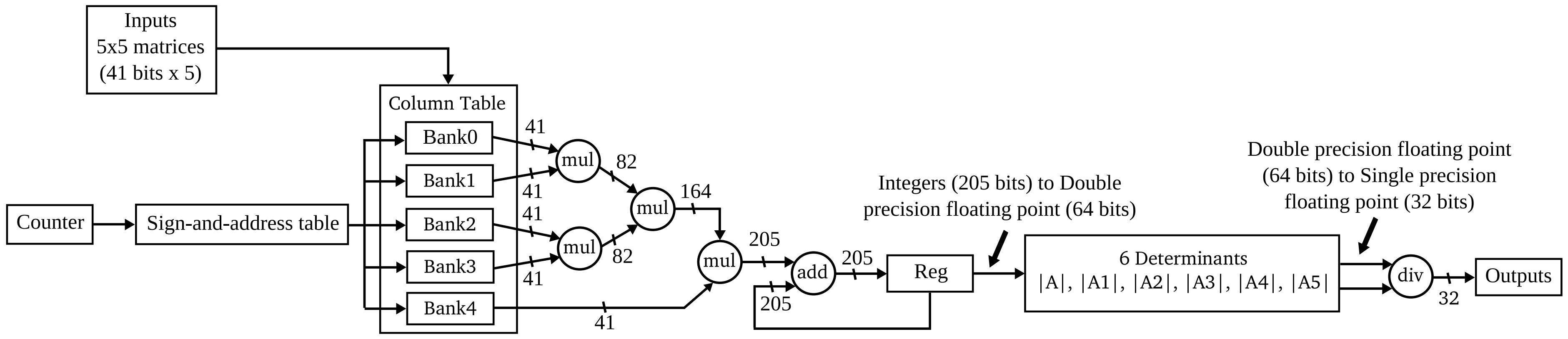

Inputs
5x5 matrices
(41 bits x 5)
Counter
Sign-and-address table
Column Table
Bank0
Bank1
Bank2
Bank3
Bank4
41
41
41
41
41
mul
mul
82
82
mul
164
mul
205
add
205
205
Reg
Integers (205 bits) to Double precision floating point (64 bits)
6 Determinants
|A|, |A1|, |A2|, |A3|, |A4|, |A5|
Double precision floating point (64 bits) to Single precision floating point (32 bits)
div
32
Outputs

Figure 4.8: Overview of Cramer module for long integer (ellipse 5 points)

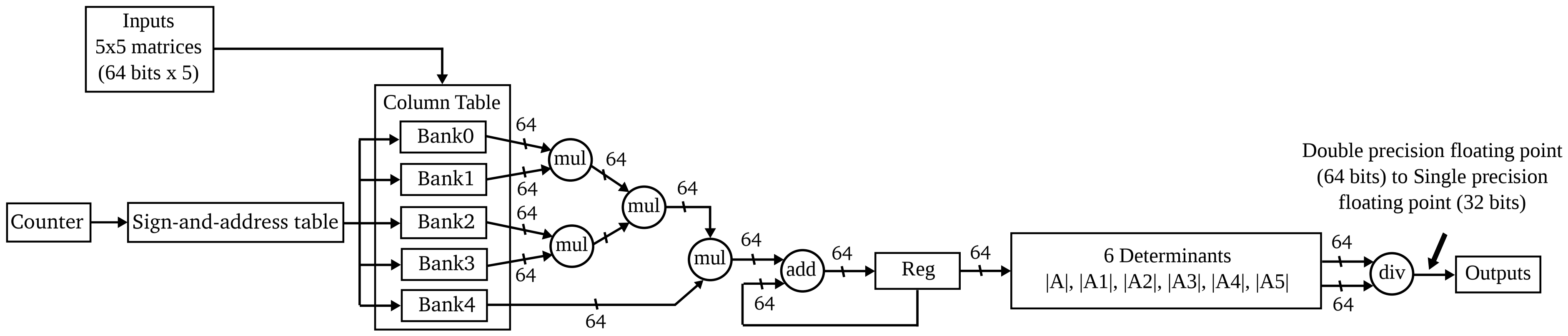


Figure 4.9: Overview of Cramer module for double precision FP (ellipse 5 points)

only thing that matters. Hence, pivot exchanging was not considered in this design.

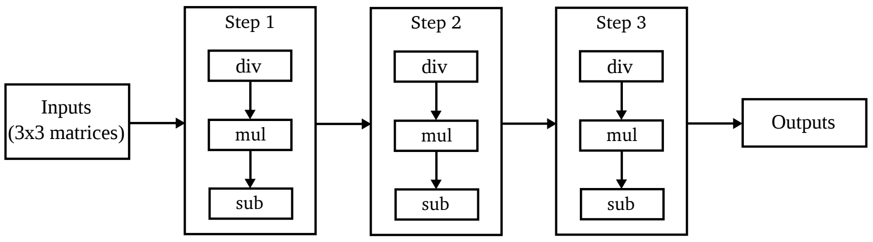


Figure 4.10: Gauss-Jordan Algorithm for circle estimation

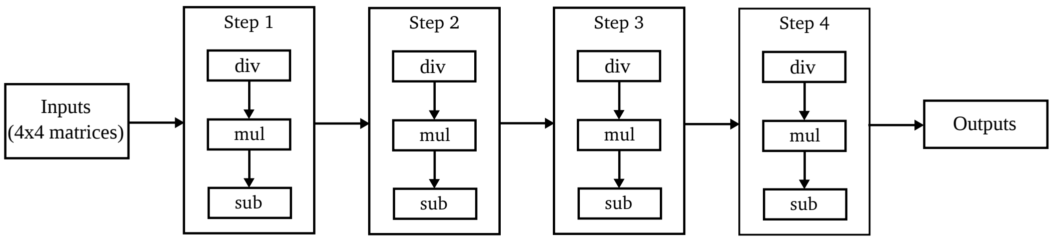


Figure 4.11: Gauss-Jordan Algorithm for ellipse estimation

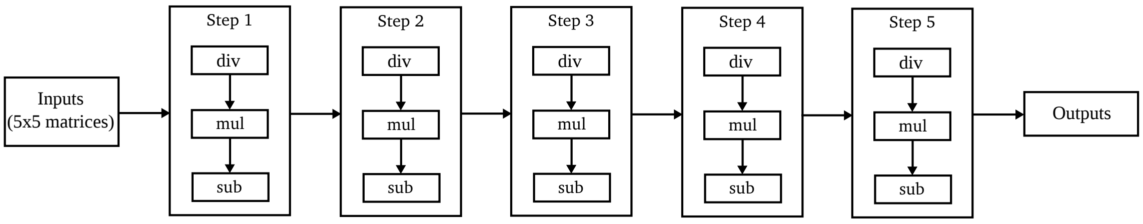


Figure 4.12: Gauss-Jordan Algorithm for ellipse with angle estimation

Fig. 4.10 and Fig. 4.11 illustrate the circle and ellipse estimation with Gauss-Jordan method. Moreover, Fig. 4.12 shows the structure of a hypothesis generation module for ellipse with angle estimation based on the Gauss-Jordan method. It consists of cascaded five sub-modules and the computation is pipelined. Each sub-module consists of division, multiplication and subtraction. The row operations are preformed using the above arithmetic operations until the matrices are in reduced row echelon form. These sub-modules work in a macro pipelined manner, that is, a new hypothesis generation can be started after the first sub-module finishes its calculation.

Integer method cannot apply to Gauss-Jordan elimination compared to Cramer's rule because division process involves in every step of modules. Since a dynamic range required for Gauss-Jordan elimination is moderate, all the arithmetic modules were implemented in the single-precision FP format. On the other hand, Gauss-Jordan elimination for double FP also requires for fair comparison of Cramer module with double FP. Although the computation is the same as single FP of Gauss-Jordan elimination, double-precision Gauss-Jordan algorithm needs to change the output size of each step using double to single FP converter.

4.4 Evaluation Environment and Method

The proposed systems are implemented on a Xilinx Kintex-7 xc7k325t FPGA using ISE design tools 14.7. To evaluate the estimation accuracy of each solver, synthetic benchmark models are prepared for circles, aligned ellipses, and ellipses with angles. First, an original model is randomly generated on a 640×480-pixel frame. A center coordinate (x_i, y_i), a major radius r_a, a minor radius r_b, and slope ϕ of the ellipse are given by uniform random numbers so that $160 < x_i \leq 480$, $80 < y_i \leq 240$, $20 < r_a \leq 80$, $20 < r_b \leq 80$, and $0 < \phi \leq 2\pi$, respectively. Then, a total of 128 points are randomly selected from the circumference of the ellipse as feature points. Finally, to mimic the uncertainties of feature point extraction, noise with the normal distribution ($\mu = 0$, $\sigma = r_b/50$) is added to each coordinate of the feature points.

The Random selector module selects the minimum sample points from the feature point table accordance to the random addresses generated by a 32-stage linear feedback shift register. Since 640×480 pixel display, 19-bit registers are required to generate the matrix. After generating the matrix for the least square method, each element of the matrix is converted to the corresponding data type such as double precision floating-point (FP), single precision FP, or integer, depending on an evaluated solver module. Then, the RANSAC system estimates parameters of the original model from the coordinates of feature points with the noise. For given 128 feature points, the systems iteratively performs the process of the feature point selection, hypothesis generation by solving simultaneous equations, and evaluation of the generated hypothesis. Four solver modules, CRAMER with long-integer arithmetic, CRAMER with double precision FP arithmetic, GAUSS-JORDAN with single precision FP arithmetic, GAUSS with double precision FP arithmetic, are compared. As aforementioned, CRAMER with single precision FP arithmetic cannot be implemented due to the dynamic range required for calculating determinants.

For real-time execution, the system repeats these processes for 400,000 clock cycles, which corresponds to 5 ms at 80 MHz. A total of 100 benchmark models were generated and evaluated for each solver. The estimation accuracy was evaluated using three metrics: root mean square error (RMSE) for the estimated center coordinates, RMSE for the estimated lengths of the radius, and relative root mean square error (RRMSE) for the estimated areas of circles and ellipses. For example, RMSE for the estimated center coordinates is given by Eq. (4.11) and Eq. (4.12).

$$\delta_i = \sqrt{\Delta x_i^2 + \Delta y_i^2} \tag{4.11}$$

Table 4.1: Resource usage of each implementation

	CRAMER						GAUSS-JORDAN					
	Long Integer			Double Float			Single Float			Double Float		
	3x3	4x4	5x5	3x3	4x4	5x5	3x3	4x4	5x5	3x3	4x4	5x5
FF	4998(1%)	5234(1%)	8089(1%)	7606(1%)	7332(1%)	8617(2%)	10095(2%)	11897(2%)	14666(3%)	15301(3%)	18109(4%)	21914(5%)
LUT	5796(2%)	6996(3%)	9853(4%)	9345(4%)	9751(4%)	11189(5%)	8854(4%)	10339(5%)	12753(6%)	18151(8%)	23052(11%)	29373(14%)
BRAM36E1	-	4(1%)	5(1%)	3(1%)	4(1%)	5(1%)	-	-	-	5(1%)	7(1%)	10(2%)
BRAM18E1	6(1%)	6(1%)	6(1%)	6(1%)	6(1%)	6(1%)	8(1%)	10(1%)	13(1%)	5(1%)	5(1%)	5(1%)
DSP48E1s	26(3%)	43(5%)	91(10%)	36(4%)	46(5%)	64(7%)	30(3%)	39(4%)	50(5%)	57(6%)	71(8%)	95(11%)

Table 4.2: Performance of each implementation

	CRAMER						GAUSS-JORDAN					
	Long Integer			Double Float			Single Float			Double Float		
	3x3	4x4	5x5	3x3	4x4	5x5	3x3	4x4	5x5	3x3	4x4	5x5
Latency of fitting (clock cycles)	84	190	821	109	211	831	183	245	311	192	245	346
Throughput of fitting (estimations/s)	$1.07\text{x}10^6$	$0.53\text{x}10^6$	$0.12\text{x}10^6$	$1.34\text{x}10^6$	$0.55\text{x}10^6$	$0.14\text{x}10^6$	$1.50\text{x}10^6$	$1.15\text{x}10^6$	$0.75\text{x}10^6$	$0.78\text{x}10^6$	$0.62\text{x}10^6$	$0.44\text{x}10^6$
Max frequency (MHz)	89.7	98.4	97.4	146.5	116.6	119.4	273.8	281.6	233.6	150.3	151.0	150.8

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\delta_i^2} \tag{4.12}$$

where Δx_i and Δy_i show the difference in the center point coordinates between the original and estimated ellipses (or circles) for the i-th trial, while N shows the number of fitting trials and is 100 for this experiment. The RMSE for the ellipse estimated lengths is δ_i divided by 2 due to the two different radii. The RRMSE for area estimation is given by Eq. (4.13) and Eq. (4.14).

$$\gamma_i = \sqrt{\frac{(AE_i - AO_i)^2}{{AO_i}^2}} \tag{4.13}$$

$$RRMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\gamma_i^2} \tag{4.14}$$

where the estimated area and original area of the ellipse (or circle) in the i-th trial are denoted as AE_i and AO_i, respectively.

4.4.1 Resource Usage

Table 4.1 illustrates the five different types of FPGA resources for the four types of solvers. For circle estimation, CRAMER solver with long integer method utilizes the less usage in overall resources. In 4×4 matrix, it is evident that CRAMER for both versions have lower resource usages in FFs and LUTs than GAUSS. When it comes to 5×5 matrix, CRAMER remains the lowest resource usage except DSP48E1s. In accordance with DSP48E1s usage, multipliers play an important role in CRAMER and DSP48E1s hard macro modules are essential to implement the arithmetic operations efficiently. In this system, BRAMs were mainly utilized for implementing FIFO modules.

4.4.2 Performance

Table 4.2 gives information on the performance of latency, throughput and maximum clock frequency for circle and ellipses estimations. *Throughput* was defined as the number of hypothesis generation per second in this work. Although the latency of both CRAMER solvers was lower than GAUSS solver except 5×5 matrix, the maximum clock frequency was taken into account for the calculation of throughput. For this reason, GAUSS achieved the maximum throughput value. With the best

Table 4.3: Errors of each implementation

Errors	CRAMER						GAUSS-JORDAN					
	Long Integer			Double Float			Single Float			Double Float		
	3x3	4x4	5x5	3x3	4x4	5x5	3x3	4x4	5x5	3x3	4x4	5x5
RMSE for coordinate (pixels)	0.674	1.167	0.612	0.559	1.055	0.987	0.354	1.103	0.696	0.320	1.197	0.619
RMSE for radius (pixels)	0.317	0.805	0.595	0.283	0.933	1.319	0.184	0.919	0.831	0.142	0.721	0.504
RRMSE for area	0.004	0.014	0.007	0.000	0.015	0.029	0.000	0.016	0.004	0.000	0.018	0.004

performance, the improved throughput may make the accuracy of the fitting better in RANSAC algorithm. Long integer versions showed the degradation of the clock frequency. One of the reasons for this frequency degradation is large barrel shifters, which are required to convert long integer values to FP values prior to the final FP divider. Even though CRAMER solver using long integer achieved the minimum throughput value, it can successfully estimate the reasonable parameters for ellipses and circle fittings. The estimation accuracy will be discussed in the next section.

Table 4.4: Statistics of errors in 100 estimation trials for circle estimation

	CRAMER		GAUSS-JORDAN	
	Long Integer	Double Float	Single Float	Double Float
Maximum absolute error of coordinate (pixels)	1.336	1.314	0.838	0.986
Minimum absolute error of coordinate (pixels)	0.055	0.032	0.032	0.000
Standard deviation of coordinates (pixels)	0.301	0.249	0.180	0.185
Maximum absolute error of radius (pixels)	0.896	0.571	0.620	0.365
Minimum absolute error of radius (pixels)	0.000	0.000	0.000	0.000
Standard deviation of radius (pixels)	0.201	0.155	0.118	0.080
Maximum absolute relative error of area	0.032	0.000	0.000	0.000
Minimum absolute relative error of area	0.000	0.000	0.000	0.000
Standard deviation of area	0.005	0.000	0.000	0.000

4.4.3 Accuracy

Table 4.3 compares three RMSE values in terms of the coordinate, radius and area with four different arithmetic processes. Among the four processes, the GAUSS

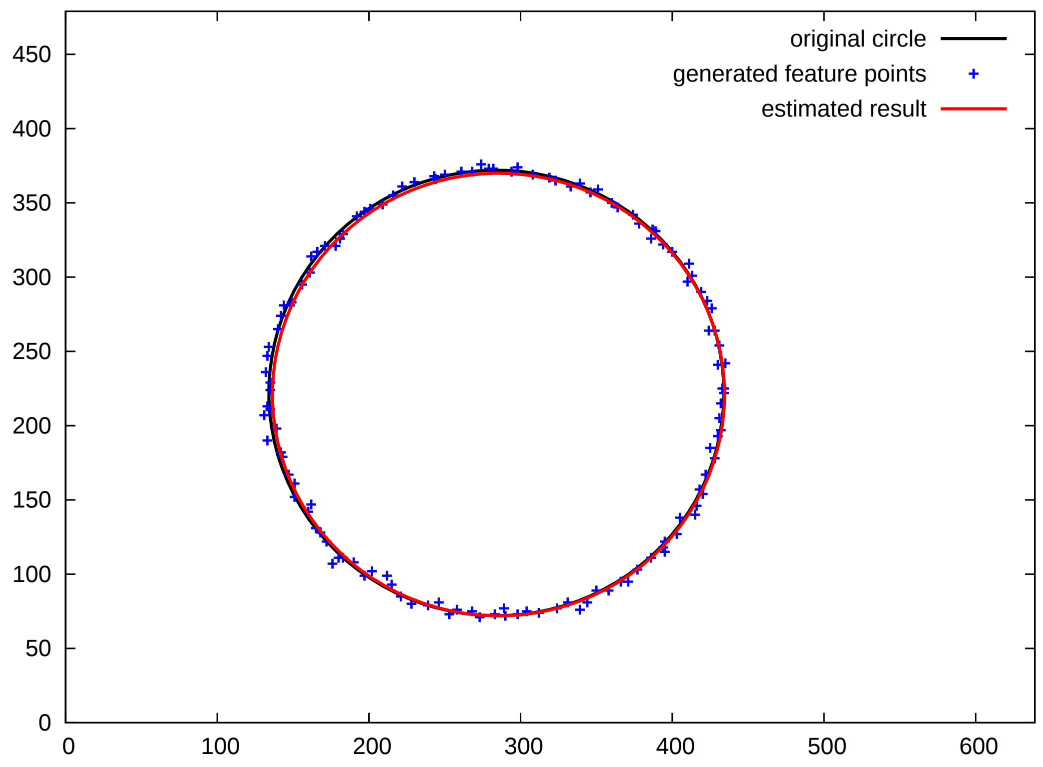


Figure 4.13: Circle estimation example

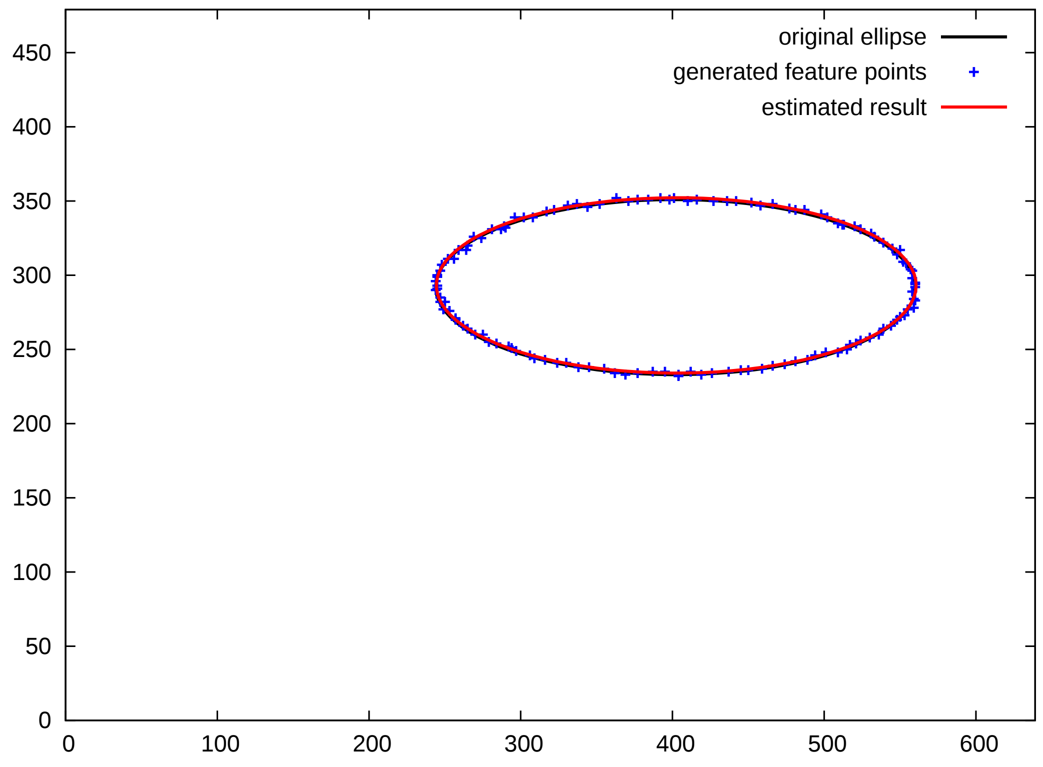


Figure 4.14: Ellipse estimation example

solver provided the highest accuracy for 3×3 matrices whereas CRAMER with long integer version showed the lowest accuracy. According to the ellipse estimations, the differences in RMSE values are not significant. According to the long integer method, Fig. 4.13 and Fig. 4.14 demonstrate estimation examples, showing the original model, generated feature points with noises, and estimated models. In accordance with the comparisons, the estimation errors among circle and ellipses were around one pixel even for the largest error in the radius of the floating-point version of CRAMER.

Moreover, Table 4.4 compares the statistics of errors in terms of maximum, minimum and standard deviation values over 100 trials. Fig. 4.13 and Fig. 4.14 demonstrate the estimation examples of CRAMER long integer method, showing the original model, generated feature points with noises, and estimated models. Table 4.3 compares three RMSE values in terms of the coordinate, radius and area with four different arithmetic processes. The number of digits after decimal point 30-digit precision is used in accumulation for statistical reliability. The average errors are displayed with double format. After 100 trials, the average errors display 7 digits to simplify the accuracy comparison because of the different number of pixel. The detailed error analysis for model fitting is concerned with the accuracy of three parameters such as center points (x, y), radius and area.

The error values are identified between the actual and estimated parameters with root mean square error method. RMSE for all parameters are generated and calibrated 100 times. Even though CRAMER with long integer version showed the lowest accuracy among the four processes whereas the GAUSS solver provided the highest accuracy, the errors are acceptable. The peak error of this method makes the accuracy degrade and occurs due to the difference of true value and estimated value.

4.4.4 Memory Optimization

When it comes to the resource utilization, memory bit width optimization makes the circuit area improved. For this reason, the number of bits are optimized depending on the maximum bit width for every multiplication step in Cramer modules coupled with long integer arithmetics. Fig. 4.15 depicts the circle estimation with optimized Cramer module. As mentioned above, element-wise multiplication is an essential step for finding the determinants of circle. In the circle estimation, the biggest bit width for element-by-element multiplication of two values becomes 51 bits, i.e. $x_i y_i$ is multiplied by $-x_i \Sigma(x_i^2 + y_i^2)$. Then, maximum bit width for three elements multiplication gives 62 bits. Fig. 4.16 and Fig. 4.17 illustrate the optimized long integer design for ellipse estimations. In terms of 4 elements multiplication, the bit

Table 4.5: FPGA mapping results for Optimized CRAMER (long integer) designs

	Resource utilization					Performance comparison		
	FF	LUT	BRAM36E1	BRAM18E1	DSP48E1s	Latency of fitting (clock cycles)	Throughput of fitting (estimations/s)	Max frequency (MHz)
3x3	4915(1%)	5793(2%)	-	6(1%)	25(2%)	83	$1.27\text{x}10^6$	105.7
4x4	5013(1%)	6459(3%)	4(1%)	6(1%)	41(4%)	190	$0.52\text{x}10^6$	98.1
5x5	7417(1%)	8756(4%)	5(1%)	6(1%)	82(9%)	821	$0.12\text{x}10^6$	98.98

width of design alternative gets one half of the bit width in original one. Moreover, the bit width of five elements multiplication in optimized design can reduce 1.46 times compared to the original version of long integer method.

Table 4.5 shows the resource utilization and performance comparison of three different matrices. Each new design can reduce the utilization of FFs and LUTs compared to the same arithmetic type of each original design as shown in Table 4.1. DSP usage obviously lessens 1% for all model estimations. In terms of the maximum frequency, new design improves 1.18 times than the original frequency. Moreover, throughput also increases from 1.07×10^6 to 1.27×10^6. Hence, memory optimization impacts on the resource utilization and performance on FPGA.

4.4.5 Comparative Discussion

The above evaluation results reveal the tradeoff relationship between circuit size, speed performance and estimation accuracy. In most cases, CRAMER with long integer arithmetic offers the lower amount of resource usage, moderate throughput and the lowest accuracy among all solver types. On the other hand, Gauss-Jordan elimination with single FP utilizes the higher resources, maximum throughput and higher accuracy. Since the estimation error of CRAMER is about one pixel and FFs usage is less than 50% of GAUSS, the CRAMER approach can be considered to be advantageous especially in embedded and mobile applications such as robotics and unmanned aerial vehicles, where compact circuit size and high energy efficient are often preferred. Accordingly, CRAMER with long integer method estimates the ellipse using less amount of resources and moderate accuracy. On the other hand, in the applications where precise accuracy is more important than resource utilization, the Gauss-Jordan method would be desirable. In addition, when the number of available DSP blocks is severely restricted compared to other general resources, the Gauss-Jordan algorithm will have an advantage.

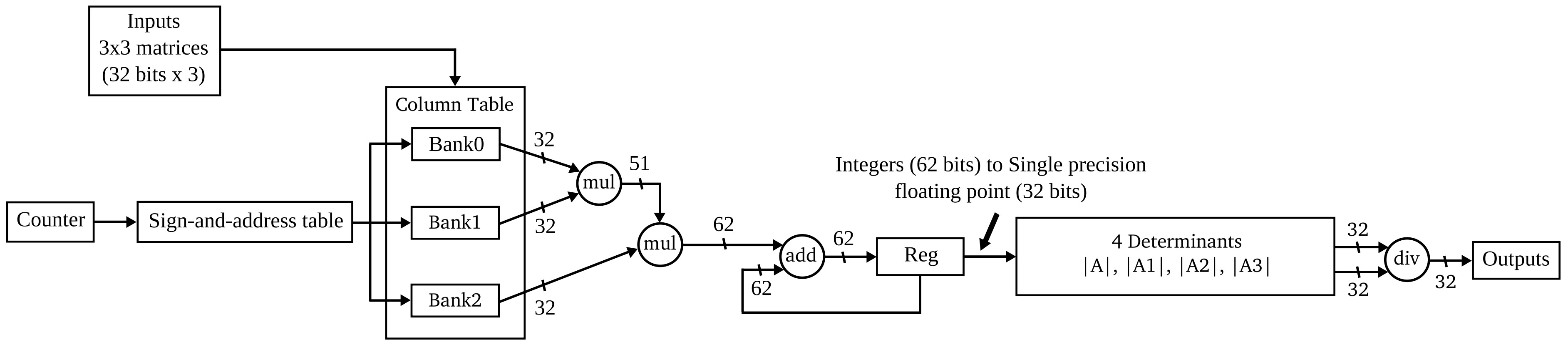
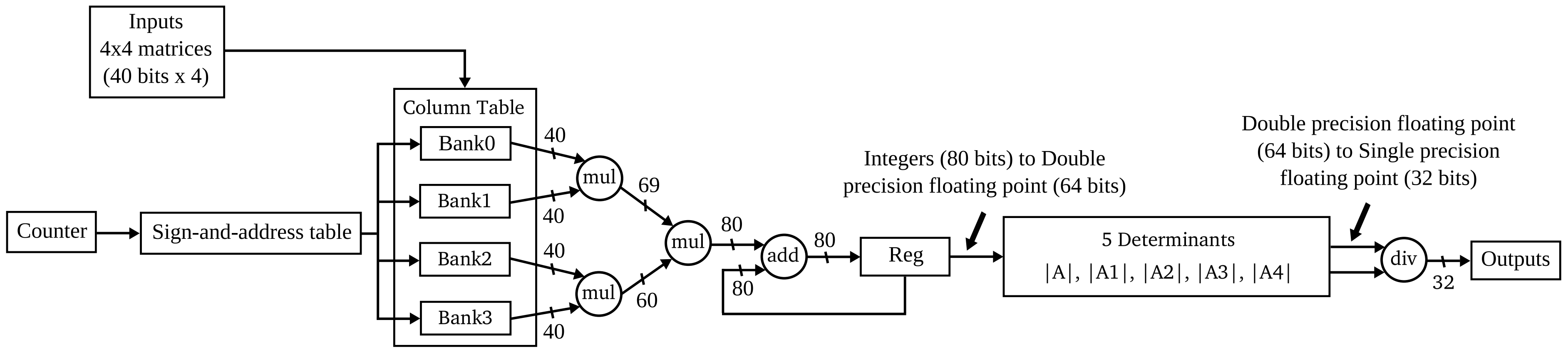


Figure 4.15: Overview of optimized Cramer module for long integer (circle)



Inputs
4x4 matrices
(40 bits x 4)
Column Table
Bank0
Bank1
Bank2
Bank3
Counter
Sign-and-address table
40
40
40
40
mul
mul
69
60
mul
80
add
80
80
Reg
Integers (80 bits) to Double precision floating point (64 bits)
5 Determinants
|A|, |A1|, |A2|, |A3|, |A4|
Double precision floating point (64 bits) to Single precision floating point (32 bits)
div
32
Outputs

Figure 4.16: Overview of optimized Cramer module for long integer (ellipse 4 points)

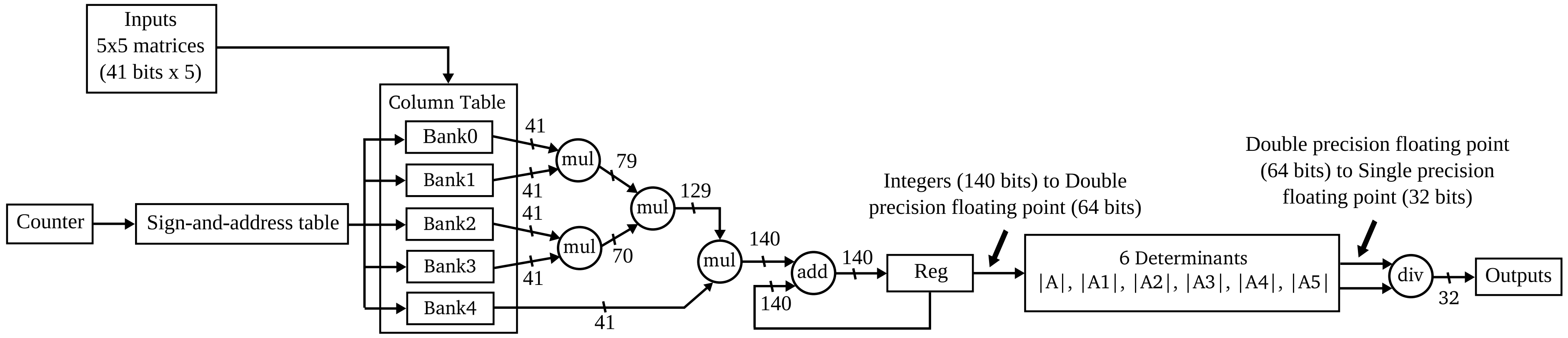

Inputs
5x5 matrices
(41 bits x 5)
Counter
Sign-and-address table
Column Table
Bank0
Bank1
Bank2
Bank3
Bank4
41
41
41
41
41
mul
mul
79
70
mul
129
mul
140
add
140
140
Reg
Integers (140 bits) to Double precision floating point (64 bits)
6 Determinants
|A|, |A1|, |A2|, |A3|, |A4|, |A5|
Double precision floating point (64 bits) to Single precision floating point (32 bits)
div
32
Outputs

Figure 4.17: Overview of optimized Cramer module for long integer (ellipse 5 points)

4.5 Summary

This chapter presented the comparative evaluation results on FPGA implementation of robust circle and ellipse estimations based on the RANSAC algorithm. Among model estimation processes, the hypothesis generation processes with the least square method are especially emphasized and four types of solver architectures are compared. In the evaluation experiments, each type of hardware solved the simultaneous equations repeatedly for synthetic benchmarks of three different models. The four solvers with the comparison criteria for resource usage, performance and accuracy are described.

Furthermore, each new design with optimized memory bit width achieved the improvement of circuit area and performance on FPGA. The evaluation results have shown that the long integer method of the Cramer's approach can reduce required FPGA resources except for DSP blocks with acceptable performance and accuracy, which would not be the case for software implementation. On the other hand, it has been revealed that Gauss-Jordan's approach with FP arithmetic can achieve better accuracy with less usage of DSP blocks.

Although this dissertation approaches to circle and ellipse estimations for the ease of quantitative comparison, other geometric shapes, such as triangle, rectangle or square, can also be implemented by substituting the related equation in Eq. (4.1), Eq. (4.2), Eq. (4.3), Eq. (4.4), Eq. (4.5) and Eq. (4.6). For example, the area of triangle is

$$area_{triangle} = \pm\frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}. \tag{4.15}$$

The matrix of three vertices triangle to generate the hypothesis is

$$\det(A) = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{4.16}$$

$$= x_1y_2 + x_2y_3 + x_3y_1 - x_1y_3 - x_2y_1 - x_3y_2.$$

Chapter 5

FPGA-based Real-time Object Tracking

5.1 Overview

Recent technological developments regarding the object tracking have led to many computer vision applications such as robotics, video surveillance, biomedical engineering, etc. According to the filtering problems in high-dimensional space, particle filters can trace the object well and enable a fully nonlinear and non-Gaussian analysis step whereas Kalman filters are untraceable. Moreover, the particle filter has become popular among the nonparametric filters in marker-less tracking applications. Study of real-time processing for camera image using stream processing on FPGAs is also popular to fulfill the demands for fast and robust object tracking with low energy consumption. Hence, implementation of the particle filter in stream processing is a promising approach for effective object tracking systems.

On the other hand, the resampling operation is the bottleneck in real-time particle filter implementation [63, 64]. Difficulty of parallel resampling comes from it needs information of all particles and thus computational dependency arises. To cope with this problem, the FPGA optimized resampling (FO-resampling) [47] has been proposed, where virtual particles are randomly generated around the real particle to avoid the elimination of real particles with small weights. Although effectiveness of FO-resampling method has been shown with mathematical models [47], attempts to combine the FO-resampling with stream processing have not been addressed.

With the deep-pipelined stream-oriented image processing architecture, a real-time throughput can be achieved at a relatively low clock frequency without requiring any external memories. Such an approach has already shown and that is of benefit for a variety of image applications in terms of performance and power consumption [42, 49, 54, 65]. In this chapter, a stream-based architecture of a particle filter is proposed and implemented based on FO-resampling method. In addition, five kinds of FPGA implementation are compared with the different resource usages and clock frequencies for the purpose of achieving compact hardware architecture. The major contributions of this chapter include:

- A deep pipelined stream architecture of a particle filter with FO-resampling is proposed.
- Tracking quality of a real-time object tracking system based on the proposed architecture is evaluated and compared with a conventional multinomial resampling approach.
- The performance and resource utilization of FPGA implementation of the proposed architecture are evaluated.
- Comparative discussion on implementation alternatives with improved area efficiency is presented in terms of the hardware amount and power consumption.

The rest of the chapter is organized as follows. Section 5.2 outlines the algorithm of particle filter. The implementation of the system in detail with three steps: prediction, likelihood calculation and parallel resampling are described in Section 5.3. Section 5.4 shows the evaluation results and discussion on the first implementation. Section 5.5 discusses the improvement of the proposed architecture to fit the design in a smaller FPGA. Finally, summary and the scope of the future work are described in Section 5.6.

5.2 Particle Filter Algorithm

5.2.1 Particle Filter

Particle filter is a recursive filter that provides the estimation of non-linear and non-Gaussian processes. Each particle has application-specific states such as coordinates and velocities, and these states are randomly initialized at $t = 0$. Then, particle filter performs three steps: (1) prediction, (2) likelihood calculation and (3) resampling, for each time step as shown in Fig. 5.1.

5.2.1.1 Prediction

Prediction utilizes the previous observations to predict the state of a system in future time instants for each particle. Moreover, the noise is added to the states to adjust the irregular movement. For example, when each particle has a state like its position

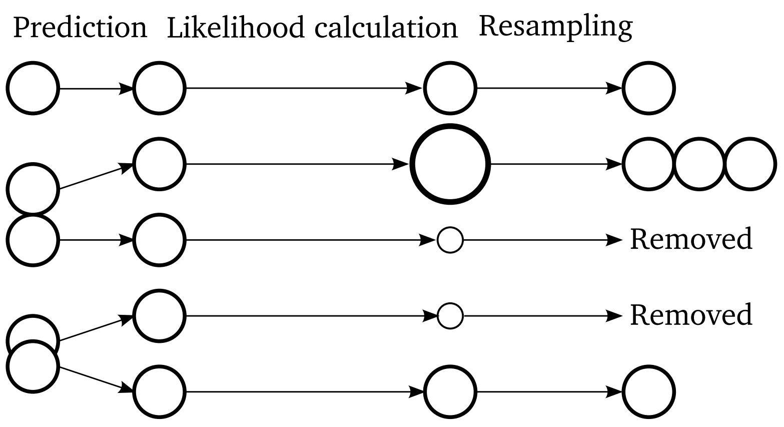


Figure 5.1: Overview of a particle filter

with a 2D coordinate (x, y) and velocities (v_x, v_y), and a weight for uniform linear motion is assumed for a tracking target, this process will be expressed as follows:

$$x_t = x_{t-1} + v_{x_{t-1}} + n_x \tag{5.1}$$

$$y_t = y_{t-1} + v_{y_{t-1}} + n_y \tag{5.2}$$

$$v_{x_t} = v_{x_{t-1}} + n_{vx} \tag{5.3}$$

$$v_{y_t} = v_{y_{t-1}} + n_{vy} \tag{5.4}$$

where x_t and x_{t-1} represent the predicted state of object at time t and $t-1$. v denotes the velocity based on the position of a particle for each time state. According to the noises, n_x and n_y stand for the positioning noises and n_{vx} and n_{vy} are the velocity noises. The initial state of position and velocities are randomly generated. Moreover, the values of noises are also randomly generated.

Fig. 5.2 demonstrates an example of prediction with a robot sample. The blue circles represent the particles and sample robot is surrounded by them. These particles are structured as an x coordinate, a y coordinate and also x, y velocities: four values to comprise a single guess. The prediction step is to have the particles guess where the robot might be moving. Two velocities: v_{x_t} and v_{x_t} is used to move the particles forward or backward and up or down.

5.2.1.2 Likelihood calculation

In likelihood calculation step, weight for the sampled particle is computed as:

$$w_t^{[m]} = p(z_t | x_t^{[m]}) \tag{5.5}$$

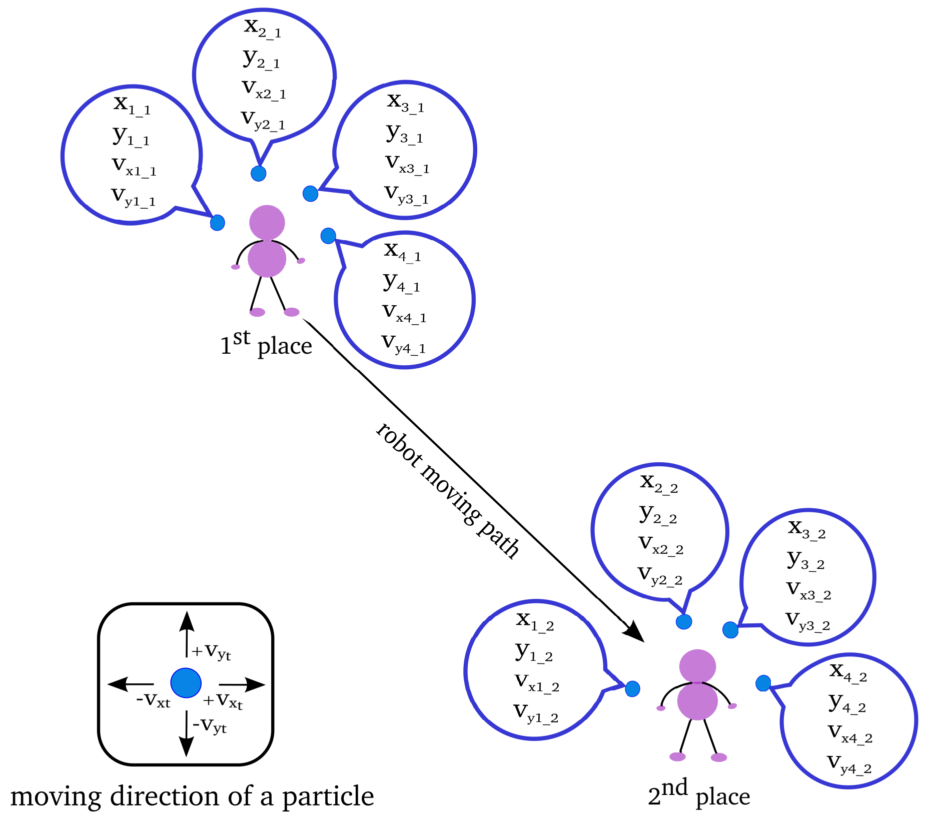

x_{2_1} y_{2_1} v_{x2_1} v_{y2_1}
x_{1_1} y_{1_1} v_{x1_1} v_{y1_1}
x_{3_1} y_{3_1} v_{x3_1} v_{y3_1}
x_{4_1} y_{4_1} v_{x4_1} v_{y4_1}
1st place
robot moving path
x_{2_2} y_{2_2} v_{x2_2} v_{y2_2}
x_{3_2} y_{3_2} v_{x3_2} v_{y3_2}
x_{1_2} y_{1_2} v_{x1_2} v_{y1_2}
x_{4_2} y_{4_2} v_{x4_2} v_{y4_2}
$+v_{yt}$
$-v_{xt}$
$+v_{xt}$
$-v_{yt}$
moving direction of a particle
2nd place

Figure 5.2: Schematic diagram of prediction step

where z_t is current sensor measurement and x_t shows the m-th particle ($1 \leq m \leq M$ and M is the total number of particles).

Fig. 5.2 presents the weights of particles based on the measurement model. For example, the robot has a sensor such as GPS or proximity sensors to range the distance of nearby obstacles. These sensors help the robot determine a good posterior distribution as to where it is. In fact, the closer particle is to the correct position, the more likely will be the set of measurements given that the position. A particle with a larger weight will survive at a higher proportion than a particle with a small weight. Each of these particles will have a specific weight as indicated by the size of blue circles.

5.2.1.3 Resampling

Resampling reduces the variance of particles to be loss of diversity and sample again from the particles until the total number of particles (M) are the same as the previous stage. During resampling step, some particles are replicated in proportion to their weights whereas some particles with small weights are eliminated to remedy for weight degeneration.

Fig. 5.2 shows the schematic diagram of resampling step. In particular, resampling means randomly drawing new particles from the old ones with replacement in

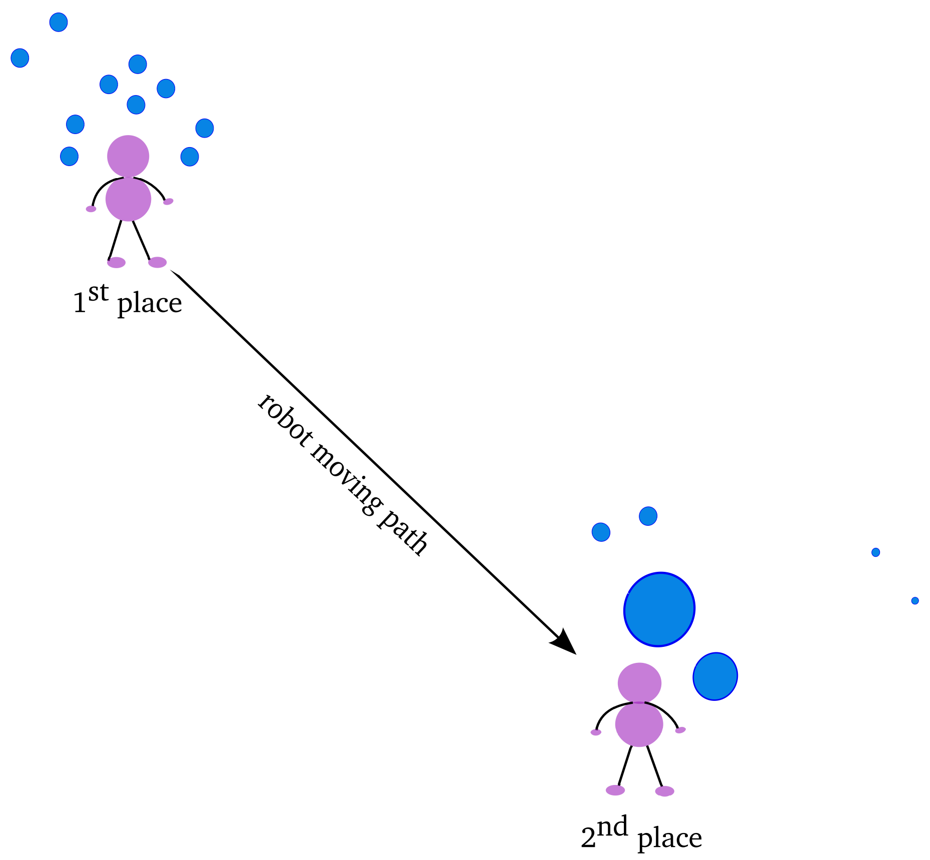


Figure 5.3: Schematic diagram of likelihood calculation step

proportion to their weights. It is important to weight degeneration. Small weight particles far from the sample robot are died out whereas big weight particles near the target are replicated many times. Hence, it can save the computational resources. In essence, resampling gives the better matched position of object, i.e. robot in this example.

5.2.2 FPGA Optimized Resampling (FO-resampling)

The conventional multinomial resampling algorithm draws M particles according to the probability defined by the weights of each particle, so that small weight particles far from target are died out and big weight particles near the target are replicated. This process needs information of all particles, making parallel FPGA implementation difficult. However, FO-resampling method [47] can solve this problem using virtual particles.

Pseudocode for FO-resampling process describes in Algorithm 1 compared with multinomial resampling process as shown in Algorithm 2. B denotes the number of virtual particles. $\hat{x}_{i,n}$ represents a virtual particle and is randomly generated around a real particle x_i. A random number r is generated from a uniform distribution over [-1, 1]. σ_{x_i} measures the spread of $\hat{x}_{i,n}$ around x_i. To the highest possible weight,

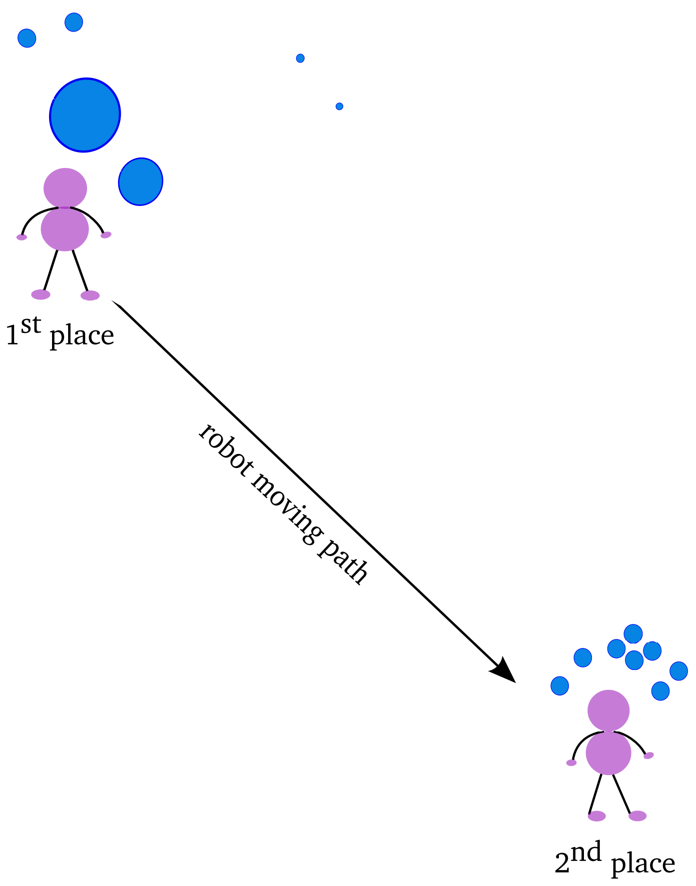


Figure 5.4: Schematic diagram of resampling step

the value of σ_{x_i} varies inversely with the initial weight.

The weight of a virtual particle $\hat{w}_{i,n}$ is obtained from the likelihood calculation of virtual particle with the observation z_t at the location of object $\hat{x}_{i,n}$. If $\hat{w}_{i,n}$ is greater than w_i, x_i and w_i will be replaced by $\hat{x}_{i,n}$ and $\hat{w}_{i,n}$, respectively whereas x_i and w_i will keep the values in the opposite case. After being compared the weights of a real particle and the related B virtual particles, FO-resampling method replaces a particle with the largest weight. Since M is same in a whole process and all particles are not related to each other, parallel resampling architecture enables an efficient chip area usage on FPGAs.

5.3 Implementation

5.3.1 Overview

Object tracking system with FO-resampling using deep-pipelined stream architecture is shown in Fig. 5.5. The whole system is synchronized with the pixel clock of the camera. Pixel data from the camera interface are streamed into the system for 1 pixel per 1 clock cycle and the weights of streamed pixel data are calculated

Algorithm 1 FO-resampling process

```
1: for $i = 1$ to $M$ do
2:     for $n = 1$ to $B$ do
3:         $\hat{x}_{i,n} = x_i + \sigma_{x_i} * r$
4:         $\hat{w}_{i,n} = p(z_t|\hat{x}_{i,n})$
5:         if $\hat{w}_{i,n} > w_i$ then
6:             $x_i = \hat{x}_{i,n}$
7:             $w_i = \hat{w}_{i,n}$
8:         end if
9:     end for
10: end for
```

Algorithm 2 Multinomial resampling process

```
1: for $i = 1$ to $M$ do
2:     if $w_i < r$ then
3:         continue
4:     else
5:         $x_i$ set
6:         $w_i = \frac{w_i}{\sum_i w_i}$
7:     end if
8: end for
```

in a pipelined manner. Compared with the weights of real and virtual particles, the maximum weight of particles are replaced as real particles. As shown in Fig. 5.6, the system is managed by a state machine with four states, which are initialization, prediction of real particles, setting of virtual particles, and weight comparison of both particles (resampling). Focusing on the fact that a video frame consists of a valid pixel region and a synchronization region, weight comparison is processed in the valid pixel region in a pipelined manner, while prediction and setting of virtual particles are processed in the synchronization region.

Fig. 5.7 presents an overview of particle filter system. Firstly, RGB color values from camera are fed into a likelihood module, which calculates and outputs a stream of weights as well as the corresponding coordinate (h_w, v_w). Then, the weights of the likelihoods are compared with M numbers of real particles, each with B numbers of virtual particles. After selecting the maximum weight of each particle, the coordinates and velocities of the selected particles are sent to a weighted center module, where the center of gravity of real particles calculated as an estimated

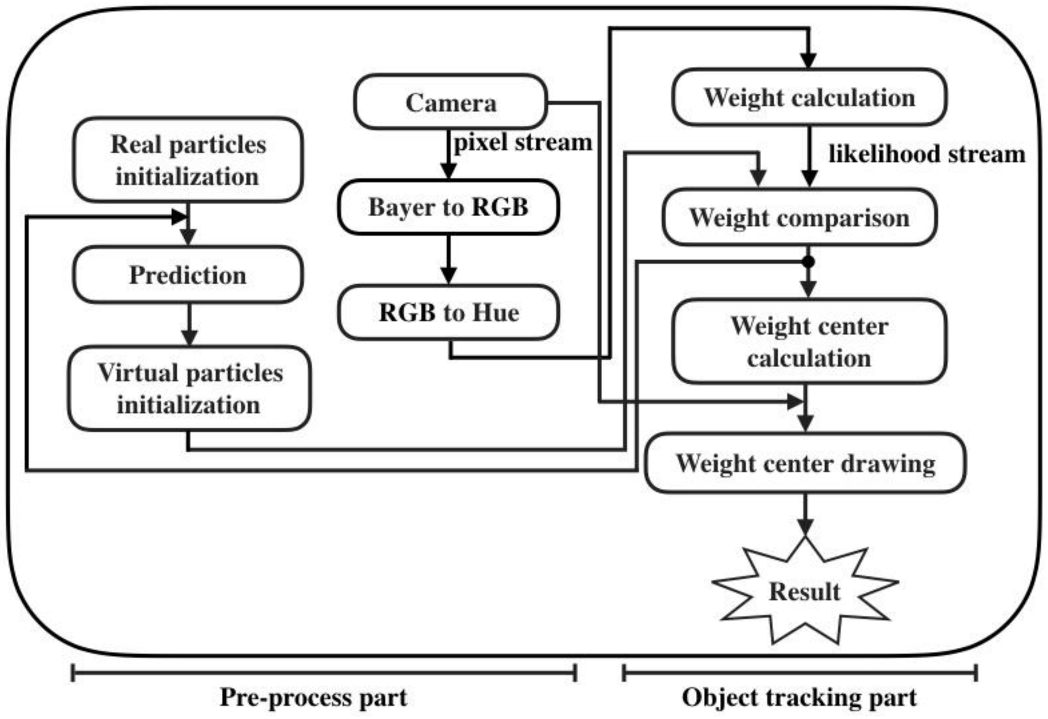


Figure 5.5: Overview of the proposed system in object tracking

position of the tracking target. In parallel with this weighted center calculation, prediction of the next states of the real particles and generation of new virtual particles are performed.

5.3.2 Weight Calculation

The likelihood module takes a stream of RGB pixels as an input, and outputs a stream of weights of likelihood in a pipelined manner. While the calculation process of likelihood may vary depending on tracking targets, a color-based object tracking function is implemented as an example in this experiment. Since RGB representation is not intuitive for color object detection, color space conversion to hue, saturation and intensity (HSI) model is performed [66]. In order to mitigate computational costs and the hardware amount, an integer-based RGB to hue conversion method [67] was adopted. In this method, hue value is calculated as: Hence, the color space conversion entails for real-time image processing. The hue-histogram algorithm for detecting colored objects comprises three steps: (1) converting a RGB color image to a hue image (2) creating a histogram over all image columns of pixels matching the object color (3) finding the maximum position in the column histogram [67]. Although H is an angle of HSI (hue, saturation and intensity) model ranging from $0°$ to $360°$, each coordinate of RGB can vary only from 0 to 255. Thus, a value of H from -1 to 252 are specified in this implementation. According to the comparison

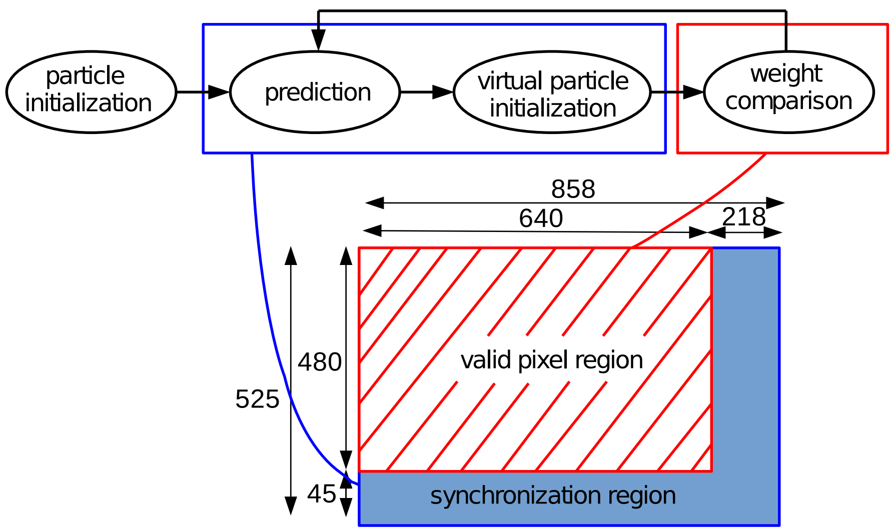


Figure 5.6: State transition diagram of weight comparison step

of three colors in 1 pixel, H is equivalent to the maximum color given by Eq. 5.6. The red color channel has been moved to 42 from 60 and the digital values 126 and 210 are equivalent to 180 and 300 degrees [66]. The difference between maximum and minimum colors is defined as the delta which indicates the gray involvements as shown in Eq. 5.7. The combination equal parts of maximum color (white) and minimum color (black) make the gray color obtained. Similarly, if the three color values are similar or identical, H is defined as NO_HUE (invalid hues) or (-1) to avoid the arbitrary hue values.

$$H = \begin{cases} 42 + \left\lfloor \dfrac{42(G-B)}{\text{delta}} \right\rfloor & \text{if } R = \max(R,G,B), \text{delta} > \dfrac{\max(R,G,B)}{2} \\ 126 + \left\lfloor \dfrac{42(B-R)}{\text{delta}} \right\rfloor & \text{if } G = \max(R,G,B), \text{delta} > \dfrac{\max(R,G,B)}{2} \\ 210 + \left\lfloor \dfrac{42(R-G)}{\text{delta}} \right\rfloor & \text{if } B = \max(R,G,B), \text{delta} > \dfrac{\max(R,G,B)}{2} \\ -1 & \text{otherwise} \end{cases} \tag{5.6}$$

$$\text{delta} = \max(R,G,B) - \min(R,G,B) \tag{5.7}$$

where R, G, B are 8-bit color intensities for red, green and blue, while H is a hue value ranging from -1 to 252. Then, the difference value of hue (H_d) is calculated as follows:

$$H_d = \min(|H - H_t|, 253 - |H - H_t|) \tag{5.8}$$

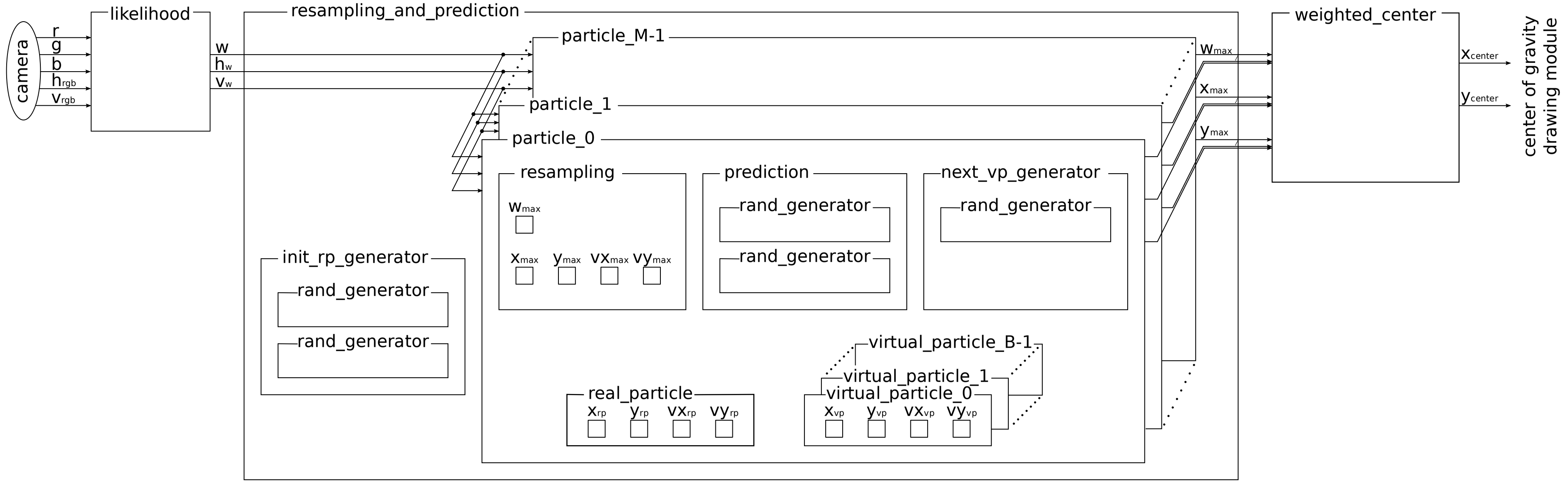


Figure 5.7: Overview of a particle filter module

where H_t shows the hue of the tracked object and was set to 40 for this experiment. In the final step, the weight of the likelihood of the input pixel is calculated as:

$$w = \begin{cases} \left\lfloor \alpha \exp\{-\frac{{H_d}^2}{2s^2}\} \right\rfloor & \text{if } H \neq -1, R \geq 64, \\ \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

where α denotes a parameter related to the scale of weights and s represents a parameter indicating the spread of weight distribution. In this implementation, α and s were set to 1023 and 20, respectively. Since H_d is an 8-bit integer value, the function in Eq. 5.9 can be simply implemented as a table in FPGAs.

5.3.3 Weight Comparison or Resampling

As shown in Fig. 5.8, a resampling module takes weights of likelihood (w) as an input stream, as well as the corresponding coordinates (x_w, y_w). Each resampling module includes one real particle and B virtual particles. The purpose of this module is to find the particle which has the coordinate of the maximum weight in the frame. This particle will become the next real particle in this resampling module.

Each particle compares its coordinate with the input coordinate every clock cycle in parallel. If one of the particles matches and the input weight is larger than the value of the w_{max} register, the register is updated by the input weight, so that the w_{max} register stores the maximum weight after processing all the weights in the frame. When the value of w_{max} is updated, x_{max}, y_{max}, vx_{max}, and vy_{max} are also updated by the states of the matched particle. Logically, the matched particle can send its states to the registers by a bus. However, since wired-OR buses cannot be implemented inside an FPGA chip, this mechanism was implemented as actual OR gates and multiplexers.

Maximum likelihood calculation employs the choice of x_{max}, y_{max} and w_{max} from the comparison results of a real particle and virtual particles in Fig. 5.8. According to the hardware structure, one real particle is rounded by the predefined virtual particles. For example, if the number of real particles ($M = 100$) and virtual particles ($B = 50$) are chosen, one process includes one real particle and fifty virtual particles for one hundred times in parallel. Accordingly, the likelihoods of stream processing are compared to the respective coordinates of one real particle and 0 to 49 virtual particles. When the coordinates of likelihood stream and that of real or virtual particle are same initially, the coordinates and weight become x_{max}, y_{max} and w_{max}. Then, the same process performs as the previous one and the weight of current coordinates are compared with the previous maximum weight.

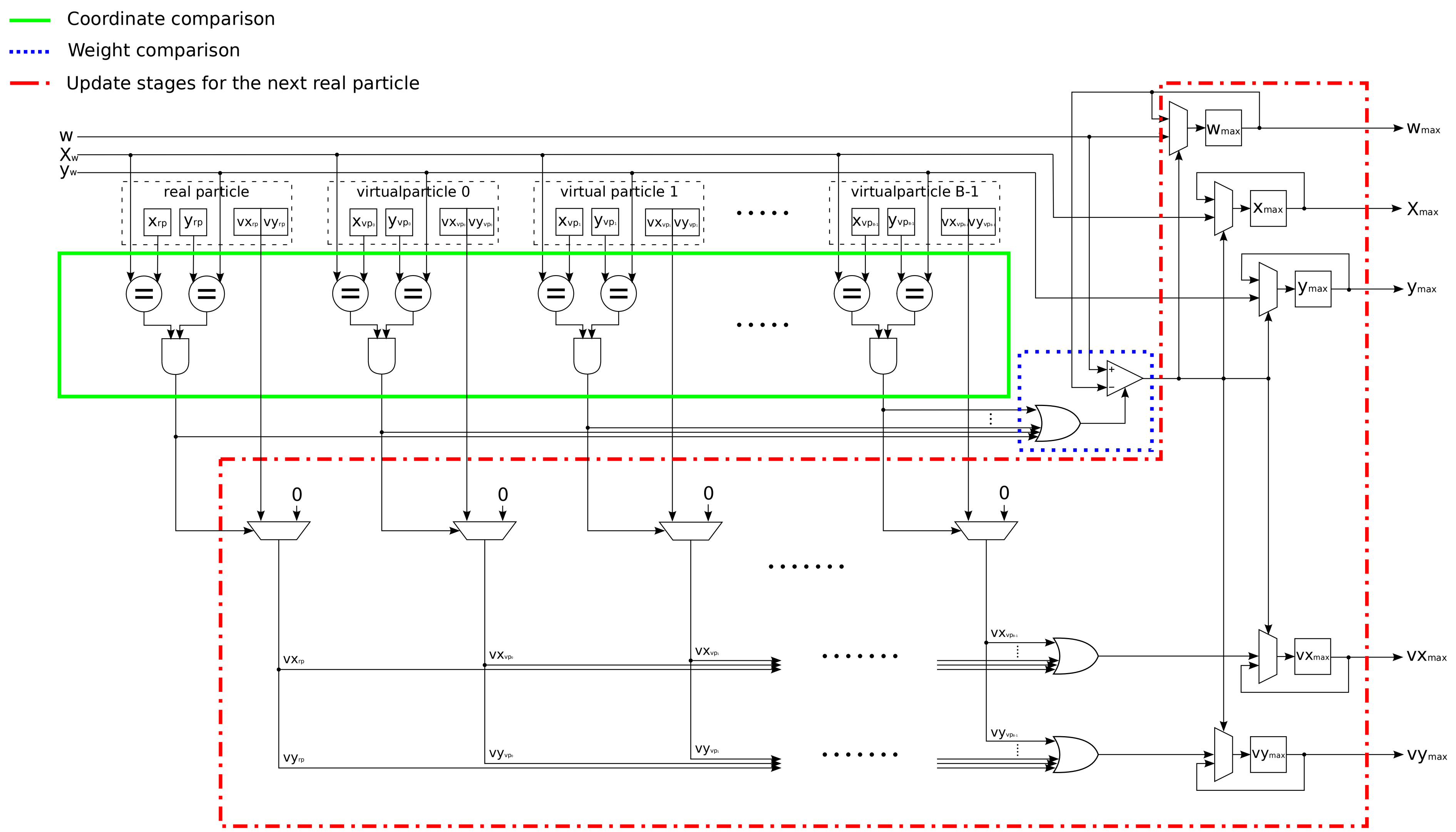


Figure 5.8: Resampling module

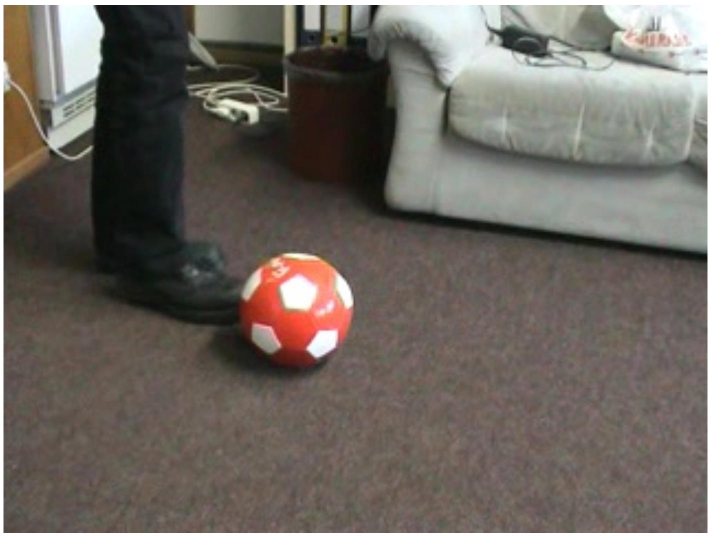


Figure 5.9: A captured image from video

After comparing the weights of one real particle and fifty virtual particles in different positions, the maximum weight value with the related coordinates and velocities of x and y are selected. Since the number of real particle is one hundred, the maximum likelihood, related coordinates and velocities will be one hundred values after the resampling process had finished.

5.3.4 Virtual Particle Arrangement

After selecting the real particles in the resampling modules, the next states of the real particle are predicted according to Eq. 5.1 to Eq. 5.4. Then, new B virtual particles are generated by adding a random number ranging from $-\sigma$ to σ to the states of the real particle, to avoid too big or too small dispersion of virtual particles around one real particle. In this experiment, considering the ease of FPGA implementation σ was defined as:

$$\sigma = \left\lfloor \frac{1023 - w}{16} \right\rfloor \tag{5.10}$$

where the division by 16 can be simply implemented with a shift operation.

5.3.5 Center of Gravity Calculation

To the tracking of a moving model, the center of gravity calculation purposes for estimation of the object position across frames. The center of gravity calculation is given by:

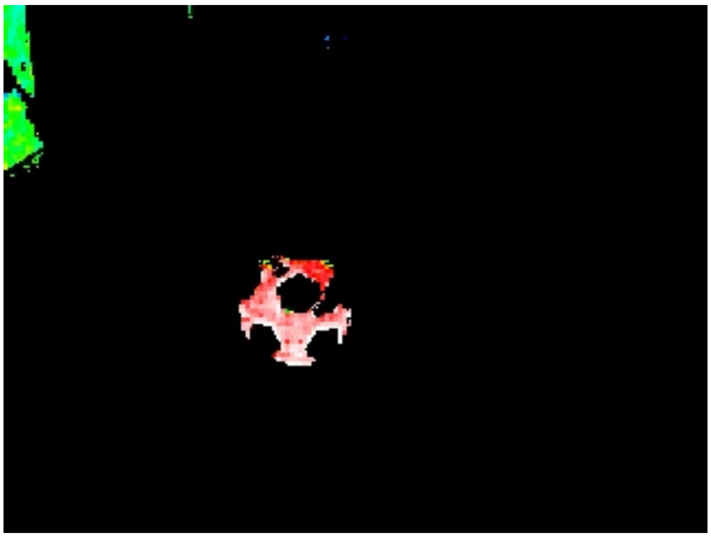


Figure 5.10: Corresponding image of weight representation

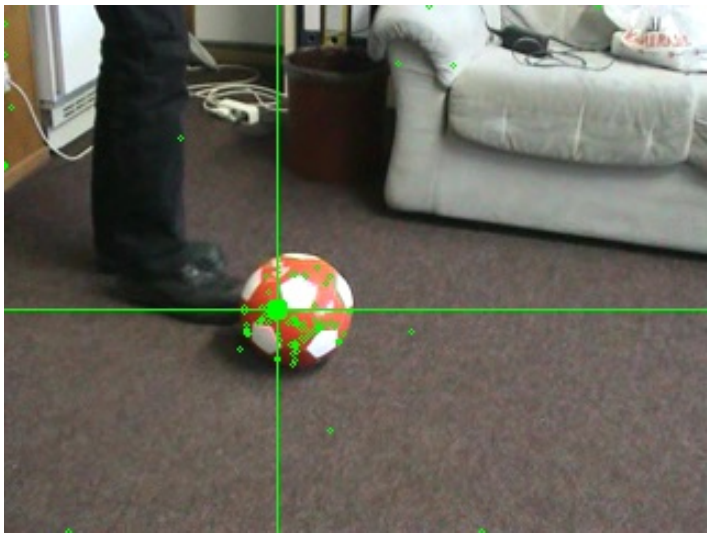


Figure 5.11: Example frame of benchmark video

$$g(x, y) = \left(\frac{\sum_{i=1}^{M} x_i w_i}{\sum_{i=1}^{M} w_i}, \frac{\sum_{i=1}^{M} y_i w_i}{\sum_{i=1}^{M} w_i} \right) \tag{5.11}$$

where x_i, y_i and w_i represent x and y coordinates and weight of the i-th particles, respectively. After the calculation, the center of gravity known as the intersection point of two lines in lime-green color is drawn on the object for tracing the trajectories of the object.

5.3.6 Random Number Generation

A linear feedback shift register (LFSR), which can generate two random numbers for each clock cycle, was implemented. By adding x^{32} in LFSR, the first random generator (rand1) generates the random bits in the range of 31 to 0 and the second one (rand2) selects the numbers from the range 32 through 1. Bitwise XOR operators provide the four feedback taps at 31, 21, 1, and 0-th bit. Thus, the characteristic polynomial is

$$x^{31} + x^{21} + x^{1} + 1. \tag{5.12}$$

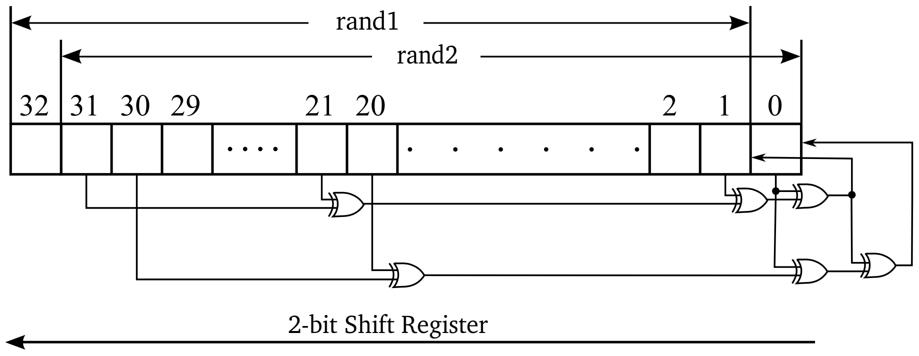


Figure 5.12: Circuit diagram of LFSR

The 33-bit pseudo-random number generator is shown in Fig. 5.12. A two-bit shift operation is performed every clock cycle. Random numbers generated by this module are used for initialization of particles, prediction of next states of particles, and arrangement of virtual particles.

5.4 Evaluation and Discussion

5.4.1 Preliminary Evaluation

At first, the required number of real and virtual particles are evaluated for the FO-resampling method in software before implementing on an FPGA, using an object tracking benchmark video [68] whose tracking target is a red soccer ball. The video-captured image and a snapshot of weight representation are shown in Fig. 5.9 and Fig. 5.10. An example of tracking results is also shown in Fig. 5.11.

The estimation accuracy was evaluated using four metrics: Tracker Detection Rate (TDR), Average Tracking Error (ATE), Maximum Tracking Error (MTE) and Root Mean Square Error (RMSE) given by Eq. 5.13, Eq. 5.14, Eq. 5.15 and Eq. 5.16.

$$\text{TDR} \;=\; \frac{F_t}{F} \tag{5.13}$$

$$\text{ATE} = \frac{1}{F}\sum_{n\,=\,1}^{F}\sqrt{(x_g - x_n)^2 + (y_g - y_n)^2} \tag{5.14}$$

$$\text{MTE} = \max\Big(\sqrt{(x_g - x_n)^2 + (y_g - y_n)^2}\Big) \tag{5.15}$$

$$\text{RMSE} = \sqrt{\frac{1}{F}\sum_{n\,=\,1}^{F}(x_g - x_n)^2 + (y_g - y_n)^2} \tag{5.16}$$

where F and F_t denote the total number of frames and successfully tracked frames in a video. The frame size for benchmark video is 320×240 and F is 602. A tracker is initialized in the first frame of a sequence and tracks the object of interest up to the end. The produced trajectory is then compared to ground truth using a number of measures specified in the particular experiment

Table 5.1: Tracker Detection Rate of each implementation

TDR		No. of real particles: M				
		50	100	500	1000	2000
No. of virtual particles: B	20	0.934	0.935	0.935	0.939	0.942
	40	0.965	1.000	1.000	1.000	1.000
	50	1.000	**1.000**	1.000	1.000	1.000
	60	1.000	1.000	1.000	1.000	1.000
	80	1.000	1.000	1.000	1.000	1.000

The ground truth data indicating the correct coordinate of the target object, which is distributed with the benchmark video, is given as the rectangular area for each frame. If the target object is inside a rectangular area, F_t will increase. ATE measures average discrepancy between centroid of ground truth bounding box and that of tracking system. MTE specifies the maximum tracking error between the center coordinate of each ground truth rectangular (x_g, y_g) and tracking coordinate (x_n, y_n). When M is above 100, the tracker detecting rate and the accuracy errors are not much different as shown in Table. 5.1, Table. 5.2 and Table. 5.3. The number of real particles ($M = 100$) on the number of virtual particles ($B = 50$) is chosen as the appropriate parameters in terms of the reasonable tracking accuracy and

Table 5.2: Average Tracking Error of each implementation

ATE		No. of real particles: M				
		50	100	500	1000	2000
No. of virtual particles: B	20	0.033	0.029	0.027	0.026	0.025
	40	0.023	0.018	0.017	0.016	0.016
	50	0.019	**0.017**	0.015	0.016	0.015
	60	0.020	0.018	0.016	0.016	0.016
	80	0.020	0.019	0.018	0.018	0.018

Table 5.3: Maximum Tracking Error of each implementation

MTE		No. of real particles: M				
		50	100	500	1000	2000
No. of virtual particles: B	20	0.452	0.315	0.387	0.365	0.368
	40	0.155	0.082	0.100	0.083	0.076
	50	0.106	**0.054**	0.048	0.048	0.045
	60	0.061	0.056	0.048	0.047	0.048
	80	0.085	0.063	0.055	0.053	0.052

Table 5.4: FPGA mapping result for the first design (X1_NORMAL)

Resources	Available Resources	X1_NORMAL
LUT	203,800	160,940(78.97%)
FF	407,600	177,229(43.48%)
BRAM	445	0(0.00%)
DSP48E	840	408(48.57%)
Max clock frequency (MHz)		28.00
Available clock cycles in one frame (clock cycles)		450,450
Throughput (FPS)		62.16

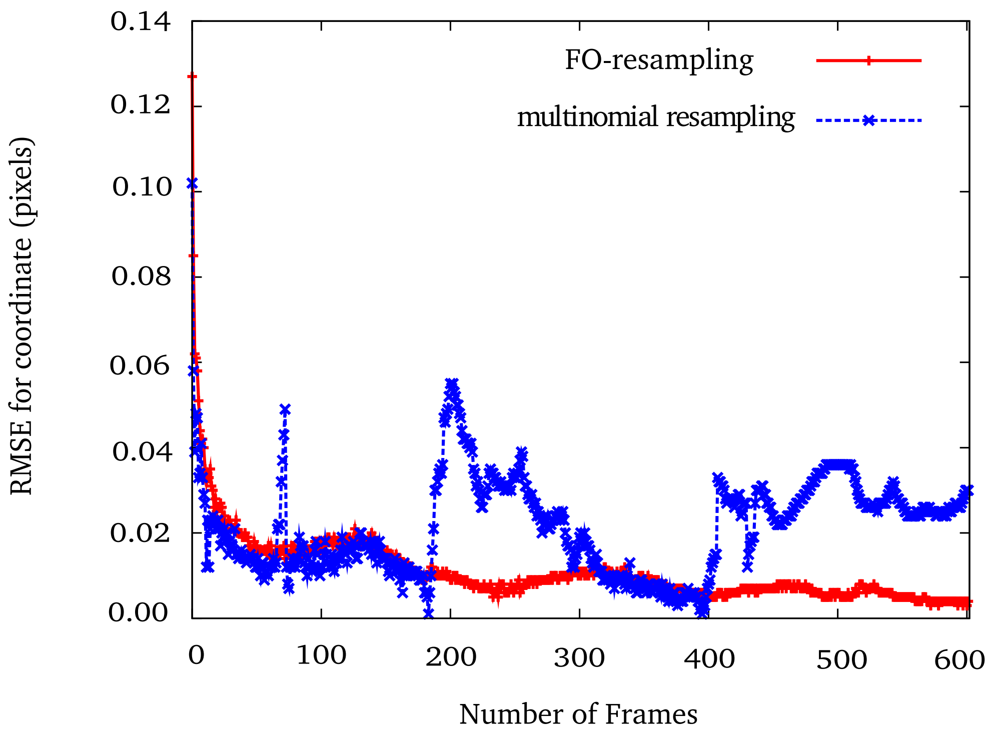


Figure 5.13: RMSE comparison of FO-resampling and multinomial resampling

less amount of resource utilization. Fig. 5.14 illustrates the comparison of tracker detection rate with five types of real particles. Fig. 5.15 and Fig. 5.16 demonstrate the examples of average and maximum tracking errors based on the real and virtual particles. The different color lines represent the five different groups of real particles on the number of virtual particles.

Table. 5.5 shows the accuracy comparisons using FO-resampling ($B = 50$) and multinomial resampling on 602 frames. Obviously, FO-resampling can track successfully in all frames with smaller errors. Fig. 5.13 shows how tracking error changes along the frames with the two resampling methods. While the error with the FO-resampling was decreased along frame goes, the error was sometimes increased with the multinomial resampling, showing the effectiveness of the FO-resampling.

5.4.2 FPGA Mapping

The proposed system is implemented on a Kintex-7 XC7K325T FPGA using Xilinx Vivado 2016.3. According to the results of the preliminary evaluation, M and B were set to 100 and 50, respectively. The constraint for the clock frequency was set to 27 MHz, which is the pixel clock of a Video Graphic Array (VGA: 640×480 pixels) camera with the frame rate of more than 60 FPS. Table. 5.4 illustrates FPGA resources for the first design, namely X1_NORMAL. As shown in the table, the clock constraint was met with the maximum clock frequency. In terms of throughputs, more than 60 FPS was shown for every design, demonstrating the realtime

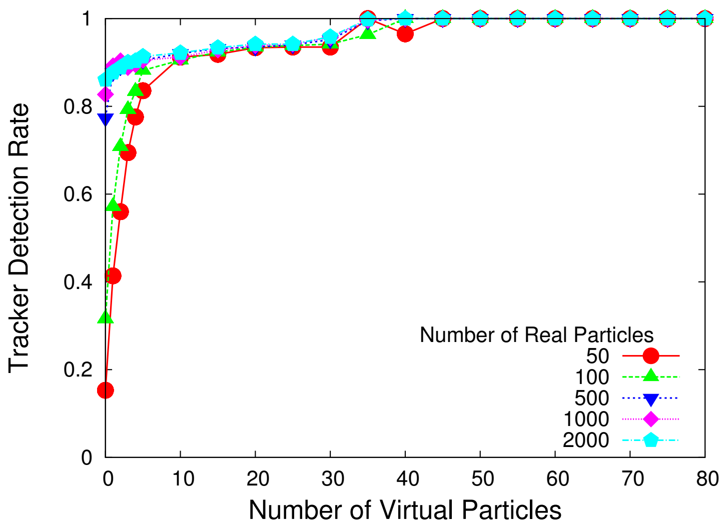


Figure 5.14: Tracker Detection Rate for real and virtual particles

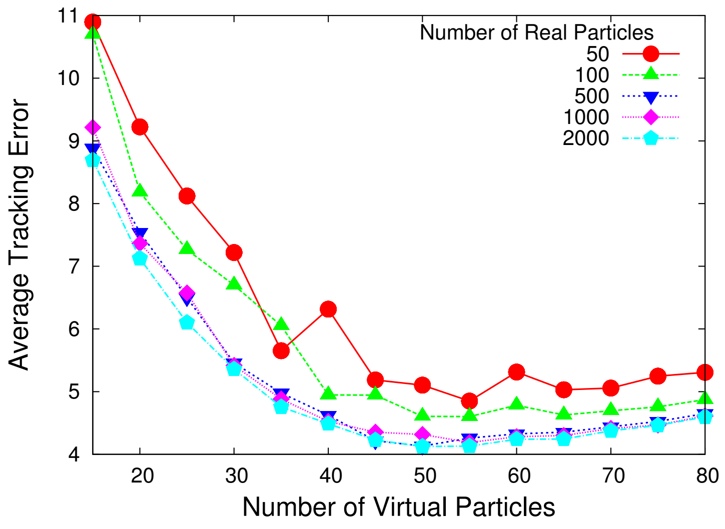


Figure 5.15: Average Tracking Error for real and virtual particles

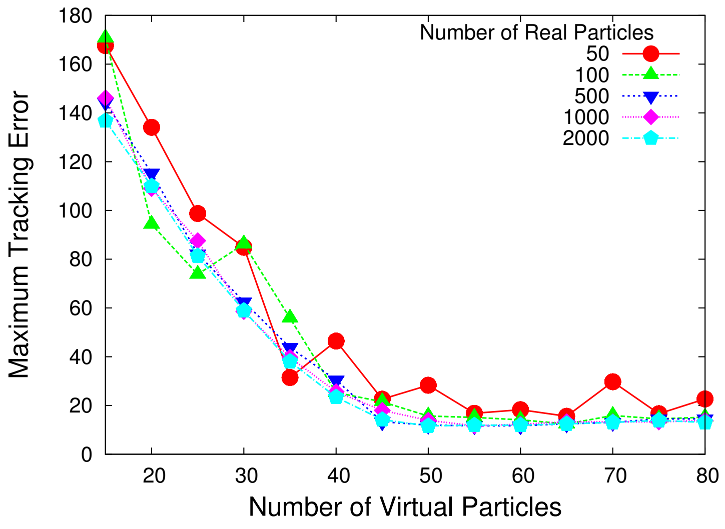


Figure 5.16: Maximum Tracking Error for real and virtual particles

Table 5.5: Accuracy comparison of FO-resampling and Multinomial Resampling

M	Sampling method	F_t	$F - F_t$	TDR	ATE	MTE	RMSE
50	Multinomial	158	444	0.262	0.476	0.864	0.037
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.019**	**0.106**	**0.001**
100	Multinomial	256	346	0.425	0.388	0.910	0.022
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.017**	**0.054**	**0.001**
500	Multinomial	600	2	0.997	0.052	0.383	0.005
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.015**	**0.048**	**0.001**
1000	Multinomial	600	2	0.997	0.051	0.171	0.004
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.016**	**0.048**	**0.001**
2000	Multinomial	602	0	1.000	0.050	0.094	0.004
	FO ($B = 50$)	**602**	**0**	**1.000**	**0.015**	**0.045**	**0.001**

performance for input video images.

When it comes to the latency in clock cycles, latencies for the likelihood calculation ($L_{\text{likelihood}}$), resampling ($L_{\text{resampling}}$), prediction ($L_{\text{prediction}}$), virtual particle generation (L_{virtual}), and weighted center calculation (L_{center}) have to be considered. As aforementioned, the prediction and virtual particle generation are performed in parallel with the weighted center calculation. Thus, the total latency is given as:

$$L_{\text{total}} = L_{\text{likelihood}} + L_{\text{resampling}} + \max(L_{\text{prediction}} + L_{\text{virtual}}, L_{\text{center}}) \tag{5.17}$$

and each latency in this implementation is given as:

$$L_{\text{likelihood}} = 3 \tag{5.18}$$

$$L_{\text{resampling}} = 858 \times 479 + 640 = 411,622 \tag{5.19}$$

$$L_{\text{prediction}} = 1 \tag{5.20}$$

$$L_{\text{virtual}} = B + 1 \tag{5.21}$$

$$L_{\text{center}} = M + 36. \tag{5.22}$$

By assigning Eq. 5.18 $\sim$ Eq. 5.22, $M = 100$, and $B = 50$ to Eq. 5.17, the following equation descirbes the total latency of the proposed design:

$$L_{\text{total}} = 411,625 + \max(B + 2, M + 36) = 411,761. \tag{5.23}$$

On the other hand, as Fig. 5.6 shows, available clock cycles for one frame including the synchronization region is

$$L_{\text{frame}} = 858 \times 525 = 450,450. \tag{5.24}$$

Since $L_{\text{total}} < L_{\text{frame}}$, it has been shown that the implemented particle filter system achieves an in-frame operation.

The proposed design fits in XC7K325T which is a mid-range Kintex-7 device and achieves real-time performance in terms of both throughput and latency. At the same time, the mapping results in Table 5.4 suggest there is a possibility of design improvement for three reasons. Firstly, the resource utilization was not well-balanced especially due to no BRAM utilization. If the balance of resource utilization can be changed, the design may fit in a smaller chip. Secondly, the slack of clock cycles can be taken up in synchronization region. If so, then more tasks can be put into synchronization region to change resource balance. Finally, the required clock frequency of 27 MHz is too slow for typical FPGA design. The FPGA actually may operate much faster, by inserting registers in combinations logic paths. By increasing clock frequency, the hardware amount can be reduced by reusing hardware resources in a time-sharing manner. Section 5.5 discusses the improvement of the proposed architecture to fit the design in a smaller FPGA.

5.5 Improvement of Resource and Time Management

5.5.1 Design Alternatives

In this section, the impact of the resource and time management which is available to improve area and speed on FPGA are mainly highlighted. The following four additional design alternatives are implemented, aiming at reducing the hardware amount.

5.5.1.1 X1_NORMAL_SEP

The first attempt of improvement comes from the separate implementation of modules in X1_NORMAL. In likelihood module, three modules are separated with resource sharing manner to provide the compact design. The first module is used to select the maximum value from two values. Then, the second module compares the third value and maximum value from the first one. In this case, these modules are used for the comparison of red, green and blue pixel values. The third module is used to calculate the minimum distance of hue value given by Eq. (5.8).

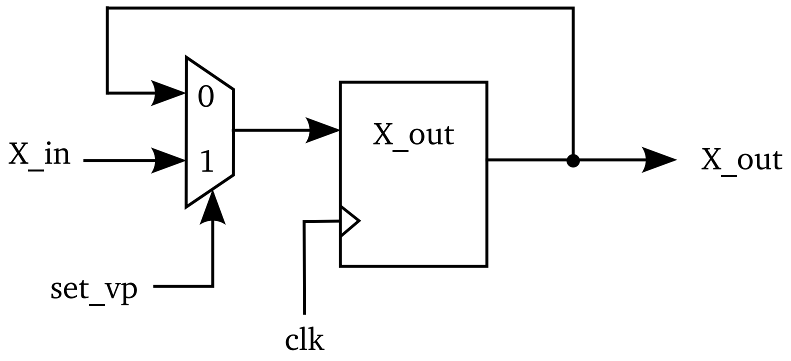


Figure 5.17: A feedback path in virtual particle module

Moreover, the asynchronous reset of particle module in X1_NORMAL is replaced with a synchronous alternative to the gated clock using a data path in terms of area optimizations. As shown in Fig. 5.17, *set_vp* performs one-hot encoding to generate the random pixel coordinates and velocities for virtual particles within the valid pixel region. This design can reduce one-half of the DSP usage compared to the original design.

5.5.1.2 X1_SYNC_V

The improvement of X1_SYNC_V focuses on the resampling module shown in Fig. 5.8. The inefficiency comes from the $(B + 1)$-input OR gates, which are utilized to sent the states of the matched particle to registers. Considering any BRAM resources have not been used and there is a slack of the latency, the task mapping of the resampling is modified. Instead of completing the whole resampling task in the valid pixel region, the resampling task is divided into two sub-tasks: weight comparison and state reading. While the weight comparison is performed in the same manner with the first design, the velocity of the next real particle is sequentially searched and read out in the synchronization region. In this way, large amounts of OR gates can be eliminated as shown in Fig. 5.18. On the other hand, as Fig. 5.19 illustrates, BRAM with a 29-bit width and 50 depth is used to store velocity of the particles. In the weight comparison sub-task, the coordinate of the next real particle is found. After that, using this coordinate as a key, the corresponding velocity is searched on this BRAM. In addition, in order to reduce the logic amount for arithmetic, bit widths for each particle data are optimized.

5.5.1.3 X5_LUT_RAM

This design is based on X1_SYNC_V, but operates at a 5 times higher clock frequency, that is 135 MHz. Thus, a new pixel is given in this design every 5 clock cycles. Comparisons with up to 5 different particles can be performed using the same logic in a time sharing manner. As shown in Fig. 5.20, the states of particles are stored in on-chip memory with a 190-bit width and 5 depth, since only one fifth of the particles are accessed at the same time. This memory is implemented as distributed RAM using LUTs.

5.5.1.4 X5_BRAM

This design is based on X5_LUT_RAM. The difference is that the 5-depth table for particle states is implemented as BRAM, not as distributed RAM unlike the previous design.

5.5.2 Mapping Results

The above four designs are implemented on the Kintex-7 XC7K325T FPGA with 100 real particles and 50 virtual particles. Table. 5.6 summarizes the FPGA mapping results. Compared to the results of the original X1_NORMAL design (Table. 5.4) each new design reduces the utilization of LUTs and FFs while increasing BRAM usage. Especially, as shown in Table. 5.6, the X5_LUT_RAM design was successfully implemented on a smaller FPGA chip (XC7K160T), demonstrating that resource sharing with a high clock frequency leads to the compact architecture and the cost down. Although X5_BRAM achieved the highest resource reduction rates in terms of LUTs and FFs, it consumed approximately 80% of the BRAM offered by XC7K325T. That is why this design cannot be fitted in the smaller chip.

The clock constraints were met for each new design, achieving the throughput of above 60 FPS for VGA frames. The difference in the latency between X1_SYNC_V and X1_NORMAL only exits in the resampling:

$$L_{\text{resampling}} = 411,622 + B \tag{5.25}$$

since the state reading sub-task was added. Thus, the total latency becomes:

$$L_{\text{total}} = 411,811 \tag{5.26}$$

which is also smaller than L_{frame} given by Eq. 5.24.

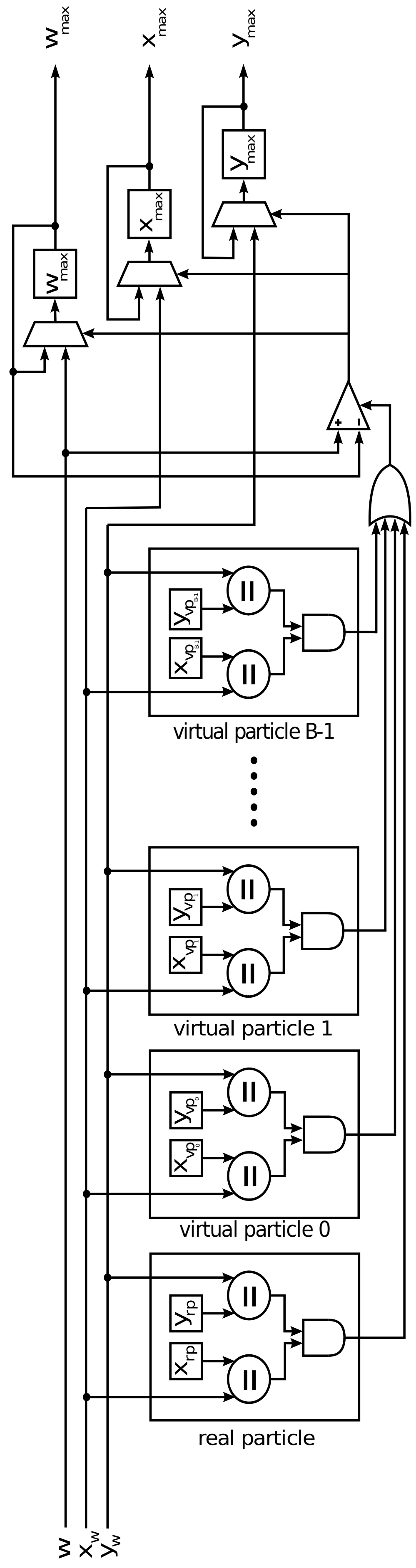


Figure 5.18: Weight comparison module of X1_SYNC_V

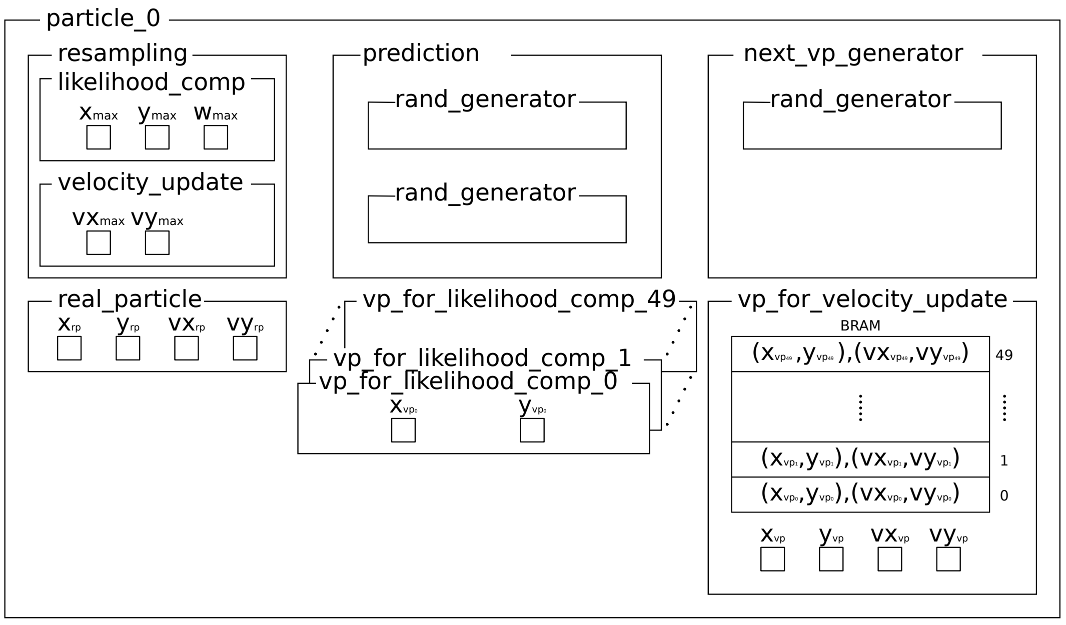

particle_0
resampling
likelihood_comp
x_{max} y_{max} w_{max}
velocity_update
vx_{max} vy_{max}
prediction
rand_generator
rand_generator
next_vp_generator
rand_generator
real_particle
x_{rp} y_{rp} vx_{rp} vy_{rp}
vp_for_likelihood_comp_49
vp_for_likelihood_comp_1
vp_for_likelihood_comp_0
x_{vp_0} y_{vp_0}
vp_for_velocity_update
BRAM
$(x_{vp_{49}},y_{vp_{49}}),(vx_{vp_{49}},vy_{vp_{49}})$ 49
$(x_{vp_1},y_{vp_1}),(vx_{vp_1},vy_{vp_1})$ 1
$(x_{vp_0},y_{vp_0}),(vx_{vp_0},vy_{vp_0})$ 0
x_{vp} y_{vp} vx_{vp} vy_{vp}

Figure 5.19: Overview of a particle filter module for X1_SYNC_V

The total latency (L_{total}) of X1_NORMAL_SEP is the sum of latencies for likelihood calculation ($L_{\text{likelihood}}$), resampling ($L_{\text{resampling}}$), prediction ($L_{\text{prediction}}$), virtual particle generation (L_{virtual}) and weighted center calculation (L_{center}) as shown in Eq. (5.27) ∼ Eq. (5.31).

$$L_{\text{likelihood}} = 5 \tag{5.27}$$

$$L_{\text{resampling}} = (858 \times 479 + 640) + B + 2 \tag{5.28}$$

$$L_{\text{prediction}} + L_{\text{virtual}} = 3 + (B + 5) \tag{5.29}$$

$$L_{\text{center}} = M + 36 \tag{5.30}$$

$$\begin{aligned} L_{\text{total}} &= 5 + 411,674 + \max(3 + (50 + 5), 100 + 36) \\ &= 411,815 \end{aligned} \tag{5.31}$$

For X5_LUT_RAM and X5_BRAM, the latencies were changed as follows.

$$L_{\text{likelihood}} = 7 \tag{5.32}$$

$$L_{\text{resampling}} = (858 \times 479 + 640) \times 5 + B + 2 = 2,058,112 + B \tag{5.33}$$

$$L_{\text{prediction}} = 4 \tag{5.34}$$

$$L_{\text{virtual}} = B + 5 \tag{5.35}$$

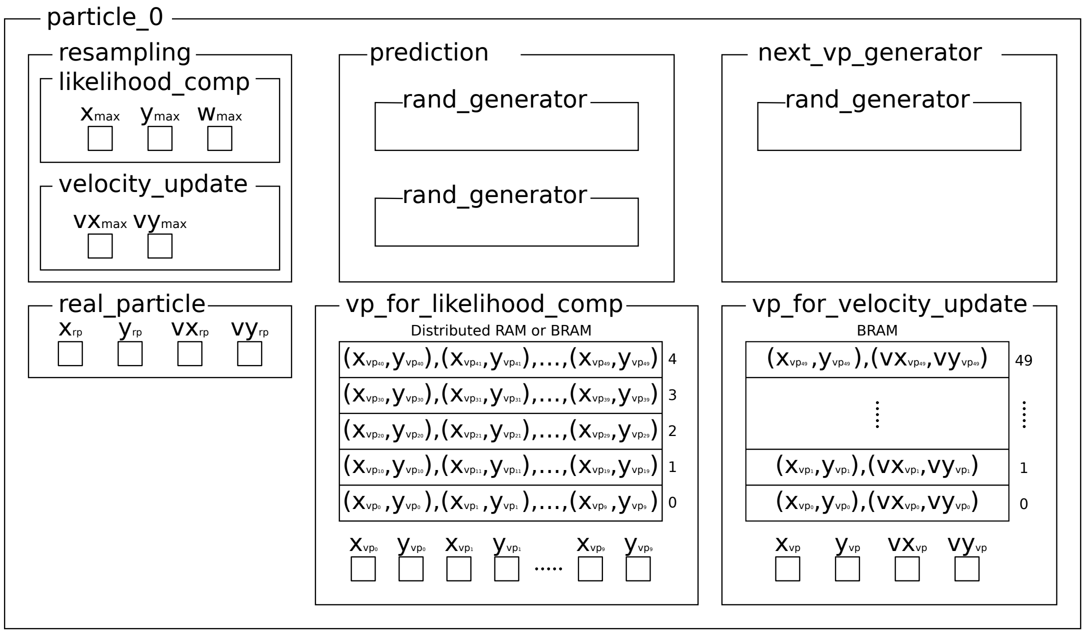

particle_0
resampling
likelihood_comp
x_{max} y_{max} w_{max}
velocity_update
vx_{max} vy_{max}
prediction
rand_generator
rand_generator
next_vp_generator
rand_generator
real_particle
x_{rp} y_{rp} vx_{rp} vy_{rp}
vp_for_likelihood_comp
Distributed RAM or BRAM
$(x_{vp_{40}},y_{vp_{40}}),(x_{vp_{41}},y_{vp_{41}}),\dots,(x_{vp_{49}},y_{vp_{49}})$ 4
$(x_{vp_{30}},y_{vp_{30}}),(x_{vp_{31}},y_{vp_{31}}),\dots,(x_{vp_{39}},y_{vp_{39}})$ 3
$(x_{vp_{20}},y_{vp_{20}}),(x_{vp_{21}},y_{vp_{21}}),\dots,(x_{vp_{29}},y_{vp_{29}})$ 2
$(x_{vp_{10}},y_{vp_{10}}),(x_{vp_{11}},y_{vp_{11}}),\dots,(x_{vp_{19}},y_{vp_{19}})$ 1
$(x_{vp_0},y_{vp_0}),(x_{vp_1},y_{vp_1}),\dots,(x_{vp_9},y_{vp_9})$ 0
x_{vp_0} y_{vp_0} x_{vp_1} y_{vp_1} ..... x_{vp_9} y_{vp_9}
vp_for_velocity_update
BRAM
$(x_{vp_{49}},y_{vp_{49}}),(vx_{vp_{49}},vy_{vp_{49}})$ 49
$(x_{vp_1},y_{vp_1}),(vx_{vp_1},vy_{vp_1})$ 1
$(x_{vp_0},y_{vp_0}),(vx_{vp_0},vy_{vp_0})$ 0
x_{vp} y_{vp} vx_{vp} vy_{vp}

Figure 5.20: Overview of a particle filter module for X5_LUT_RAM and X5_BRAM

$$L_{\text{center}} = M + 37. \tag{5.36}$$

For $M = 100$ and $B = 50$, the total latency becomes:

$$L_{\text{total}} = 2,058,119 + B + \max(B + 9, M + 37) = 2,058,306 \tag{5.37}$$

which is smaller than $L_{\text{frame}} \times 5 = 2,252,250$. Thus, it has been revealed that for each design alternative, realtime performance was achieved in terms of not only throughput but also latencies.

5.5.3 Power Consumption

Particle filter design can be fitted in a smaller FPGA chip, by introducing resource time sharing with a higher clock frequency. Use of a smaller chip will also contribute in reduction of power consumption, while increase in the clock frequency has a negative effect for power reduction. In order to analyze and discuss the tradeoff, power consumption of each design is estimated using Xilinx Vivado tool. Table. 5.7 shows the results. By comparing X5_LUT_RAM on XC7K325T and the same design on XC7K160T, approximately 6% of total on-chip power reduction was shown. However, the power consumption for X5_LUT_RAM on XC7K160T was about 2.5 times higher than the designs synchronized with the camera clock. That is, the increase in power consumption caused by the increase in the clock frequency cannot be compensated by downsizing the chip. In summary, large designs with a slow

Table 5.6: FPGA mapping results for improved designs

Resources	X1_NORMAL_SEP	X1_SYNC_V	X5_LUT_RAM (XC7K325T)	X5_LUT_RAM (XC7K160T)	X5_BRAM
LUT	124,395(61.04%)	102,750(50.42%)	81,895(40.18%)	81,868(80.74%)	69,203(33.96%)
FF	204,494(50.17%)	156,119(38.30%)	106,646(26.16%)	106,646(52.59%)	68,646(16.84%)
BRAM	1.5(0.34%)	51(11.46%)	51(11.46%)	51(15.69%)	351(78.88%)
DSP48E	203(24.17%)	203(24.17%)	203(24.17%)	203(33.83%)	203(24.17%)
Max frequency (MHz)	27.81	33.70	139.38	141.18	141.25
Available clock cycles in one frame (clock cycles)	450,450	450,450	2,252,250	2,252,250	2,252,250
Throughput (FPS)	61.74	74.81	61.88	62.68	62.72

Table 5.7: Power consumption comparison

	X1_NORMAL	X1_NORMAL_SEP	X1_SYNC_V	X5_LUT_RAM (XC7K325T)	X5_LUT_RAM (XC7K160T)	X5_BRAM
Dynamic Power [W]	0.842	0.733	0.772	2.289	2.186	3.530
Static Power [W]	0.165	0.165	0.169	0.179	0.127	0.209
Total on-chip power [W]	1.008	0.898	0.941	2.469	2.314	3.740

clock frequency were more efficient than small designs with a fast clock frequency in terms of power consumption.

5.6 Summary

This chapter proposed efficient FPGA implementation of particle filters with an FO-resampling method. The software-based preliminary evaluation demonstrated that the FO-resampling method can achieve better object tracking quality compared to multinomial resampling by setting an appropriate number of real particles and virtual particles. According to the tradeoff between accuracy and FPGA resources, 100 real particles and 50 virtual particles are chosen as an optimized result based on the accuracy and performance comparison given by Table. 5.5. The implementation experiment on a KC7K325T FPGA revealed that the proposed architecture achieved realtime performance of higher than 60 FPS for VGA images without using any external memory devices, by making best use of a stream processing approach in

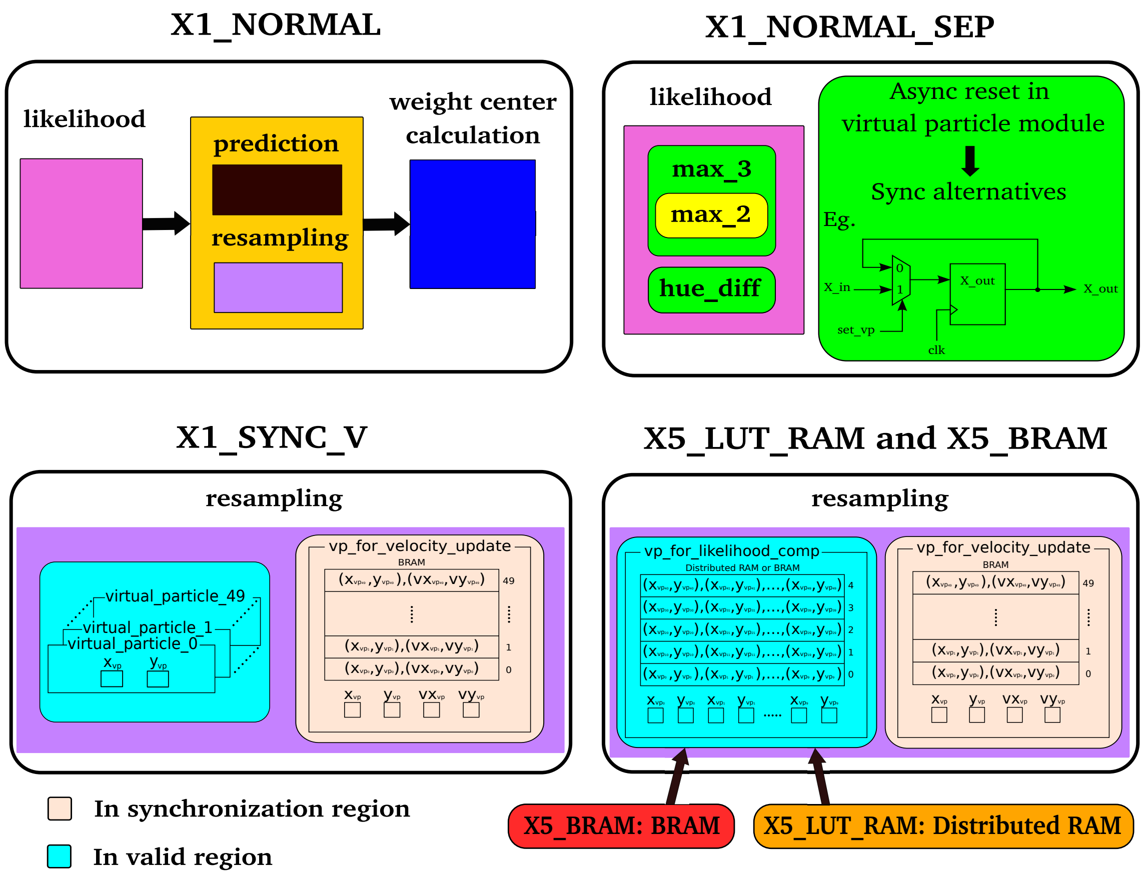


Figure 5.21: Design comparison

a valid pixel region while performing some sequential process in a synchronization region. The accuracy and performance were evaluated using four metrics: tracker detection rate, maximum tracking error, average tracking error and root mean square error. Tracker detection rate in FO-resampling can successfully tracked all frames whereas the multinomial resampling can only track 42.5% with the number of real particles 100. The percentages of average error in two resampling methods based on 100 real particles and 50 virtual particles are FO-resampling approach (4.7%) and multinomial resampling approach (95.3%).

Fig. 5.21 illustrates the design comparison of original design and improved designs. X1_NORMAL design includes three main modules: likelihood, resampling and prediction, and weight center calculation. In X1_NORMAL_SEP design, three modules are separated using resource sharing manner in likelihood module. These three modules are (1) finding the maximum values from two values comparison, (2) finding the maximum values from three values comparison by reusing the two values comparison module, and (3) finding the difference of hue called color appearance parameters. Moreover, the asynchronous reset of particle module in X1_NORMAL is replaced with a synchronous alternative in terms of area optimization. Hence, this design can reduce one-half of the DSP usage compared to the original design with the minimum power consumption. The improvement of X1_SYNC_V focuses on the resampling module. Instead of implementing the whole resampling task in the valid pixel region, the velocity of the next real particle is sequentially searched and read out in the synchronization region. In this way, the large amounts of OR gates can be eliminated. X1_SYNC_V design achieved the maximum throughput 74.81 FPS.

X5_LUT_RAM design is based on X1_SYNC_V, but operates at 135 MHz compared to 27 MHz in the original design. The memory in this design is implemented as distributed RAM using LUTs. In addition, by introducing a higher clock frequency than the pixel clock and by improving a balance of resource utilization, this design can be fitted in a smaller XC7K160T FPGA. The difference between X5_LUT_RAM and X5_BRAM is that the 5-depth table for particle states is implemented as BRAM, not as distributed RAM unlike the previous design. X5_BRAM design utilized the minimum FFs and LUTs usage with the maximum BRAM usage. In addition, it was also shown that the power consumption for a smaller design with a 5 times clock frequency was increased by approximately 4.0 times, compared to a design implemented on a larger chip synchronizing with a slow camera pixel clock. Hence, the proposed system enhances the real-time object tracking system when it comes to the low power, high performance and high accuracy.

Chapter 6

Conclusions

This dissertation deals with the design and implementation for real-time image processing with random sampling. To achieve the high energy efficiency, cost effectiveness and high performance of dedicated hardware, the proposed systems in this dissertation are based on FPGA which owns parallel architectures to perform high speed computing for general-purpose applications. This dissertation presents many evaluation results with the aim of reducing the hardware costs and power consumption. More specifically, the key contributions of this study are fast and low-power FPGA implementation for real-time image processing.

Despite some progress in object fitting and tracking technology on FPGA, not much is known about a deep-pipelined stream architecture of image processing. Therefore, this dissertation focused on stream-oriented process to enhance the computation performance and memory utilization. FPGA-based object fitting system described in Chapter 4 utilized the deep-pipelined stream architecture to access one pixel per one clock cycle intended to get the high throughput and low latency.

Four solver types are proposed in this study because single precision FP for Cramer's rule causes numeric overflow in multiplication steps and long integer version is not appropriate for division step in Gauss-Jordan elimination method. According to the tradeoff analysis between the resource utilization and performance, CRAMER with long integer arithmetics are selected for best approach because it can reduce most hardware resources with acceptable estimation accuracy compared to other arithmetic types.

On the other hand, optimizing the bit width of operands in long integer arithmetic for Cramer's modules reduced the resource utilization on FPGA. More specifically, DSP usage reduced 1% for each new design compared to each original design. Moreover, used resources for FFs and LUTs are less than the original designs. Throughput of circle estimation increases almost 19% of the original design. The proposed system can apply for various geometric shapes including triangle, rectangle, circle, and so on. The three different models with various matrix sizes are selected as examples of real-time object fitting system to easily evaluate the matrix size and performance. For future work, the tradeoffs for other solver algorithms will be analyzed. In addition, the estimation associated with hardware devices will be performed for practical real-time applications such as eye pupil detection, traffic sign detection, mobile robotics, and so on.

As mentioned in Chapter 5, deep-pipelined stream architecture makes best use of on-chip memories with respect to achieving execution efficiency and the proposed systems can operate above 60 FPS for 640×480-pixel images at a pixel clock camera frequency of 27 MHz. Since technological developments of object tracking enhances many computer vision applications, this study focused on the real-time object tracking of non-linear and non-Gaussian systems. Particle filter with stream processing is an effective algorithm for this purpose. In accordance to the object tracking, color-based real-time object tracking is one of the applications of particle filter. The color-based particle filtering requires minimal computation time and is very well suited for embedded vision systems.

In fact, traditional resampling method in particle filtering cannot work in parallel processing because the current particle sets are related to the next particle sets. Therefore, FO-resampling method was chosen to implement the fully parallel particle filter architecture on FPGA for real-time object tracking. In order to compare the performance and accuracy of object tracking with the impact of multinomial resampling and FO-resampling, particle filter with both resampling methods are performed by software in advance. The simulation is done using Intel(R) Xeon(R) CPU E3-1240 v3 @ 3.40GHz and one frame execution time takes 0.00704 seconds. However, X1_NORMAL design takes $411,761$ ns in hardware simulation. Hence, FPGA-based design is about 17 times faster than the software version. On the other hand, the design with higher clock frequency is 3.42 times faster than the software simulation.

When it comes to Tracking Detection Rate, FO-resampling can successfully track all frames. From the error comparison, FO-resampling gives more accurate results than multinomial resampling for color-based object tracking process. Thus, the evaluation result shows that FO-resampling is far superior to multinomial resampling in terms of accuracy and performance. In hardware architecture, the tradeoff between the accuracy and hardware resource usage becomes the most important factor. While the optimal result requires the tradeoff analysis of the amount of resources on FPGAs and required criteria, the number of real particles ($M = 100$) and virtual particles ($B = 50$) is promised as a better solution for the stream-based FPGA implementation using the fully parallel particle filter.

In summary, large designs with a slow clock frequency were more efficient than smaller designs with a fast clock frequency in terms of power consumption. FPGA-based real-time object tracking architecture in this study will be a benefit for endoscope motion estimation, video surveillance, robot localization, etc. The challenging future work includes the evaluation of particle filters with a large number of particles for non-rigid objects and to find the best tradeoff point between chip size and clock frequency. Moreover, real-time object tracking with particle filter algorithm can assist machine learning classifier for FPGA implementation. For example, Histograms of Oriented Gradients (HOG) features are widely used as image features for machine learning [69]. Real Adaptive Boosting (AdaBoost) and a linear Support Vector Machine (SVM) are typically utilized for HOG-based human detection. The particle filter algorithm associated with the machine learning techniques on FPGA is actively addressed in a wide range of applications such as Advanced Driving Assistant Systems (ADAS), surveillance system, robotics and so on.

The combination of both object fitting and object tracking systems is essential for many real-world applications. For example, real-time eye tracking system can improve the security, medical diagnostics, robot navigation, eye control system aiming at people with physical disabilities to communicate with others, etc. Efficient eye-tracking system on FPGA can be divided into four main parts. They are (1) pre-processing part, (2) pupil contour detection part, (3) RANSAC part and (4) particle filter part. The first part will include (1) image data fetching from camera device as a pixel stream, (2) bayer to RGB conversion, (3) RGB to luminance conversion, (4) removing cornel reflection and (5) box blur implementation. Although most of the pre-processing can be straightforwardly implemented on stream-based pipeline architecture, the reflection removal process is relatively complex. Hence, the process will be split into two steps: the determination of envelop pixels and the bilinear interpolation. As a result, dynamic control flows were mitigated and all the pre-processing modules will be implemented on the streamed architecture.

The second part focuses on the extraction of feature points of a pupil contour. Some feature detection algorithms for corner or interest points are Harris Corner Detector, SUSAN corner detector, Shi-Tomasi corner detector, Laplacian of Gaussian corner detector, Hessian corner detector, Features from Accelerated Segment Test (FAST) corner detector, etc. From the real-time image processing point of view, FAST algorithm is fast enough for feature detection [70]. The main functions of FAST algorithm are (1) selecting a base point (P_s) in an image to distinguish whether the pixel is an interest point or not, (2) selecting an appropriate threshold value, and (3) choosing a circle of 16 pixels around the center of a detected corner candidate [71]. In addition, starburst algorithm is designed to find feature points from a base point for non-uniform target distributions. It returns a set of the nearest points which have larger intensity derivative than a threshold on each ray. Moreover, another extraction process will start from firstly extracted features towards the base point to improve the robustness. Therefore, the starburst feature extraction process can be divided into three parts: (1) calculation of intensity derivatives for all the pixels with FAST corner detection, (2) calculation of distances and angles from the center point, and (3) update of the feature point table which has 128 entries.

In RANSAC part, ellipse estimation is more suitable than circle estimation for eye tracking system because the eye pupil area is not modeled by perfect circle with the eye movements. In that case, CRAMER and GAUSS approaches have own benefits and one of them will be chosen depending on the applications. In particular, CRAMER approach coupled with long integer arithmetic brings benefits for embedded and mobile applications such as robotics and unmanned aerial vehicles, where compact circuit size and high energy are preferred. On the other hand, GAUSS approach is far superior to CRAMER approach where precise accuracy is more important than resource utilization and the number of DSP blocks is severely restricted compared to other general resources. After selecting the best parameters with RANSAC algorithms, the center point and estimated ellipse come out. To localize the center of the eye, particle filter algorithm with FO-resampling will estimate the object position across frames using the center of gravity calculation. In this way, the combination method will develop the accuracy and performance of real-time eye tracking system and overcome the constraint of bleary pupil on the system. To summarize, the random sampling with stream-oriented image processing architectures achieve low power and high performance designs.

Bibliography

[1] S. Pllana and F. Xhafa, Programming multi-core and many-core computing systems, Wiley, Jan 2017.

[2] S. Brown and Z. Vranesic, Fundamentals of digital logic with verilog design, 3^{rd} ed., McGraw-Hill Higher Education, Feb. 2013.

[3] Thomas L. Floyd, Digital fundamentals, 11^{st} ed., Pearson, July 2014.

[4] C. Maxfield, The design warrior's guide to FPGAs, Newnes Newton, 2004.

[5] `https://www.blog.digilentinc.com`.

[6] `https://www.coursera.org/learn/intro-fpga-design-embedded-systems`.

[7] E. Stavinov, 100 power tips for FPGA designers, CreateSpace Paramount, 2011.

[8] Xilinx, "Xilinx 7 series FPGA libraries guide for schematic designs v14.1, user guide ug799."

[9] Xilinx, "7 series FPGAs clocking resources v1.13, user guide ug472." `https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf`.

[10] `https://reference.digilentinc.com/_media/reference/programmable-logic/pynq-z1/pynq-rm.pdf`.

[11] `https://www.sparkfun.com/datasheets/Components/General/COM-09622-MAX7219-MAX7221.pdf`.

[12] Xilinx, "Vivado design suite 7 series FPGA libraries guide v2012.2, user guide ug953." `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug953-vivado-7series-libraries.pdf`.

[13] P. Chu, FPGA Prototyping By Verilog Examples: Xilinx Spartan-3 Version, Wiley, 2011.

[14] Xilinx, “AXI reference guide v13.4, user guide ug761.” https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/v13_4/ug761_axi_reference_guide.pdf.

[15] D.W. Hawkins, “Altera JTAG-to-Avalon-MM tutorial v1.0.” https://www.ovro.caltech.edu/~dwh/correlator/pdf/altera_jtag_to_avalon_mm_tutorial.pdf.

[16] Xilinx, “7 series FPGAs PCB design guide v1.13, user guide ug483.” https://www.xilinx.com/support/documentation/user_guides/ug483_7Series_PCB.pdf.

[17] Xilinx, “LVDS source synchronous 7:1 serialization and deserialization using clock multiplication v1.1.1, user guide xapp585.” https://www.xilinx.com/support/documentation/application_notes/xapp585-lvds-source-synch-serdes-clock-multiplication.pdf.

[18] Xilinx, “7 series FPGAs selectio resources v1.9, user guide ug471.” https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf.

[19] Xilinx, “Implementing a TMDS video interface in the Spartan-6 FPGA v1.0, user guide xapp495.” https://www.xilinx.com/support/documentation/application_notes/xapp495_S6TMDS_Video_Interface.pdf.

[20] D.D.W. Group, “Digital visual interface DVI revision 1.0.” http://www.cs.unc.edu/~stc/FAQs/Video/dvi_spec-V1_0.pdf.

[21] Omnivision, “Advanced information preliminary datasheet v1.01.” https://www.voti.nl/docs/OV7670.pdf.

[22] M.A. Fischler and R.C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” Commun. ACM, vol.24, no.6, pp.381–395, June 1981.

[23] S. Martelli, R. Marzotto, A. Colombari, and V. Murino, “FPGA-based robust ellipse estimation for circular road sign detection,” Proc. IEEE 6th IEEE Workshop on Embedded Computer Vision, pp.53–60, June 2010.

[24] W.J. MacLean, “An evaluation of the suitability of FPGAs for embedded vision systems,” 2005 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, pp.131–138, IEEE, 2005.

[25] R.J. Petersen and B.L. Hutchings, “An assesment of the suitability of FPGA-based systems for use in digital signal processing,” 5th Int’l Workshop on Field-Programmable Logic and Applications, pp.293–302, Springer, 1995.

[26] F. Dellaert and S. Tariq, “A multi-camera pose tracker for assisting the visually impaired,” 2005 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, IEEE, 2005.

[27] D. Bekele, M. Teutsch, and T. Schuchert, “Evaluation of binary keypoint descriptors,” 2013 IEEE Int’l Conf. on Image Processing, pp.3652–3656, IEEE, 2013.

[28] T.L. Chao and K.H. Wong, “An efficient FPGA implementation of the Harris Corner feature detector,” 2015 14th IAPR Int’l Conf. on Machine Vision Applications (MVA), pp.89–93, IEEE, 2015.

[29] J. Svab, T. Krajnik, J. Faigl, and L. Preucil, “FPGA based speeded up robust features,” 2009 IEEE Int’l Conf. on Technologies for Practical Robot Applications, pp.35–41, 2009.

[30] D. Bouris, A. Nikitakis, and I. Papaefstathiou, “Fast and efficient FPGA-based feature detection employing the SURF algorithm,” 2010 18th IEEE Annual Int’l Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.3–10, IEEE, 2010.

[31] J. Zhao, S. Zhu, and X. Huang, “Real-time traffic sign detection using SURF features on FPGA,” 2013 IEEE High Performance Extreme Computing Conf. (HPEC), pp.1–6, IEEE, 2013.

[32] F. C. Huang, S. Y. Huang, J. W. Ker, and Y. C. Chen, “High-performance SIFT hardware accelerator for real-time image feature extraction,” IEEE Trans. Circuits Syst. Video Technol., vol.22, no.3, pp.340–351, 2012.

[33] J. Jiang, X. Li, and G. Zhang, “SIFT hardware implementation for real-time image feature extraction,” IEEE Trans. Circuits Syst. Video Technol., vol.24, no.7, pp.1209–1220, 2014.

[34] J. Vourvoulakis, J. Kalomiros, and J. Lygouras, “Fully pipelined FPGA-based architecture for real-time SIFT extraction,” Microprocessors and Microsystems, vol.40, pp.53–73, 2016.

[35] R. de Lima, J. Martinez-Carranza, A. Morales-Reyes, and R. Cumplido, “Accelerating the construction of BRIEF descriptors using an FPGA-based architec-

ture," 2015 Int'l Conf. on ReConFigurable Computing and FPGAs (ReConFig), pp.1–6, IEEE, 2015.

[36] R. Marzotto, P. Zoratti, D. Bagni, A. Colombari, and V. Murino, "A real-time versatile roadway path extraction and tracking on an FPGA platform," Computer Vision and Image Understanding, vol.114, no.11, pp.1164–1179, 2010.

[37] M. Fularz, M. Kraft, A. Schmidt, and A. Kasiński, "FPGA implementation of the robust essential matrix estimation with RANSAC and the 8-point and the 5-point method," in Facing the Multicore-Challenge II, pp.60–71, Springer, 2012.

[38] A. Fijany and F. Hosseini, "Image processing applications on a low power highly parallel SIMD architecture," 2011 IEEE Aerospace Conf., pp.1–12, IEEE, 2011.

[39] B. Tippetts, S. Fowers, K. Lillywhite, D.J. Lee, and J. Archibald, "FPGA implementation of a feature detection and tracking algorithm for real-time applications," Int'l Symposium on Visual Computing, pp.682–691, 2007.

[40] G. Zhou, J. Ye, W. Ren, T. Wang, and Z. Li, "On-board inertial-assisted visual odometer on an embedded system," 2014 IEEE Int'l Conf. on Robotics and Automation (ICRA), pp.2602–2608, IEEE, 2014.

[41] G. Lentaris, I. Stamoulias, D. Soudris, and M. Lourakis, "HW/SW co-design and FPGA acceleration of visual odometry algorithms for rover navigation on Mars," 2015.

[42] K. Dohi, Y. Hatanaka, K. Negi, Y. Shibata, and K. Oguri, "Deep-pipelined FPGA implementation of ellipse estimation for eye tracking," Proc. IEEE 22nd Int'l Conf. Field Programmable Logic and Applications, pp.458–463, 2012.

[43] L.V.G. Katja Nummiaro, Esther Koller-Meier, "An adaptive color-based particle filter," in Image and vision computing, pp.99–110, Elsevier, 2003.

[44] M.S. Arulampalam, S. Maskell, and N. Gordon, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," in IEEE Transactions on Signal Processing, pp.174–188, Feb 2002.

[45] S. Fleck and W. Strasser, "Adaptive probabilistic tracking embedded in a smart camera," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), pp.134–134, IEEE, 2005.

[46] L. Miao, J.J. Zhang, C. Chakrabarti, and A. Papandreou-Suppappola, “A new parallel implementation for particle filters and its application to adaptive waveform design,” Signal Processing Systems (SIPS), 2010 IEEE Workshop on, pp.19–24, IEEE, 2010.

[47] F. Schwiegelshohn, E. Ossovski, and M. Hübner, “A fully parallel particle filter architecture for FPGAs,” in Applied Reconfigurable Computing, pp.91–102, Springer, 2015.

[48] K. Dohi, Y. Yorita, Y. Shibata, and K. Oguri, “Pattern compression of FAST corner detection for efficient hardware implementation,” Proc. IEEE 21st Int. Conf. Field Programmable Logic and Applications, pp.478–481, Sept. 2011.

[49] K. Negi, K. Dohi, Y. Shibata, and K. Oguri, “Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm,” Proc. Int. Conf. Field-Programmable Technology, pp.1–8, Dec. 2011.

[50] Theint Theint Thu, Y. Hayashida, A. Tahara, Y. Shibata and K. Oguri, “Deep-pipelined FPGA implementation of real-time object tracking using a particle filter,” International Journal of Networking and Computing, vol.7, no.2, pp.372–386, July 2017.

[51] A. Tahara, Y. Hayashida, Theint Theint Thu, Y. Shibata, and K. Oguri, “FPGA-based real-time object tracking using a particle filter with stream architecture,” Proc. CANDAR, 2016 4th International Symposium on Computing and Networking, pp.422–428, Sept. 2016.

[52] J.L.H. David A. Patterson, Computer architecture: a quantitative approach, Morgan Kaufmann Publishers Inc., 2011.

[53] `http://www.community.cadence.com`.

[54] H. Matsubayashi, S. Nino, T. Aramaki, Y. Shibata, and K. Oguri, “Retrieving 3-D information with FPGA-based stream processing,” Proc. 16th ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp.261–261, Feb. 2008.

[55] A. Tahara, Y. Hayashida, Theint Theint Thu, Y. Shibata, and K. Oguri, “Power performance analysis of FPGA-based particle filtering for realtime object tracking,” Advances in Intelligent Systems and Computing, vol.611, pp.451–462, July 2017.

[56] Theint Theint Thu, J. Hamamura, R. Soejima, Y. Shibata and K. Oguri, “Comparative evaluation of FPGA implementation alternatives for real-time robust

ellipse estimation based on RANSAC algorithm," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E-100A, no.7, pp.1409–1417, July 2017.

[57] Theint Theint Thu, Y. Shibata and K. Oguri, "FPGA implementation alternatives of robust circle and ellipse estimation based on ransac algorithm," Proc. ICSE, 2015 6th International Conference on Science and Engineering, Yangon, Myanmar, Paper No. McE-01, Dec. 2015.

[58] Theint Theint Thu, A. Tahara, Y. Hayashida, Y. Shibata and K. Oguri, "A parallel resampling method of an FPGA-based particle filter for real-time object tracking," Proc. ICSE, 2016 7th International Conference on Science and Engineering, Yangon, Myanmar, Paper No. McE-01, Dec. 2016.

[59] Theint Theint Thu, J. Hamamura, Y. Shibata and K. Oguri, "A cost effective FPGA implementation of robust circle estimation based on RANSAC algorithm," COOLChips XVIII, 2015 IEEE Symposium on Low-Power and High-Speed Chips, COOLXVIII-P23, Yokohama, Japan, April 2015.

[60] Theint Theint Thu, Y. Hayashida, A. Tahara, Y. Shibata and K. Oguri, "FPGA implementation of a particle filter for stream image processing," Proc. JSST, 2017 The 14th Joint Symposium of Jeju National University and Nagasaki University on Science and Technology, Jeju Island, Korea, May 2017.

[61] `http://www.cse.psu.edu`.

[62] "Streaming SIMD Extensions - Inverse of 4x4 Matrix," tech. rep., Intel Corp., 1999.

[63] A. Athalye, M. Bolic, S. Hong and P. M. Djuric, "Architectures and memory schemes for sampling and resampling in particle filters," Proc. IEEE 11th Digital Signal Processing Workshop, pp.92–96, IEEE, Aug 2004.

[64] A. Athalye, M. Bolic, S. Hong and P. M. Djuric, "Generic hardware architectures for sampling and resampling in particle filters," EURASIP Journal of Applied Signal Processing, pp.2888–2902, 2005.

[65] K. Dohi, Y. Yorita, Y. Shibata, and K. Oguri, "Pattern compression of FAST corner detection for efficient hardware implementation," Proc. IEEE 21st Int. Conf. Field Programmable Logic and Applications, pp.478–481, Sept. 2011.

[66] M. Hanumantharaju, G. Vishalakshi, S. Halvi, and S. Satish, "A novel FPGA based reconfigurable architecture for image color space conversion," in

Global Trends in Information Systems and Software Applications, pp.292–301, Springer Berlin Heidelberg, 2012.

[67] T. Bräunl, Embedded robotics: mobile robot design and applications with embedded systems, Springer Science & Business Media, 2008.

[68] "Bobot-bonn benchmark on tracking." `http://www.iai.uni-bonn.de/~kleind/tracking/index.html`.

[69] M. Oishi, Y. Hayashida, R. Fujita, Y. Shibata, and K. Oguri, "A comparison of machine learning classifiers for FPGA implementation of hog-based human detection," Applied Reconfigurable Computing - 12th International Symposium, ARC 2016, Mangaratiba, RJ, Brazil, March 22-24, 2016, Proceedings, pp.91–104, 2016.

[70] `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html`.

[71] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," IEEE Trans. Pattern Analysis and Machine Intelligence, vol.32, pp.105–119, 2010.