INFORMATION PROCESSING TECHNIQUES, HARDWARE DESIGN AND APPLICATIONS

A Dissertation

by

HONGXIN KONG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2020

Major Subject: Computer Engineering

ABSTRACT

We are living in an information age - information is collected, communicated and computed to improve the society and people's life. This dissertation study is focused on several different aspects of information processing, including error correction in its communication, its application and hardware platform design.

Information systems in automobiles are built upon certain protocols and highly emphasize safety. A variety of information is transmitted inside the vehicle communication system when driving a car and many messages are critical for the vehicle safety. Error handling and real-time are thus two fundamental requirements for in-vehicle network communication. Traditionally, error detection code such as CRC (Cyclic Redundancy Check) is employed and a message is retransmitted if any error is detected. However, message retransmission can conflict with real-time constraints due to extra communication delay and uncertain waiting time. Conventional error correction code can avoid message retransmission, but entails long decoding time or expensive hardware cost. In some recent in-vehicle network protocols, such as FlexRay and Time-Triggered Ethernet, redundant communication channels are supported. We propose a fast yet lightweight error correction scheme that exploits this redundancy, called DUCER (DUal Crc Error coRrection), such that errors can be corrected with low cost while real-time constraints are satisfied.

Smart and precision irrigation is in great demand as freshwater becomes increasingly scarce. Information accuracy is critical to the successful operation of a smart irrigation system. The smart irrigation system in this work consists of a positioning system, a power system and a control system. In its positioning system, Global Positioning System (GPS) is used to track real-time location of the irrigation machine. However, GPS errors are usually non-negligible, which can mislead control systems and then waste water. We developed several information processing techniques to reduce GPS errors and thereby provide more accurate location tracking for the irrigation machine. Moreover, a technique is developed to monitor the operations of the power system so that the overall system reliability is improved.

The rapid progress of neural network technology makes it a popular kernel in many information processing systems, such as image recognition and natural language processing. In dealing with the huge workload of neural network computing, hardware platforms such as FPGA demonstrate advantages over software implementations. In FPGA implementations, placement plays a key role in determining circuit characteristics. Meanwhile, its long computation time is an important factor that makes the flexibility of FPGA computing much less competitive than software compiling. One observation is that neural network circuits can be realized in a regular systolic array. A particular example is systolic array-based neural network circuit design. In this work, FPGA placement exploiting design regularity is studied. For neural network designs with 64×64 systolic arrays, our proposed regularity-aware approach achieves over 25× speedup versus Versatile Place and Route (VPR) with limited circuit performance loss. At the same time, its solution quality is almost perfectly correlated with VPR and thus renders its role for early prototyping.

DEDICATION

To my mother, my father, my grandfather, and my grandmother.

ACKNOWLEDGMENTS

I pay my deep gratitude to my advisor, Dr. Jiang Hu, for his patient guidance and advices. He taught me the ways to solve research problems and helped me to become an independent researcher. I learnt a lot from him, not only academic knowledge but also dedication and motivation to career. The experience I got in my Ph.D. program will benefit me in my entire life. I would like to thank Dr. Dana Janell Osborne, for her advices and help on my research projects. She is always patient and willing to discuss with us whenever we have some difficulties. I would like to thank Dr. Krishna Narayanan for his guidance on my projects. He helped me get my work done faster and better. I would like to thank my committee member, Dr. Tie Liu of Texas A&M University, for his helpful suggestions and comments on my research and dissertation. I would like to thank Dr. Jun Cheng, for his kind help and patient guidance in my first project. Thanks to my mates in Texas A&M University, Wenbin Xu, Jiafan Wang, Hao He, He Zhou, Chaofan Li, Justin Sun, Yanxiang Yang, Rongjian Liang, Jianfeng Song, Hailiang Hu, Yaguang Li and especially Lang Feng. I wish he has a bright future at Nanjing University. Finally, I would like to thank my parents and my wife for their support during my Ph.D. study.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor Jiang Hu (advisor), Professor Dana Janell Osborne from the Department of Biological Agricultural Engineering, Professor Tie Liu and Professor Krishna Narayanan at the Department of Electrical & Computer Engineering.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

The graduate study was partially supported by water seed grant from Texas A&M University.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND MOTIVATION*

Information is the key ingredient for many new technologies, including smart grid, driverless car, artificial intelligence, etc. In almost any systems, information needs to be acquired, communicated, processed, and computed with high efficiency and high reliability. This dissertation research is focused on several different aspects and scenarios of information processing: (1) information processing for efficient and reliable communication in automobile electronics; (2) information processing for smart irrigation systems; and (3) fast hardware layout design for neural network-based information processing systems.

1.1 Information Processing for In-Vehicle Network Communication

In-vehicle network needs to regularly deliver safety-critical information, such as radar data for collision avoidance and engine sensor data. Although channel condition for in-vehicle network is much better than wireless communication, it is still much harsher than usual indoor applications due to more electromagnetic noises, less stable power/ground supply and high ambient temperature caused by operating engine. As such, there is a low, but non-negligible probability of communication error, which is not acceptable for safety-critical applications.

In existing protocols, errors are usually detected by CRC (Cyclic Redundancy Check) and a safety-critical message is retransmitted if any error is detected [6]. Since such retransmission often contends with shared network resources [7] and may experience error again, it results in significant delay uncertainty, which is a well-known adversity to real-time constraints for in-vehicle networks. The technology trend, especially autonomous vehicle, implies that in-vehicle communication demand will continue to grow and increased resource contention will worsen such delay uncertainty. Using Error Correction Code (ECC) can avoid message retransmission and therefore largely mitigate the delay uncertainty issue. However, software execution of ECC, such as Reed Solomon

*©2018 IEEE. Part of this chapter is Reprinted, with permission, from Hongxin Kong, Jun Cheng, Krishna Narayanan, Jiang Hu, "DUCER: a Fast and Lightweight Error Correction Scheme for In-Vehicle Network Communication", IEEE International Conference on Vehicular Electronics and Safety, 09/2018.

code, takes long runtime while its hardware implementation costs large circuit area. This is the reason why currently no ECC is used for in-vehicle communication in vehicle communication network and indicates the need for a lightweight error correction scheme.

We propose a new error correction scheme that can largely avoid message retransmission, does not change encoding of existing protocols and has low overhead on decoding time and circuit area. It is based on the observation that recent in-vehicle network protocols, such as FlexRay [8] and Time-Triggered Ethernet (TTE) [9], support redundant communication channels. The key idea is to make use of the redundancy along with CRC such that most detected errors can be corrected. The proposed error correction scheme, called DUCER (DUal Crc Error coRrection), is implemented in both software and FPGA circuit. Experimental results show that it can correct 5-bit errors for 254-byte payload within $1ms$. It can correct 99.9998% of the errors detected by CRC. Compared to Read Solomon code, its software is one order of magnitude faster and its circuit implementation consumes 89% less area. Moreover, DUCER does not cause any change for the encoding of existing protocols.

1.2 Information Processing Application in Smart Irrigation

Fresh water that we use in our daily life is limited and in increasing demand due to the growth of population. Agricultural irrigation is one of the major consumer of fresh water resources and therefore plays an important role on potential water savings for our future life. How to improve the water use efficiency is an important subject that has attracted a lot of research attention. Smart irrigation guided by machine learning is a promising direction on solving this problem.

A modern irrigation machine is often a moving system. Smart irrigation intends to precisely control the amount of water irrigated at individual locations. The precise control requires accurately monitoring the location and moving speed of an irrigation machine. The precision of common GPS is not high enough, and therefore the following techniques are studied for improving the positioning precision of an irrigation machine.

- Timing based Three-point Localization method (TTPL)

- Received Signal Strength Indicator (RSSI) based Three-point Localization method (RTPL)
- U-blox [2] GPS with Gaussian Filter [10]

Based on our experiments, U-blox GPS with Gaussian filter delivers the best accuracy. The average error can finally reduce from 5 meters to 0.7 meters.

A moving irrigation machine is usually propelled by an electric motor, whose input is three-phase 480V power supply. In order to ensure the system reliability, we developed a technique for monitoring the supply voltages, which can send alarm signals whenever the supply voltage level is out of normal range.

1.3 Fast Placement for FPGA-Based Information Processing Hardware

Neural network is the dominant approach to a category of information processing problems: classification and regression. Due to its huge computation workload, there is a trend for its hardware acceleration [11]. A popular hardware neural network architecture is systolic array due to the advantage that it can avoid many global wires. However, so far there is no EDA tool dedicated to the layout of systolic arrays. The layout of a large systolic array based neural network circuit may take hours or even days.

Field Programmable Gate Arrays (FPGA) is an increasingly popular computing platform due to its unique trade off between computing efficiency and application flexibility. In general, FPGAs are more efficient than software computing on general purpose microprocessors as the overhead associated with instructions is avoided. Compared to Application-Specific Integrated Circuit (ASIC) circuits that are fixed to specific applications, FPGA circuits provide better flexibility because of their reconfigurability. One example of large-scale industrial adoption is FPGA-based accelerators at Microsoft datacenters [12]. However, FPGA circuit design time is still nowhere near software programming. There are two primary reasons. First, FPGA design is rarely a single-shot process although automatic synthesis is available. There are numerous options for high level decisions, such as loop unrolling factor and number of pipelining stages, whose effects need to be carefully assessed. Actually, this is a main subject for design space exploration [13]. Second, the effect of a high level decision depends on subsequent physical design and thus layout aware high level synthesis is motivated [14]. However, physical design is perhaps the most time consuming part among all FPGA design steps. For instance, placement of a Convolutional Neural Network (CNN) circuit by Versatile Place and Route (VPR) [15, 16], which is the dominating academic tool, may take over 10 hours. High level models [17] can help prune out some inferior solutions. Nonetheless, several trials of physical design are still needed to achieve design closure. Although circuit performance is typically a primary design goal, the large variety of FPGA users includes those who accept mediocre designs in exchange for fast deployment [18].

In this dissertation research, we investigate fast FPGA placement for designs with regularity. General FPGA placement has been studied for many years [15, 16, 19–28] while there has been little attention [29] to designs with regularity. The regularities of the few existing works are fine-grained [29] or at chip die level [30,31], which are easily obtained in a copy-and-paste manner. We consider middle-grained and relative general cases, where interactions among intra-PE (Processing Element), inter-PE and non-PE connections are simultaneously addressed.

Is it worthwhile to develop FPGA placement tools dedicated to designs with regularity? The answer depends on answers to two sub-questions: (1) is the practical need large enough? (2) is the benefit from regularity large enough? The wide applications of designs with regularity [32–39] indicate an arguably positive answer to the first sub-question. For instance, in FPGA computing, a rapidly growing application is neural network, which is often implemented in a very regular way such as systolic array [32, 37–39]. The structural regularity of systolic arrays makes it natural for systolic array placement to also have spatial regularity. Moreover, regularity can also facilitates routability. When local routability in one PE of an array is achieved, which is relatively easy, the global routability of an entire array is also ensured due to the spatial regularity. More broadly, systolic array-based FPGA designs have been widely used in many other important applications such as matrix multiplication [34], image processing [33] and digital signal processing [35]. As such, an important goal of this work is to find an answer to the second sub-question.

In the past, FPGA placement techniques were used to be dominated by simulated annealing [15, 16, 19, 21, 24] with VPR [15, 16] being its famous representative. The recent trend is analytical placement [20,22,23,25–27] due to its capability of handling large designs. Nonetheless, simulated annealing and other probabilistic meta-heuristics [31] are still valuable due to their flexibility and easiness of parallelization [21, 24]. Therefore, they are still being used in recent research [28, 31] and serve as a key component in a commercial FPGA placement tool [40].

The regularity-aware placement in this work is a hybrid approach of Integer Linear Programming (ILP) and simulated annealing. In general, ILP is difficult to scale for large problems. However, regularity allows a significant reduction of decision variables and the ILP here is applied to only the regular portion of a design. In fact, the complexity of the ILP is independent of the number of Processing Elements (PEs) in a given PE array. The regularity allows speedup techniques for the simulated annealing part as well. The benefit of fast computation speed is mostly embodied for large designs, where computing acceleration is the most needed. For neural network designs with 64×64 systolic arrays, the proposed approach achieves over 25× speedup compared to VPR. The expense is no more than 10% degradation on post-routing wirelength and circuit frequency. Partly due to the regularity, the placement solutions are generally routable and routing completion is easily obtained for million-LUT designs. At the same time, the regular placement solution quality is almost perfectly correlated with VPR solutions. These results imply two possible application scenarios for the regularity-aware placement.

1. Applications where time to market is very tight while design specifications are not aggressive. Then, the regular placement can be directly adopted in final products.

2. Fast prototyping [41] for assessing the impact of various “what-if" options is achieved by the regularity-aware placement while the final placement solution is obtained from another tool.

The contributions of this work are summarized as follows.

- To the best of our knowledge, this is the first *algorithmic* study (instead of copy-and-paste implementation) of FPGA placement for systolic array considering regularity, which is a design style with increasing popularity.

- The study result shows that regularity allows over $25\times$ speedup for large neural network designs.
- An ILP approach to PE array placement is developed, including techniques for transforming nonlinear constraints to linear ones.
- Multiple regularity-enabled techniques are proposed for accelerating the simulated annealing in FPGA placement.
- The price paid for regularity is assessed and found to be no more than 10% degradation on wirelength and frequency.
- A nearly perfect correlation is identified between the solution quality of regularity-aware placement and that of a classical approach.

2. INFORMATION PROCESSING FOR IN-VEHICLE NETWORK COMMUNICATION*

In this chapter, background on Cyclic Redundancy Code (CRC) and Reed Solomon code are briefly reviewed. Our proposed error correction scheme is described in detail and the error correction capability of our scheme is then evaluated. The design on circuit implementation of our error correction scheme is explained. Experimental results are provided at the end.

2.1 Background

2.1.1 In-Vehicle Network Channel Condition and Error Handling

In practice, a mainstream in-vehicle network protocol is CAN (Controller Area Network). Its Bit Error Rate (BER) is discussed in [42] for different environments. It is 3×10^{-11} in a benign environment, 3.1×10^{-9} in a normal environment and 2.6×10^{-7} in very noisy environment. The probability of 2-bit or more errors is from 8.33×10^{-15} (benign environment) to 1.85×10^{-7} (very noisy environment). The frequency of error occurrence is not acceptable in a high speed communication channel. As a result, significant amount of time will be spent on re-transmission, which conflicts with the real-time constraints in safety-critical systems. In evaluating our error correction scheme, we consider the near-worst case and base our analysis on channel condition of BER 10^{-6}.

In CAN bus, errors are detected by CRC. Once an error is detected for an message in safety-critical applications, the message is retransmitted. Due to the low bandwidth of CAN and continuously increasing communication demand, people developed alternative protocols such as FlexRay [8] and TTE [9]. Besides providing high-bandwidth, many of these alternatives support two communication channels [43] as shown in Figure 2.1. Both channels can receive messages after CRC encoder and send the message to the same receiver. The communication can operate in two modes: (i) independent mode, where the two channels independently send different messages; (ii)

*©2018 IEEE. Reprinted, with permission, from Hongxin Kong, Jun Cheng, Krishna Narayanan, Jiang Hu, "DUCER: a Fast and Lightweight Error Correction Scheme for In-Vehicle Network Communication", IEEE International Conference on Vehicular Electronics and Safety, 09/2018.

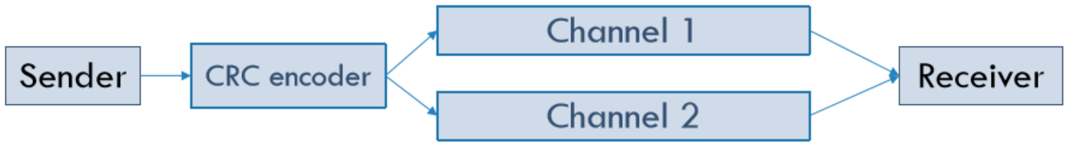


Figure 2.1: Protocol structure with two communication channels

fault-tolerant mode, where the two channels send the same message at the same time. Evidently, the independent mode provides a high communication bandwidth while the fault-tolerant mode is preferred in safety-critical applications. Although independent mode can make better use of the bandwidth, fault-tolerant mode is more preferred in the safety-critical applications in order to increase the error detection capability of the whole system.

2.1.2 Error Detection by CRC

CRC is an error detection code commonly used in in-vehicle networks. A CRC code is characterized by its generator polynomial $G(x)$. If the original message is a binary bit-stream $M = m_n m_{n-1} ... m_0$, its corresponding polynomial is $M(x) = m_n x^n + m_{n-1} x^{n-1} ... m_0$. At encoding with k-bit CRC, polynomial division $M(x) \cdot x^k \ mod \ G(x)$ is performed and the remainder $R(x)$ is appended to the original message to form a new message with polynomial $S(x) = m_n x^{n+k} + m_{n-1} x^{n+k-1} ... m_0 x^k + R(x)$. The new message is transmitted through a communication channel. The polynomial of received message $T(x)$ might be different from $S(x)$ due to possible communication errors. The decoder at receiver performs polynomial division $T(x) \ mod \ G(x)$. If the remainder of this division is not 0, then an error is detected.

Example: We use this as a running example in this chapter. A 4-bit payload message is “100” and its corresponding polynomial is $M(x) = x^3$. The CRC generator polynomial is $G(x) = x^3 + x + 1$. CRC encoding starts with performing $M(x) \cdot x^3 \ mod \ G(x)$ to obtain remainder $x^2 + 1$, which is appended to $M(x)$ to form a new message “1000101” ($S(x) = x^6 + x^2 + 1$). During transmission, due to an error, $S(x)$ is changed into “110010” ($T(x) = x^6 + x^5 + x^2 + 1$). At receiver, decoding process $T(x) \ mod \ G(x)$ is performed. The remainder is not 0 and the error is detected.

The error detection and correction capability of a code can be indicated by its Hamming distance. The Hamming distance (D) is used to define some essential notions in coding theory, such as error detecting and error correcting codes. Hamming distance in this chapter denotes the maximum number of errors that can be detected in decoding process. The larger D is, the better is the capability of the code to detect errors. In FlexRay, a message consists of 254-byte payload and 24-bit CRC. The Hamming distance of single such channel is 4 [44], and becomes 8 for dual-channel in fault tolerant mode. TTE employs 32-bit CRC with polynomial the same as the IEEE 802.3 (Ethernet) network standard [45]. Other polynomials of degree 32 are proposed in [46] to further increase Hamming distance.

Residual error probability P_{rs} is another indicator of error detection capability of a code. It is the probability that errors are not detected by the code [47]. It depends on message length, BER and the generator polynomial of CRC. According to [47], for a n-bit message, Hamming distance H, the degree of generator polynomial k and BER b, an upper bound of P_{rs} is:

$$\bar{P}_{rs} = \sum_{i=H}^{n} \binom{n}{i} \cdot b^i \cdot (1-b)^{n-i} \tag{2.1}$$

and a lower bound is

$$\underline{P}_{rs} = 2^{-k} \cdot \sum_{i=H}^{n} \binom{n}{i} \cdot b^i \cdot (1-b)^{n-i} \tag{2.2}$$

By applying the two equations, we obtain the upper bound and lower bound of P_{rs} for a 254 bytes payload message when BER equals 10^{-6} and 10^{-7}, as shown in Table 2.1 and Table 2.2, respectively.

Table 2.1: Bounds of P_{rs} when BER=10^{-6}

	Upper bound	Lower bound
FlexRay	7.07×10^{-13}	4.21×10^{-20}
TTE	7.08×10^{-16}	6.68×10^{-26}

According to Table 2.1 and Table 2.2, P_{rs} of TTE is smaller than that of FlexRay under the same BER. When BER is equal to 10^{-6}, the upper bound for FlexRay is 7.07×10^{-13}, which means

Table 2.2: Bounds of P_{rs} when BER=10^{-7}

	Upper bound	Lower bound
FlexRay	7.08×10^{-17}	4.22×10^{-24}
TTE	7.08×10^{-21}	6.68×10^{-31}

there is at most 1 bit undetected error among 7.07×10^{13} transmitted bits. If the data transmission rate is $10Mbit/s$, this residual probability implies that there is at most 1 bit undetected error for approximately 81 days of continuous data transmission.

2.1.3 Message Retransmission Time

We briefly review message retransmission time of FlexRay according to [7]. The time slot allocation of one retransmission period T is depicted in Figure 2.2. In FlexRay, static segments and dynamic segments are transmitted in static slots and minislots (MSs), respectively. A Prompt Response (PR) node is to detect errors in safety critical messages and build the PR message to controller unit. After transmission of all static slots, the PR node builds and broadcasts PR message, which contains the information about the messages to be retransmitted. The sender will retransmit the safety critical message in RE-SC (REtransmission Safety-Critical) slots.

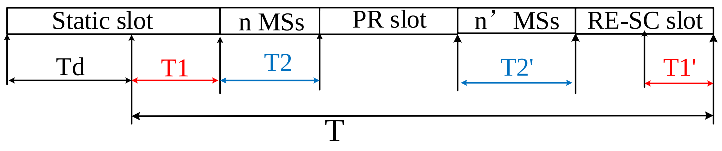


Figure 2.2: Time slot allocation in one retransmission period for FlexRay.

T_d is message transmission and propagation delay. T_1 consists of the error detection time T_{detect} ($T_{detect} < T_1$) and channel idle time. T_2 is the time for minislots, which include the time $T_{build} < T_2$ for a PR node to build PR messages. Then, PR slots send PR messages to controller unit and broadcast PR messages. RE-SC messages are prepared in T_2'. The time to prepare RE-SC messages is $T'_{build} < T'_2$. The error detection time T_{detect} for RE-SC message is in time T'_1 and $T_{detect} < T'_1$. Overall, we reach the following inequalities:

$$T > T_1 + T'_1 + T_2 + T'_2 > 2T_{detect} + T_{build} + T'_{build} \tag{2.3}$$

According to [7], in the worst case, T_{build} is greater than $0.5 \times T_{detect}$ and T'_{build} is greater than $1.5 \times T_{detect}$. Therefore, the retransmission time is at least $4\times$ of CRC error detection time. If there is error in retransmitted message, the retransmission needs to repeat again and much longer delay is incurred.

2.1.4 Reed-Solomon Code

Reed-Solomon (RS) code is a full-fledged error correction code that is employed as a reference for comparing with our proposed scheme. In RS code, every s bits are transformed into an element, called symbol, in Galois Field of 2^s (GF(2^s)) based on the value of these s bits. An RS code can be characterized by a triple (q, p, c), where q is the total number of symbols in an encoded message, p is the number of payload symbols and c is the number of error symbols that can be corrected. They satisfy the following conditions.

$$0 < p < q < 2^s \tag{2.4}$$

$$2c = q - p \tag{2.5}$$

Considering D=6 in FlexRay when data length up to 248 bytes which guarantee our proposed scheme can correct up to total 5 errors on dual channels in FlexRay, we implemented a $(254, 244, 5)$ RS code with $s = 8$, i.e., a symbol corresponds to a byte in FlexRay. According to the FlexRay protocol [48], it consists of three parts, header, payload and trailer. The header contains frame ID and other indicators which is used at decoder. The header part is fixed and cannot be changed, so is the CRC in trailer. We can only insert the RS code in payload segment which occupies some bandwidth of the data. In FlexRay, the 24 CRC bits are fixed. Therefore, the 10 bytes RS checking symbols occupy some message fields as shown in Figure 2.3. Since $c = 5$, this code can correct all 5-bit errors. RS encoding is similar to the polynomial division in CRC but based on GF addition

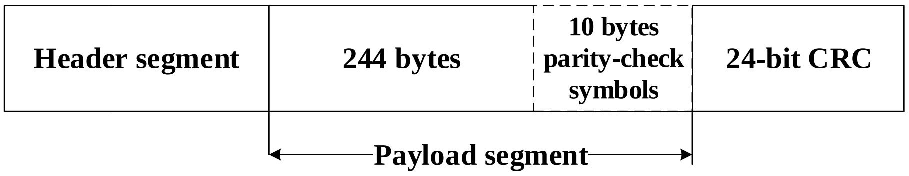


Figure 2.3: The original 254 bytes payload in FlexRay is reduced to 244 bytes if $(254, 244, 5)$ RS code is applied.

and GF multiplication. If there are communication errors, 2c-degree syndrome polynomial $Y(x)$ is not zero. The coefficients for $Y(x)$ can be calculated by substituting the 2c roots of the generator polynomial $G(x)$ into $T(x)$, which is the polynomial of received message. Based on the syndrome, one can locate the error positions and evaluate the error magnitudes using the key equation [49]:

$$\mu(x)Y(x) = \omega(x)\ mod\ x^{2c} \tag{2.6}$$

where $\mu(x)$ is error locater polynomial and $\omega(x)$ is error evaluator polynomial. In this work, we use Euclidean algorithm [50] to find the error locator polynomial. The error positions among symbols are the roots of the error locator polynomial, which can be found using the Chien-search algorithm [51]. The error values at each symbol are calculated using the Forney algorithm [52] and used to correct the errors.

2.2 Dual CRC Error Correction

In this section, we introduce our error correction scheme exploiting redundant communication of messages encoded by CRC, which is originally only for error detection. This scheme is called DUCER (DUal Crc Error coRrection). DUCER can be divided into two steps:

(1)Both channels send same messages during transmission. We compare each bit between two channels when the decoder receives them. An error occurs when the values from two channels are different. So we can detect the errors (except errors occurring at the same place of both channels) and record their positions. This step is called Error Position Identification.

(2)Next step is to identify the specific errors occurred on one of the channel and retrieve the original message. Two different schemes are proposed, named as DUCER1 and DUCER2, individually.

The encoding of DUCER is still CRC encoding like in existing protocols such as FlexRay and TTE. Suppose the CRC encoded message has q bits and its polynomial is

$$S(x) = m_{q-1}x^{q-1} + m_{q-2}x^{q-2} + ... + m_1 x + m_0 \tag{2.7}$$

and the CRC generator polynomial has degree k and is described by

$$G(x) = g_k x^k + g_{k-1}x^{k-1} + ... + g_1 x + g_0 \tag{2.8}$$

where $g_i \in \{0, 1\}, i = 0, 1, ..., k$ and $k < q$. Please note $S(x)\ mod\ G(x) = 0$. In the fault tolerant mode operation of dual communication channels, the same encoded message corresponding to $S(x)$ is simultaneously transmitted through two channels and the receiver receives two messages. If the errors in the two channels are $E_1(x)$ and $E_2(x)$, respectively, the polynomials of the received messages can be represented by

$$T_1(x) = S(x) + E_1(x), \quad T_2(x) = S(x) + E_2(x) \tag{2.9}$$

Error Position Identification: The first step of DUCER decoding is to identify bit positions of the errors through bit-by-bit comparison between $T_1(x)$ and $T_2(x)$. These are the bit positions where $T_1(x)$ and $T_2(x)$ have different values. If an error occurs at the same position in both $T_1(x)$ and $T_2(x)$, it cannot be corrected by DUCER. We will show that such probability is quite low and DUCER can correct 99.9998% of errors detected by CRC. If there are total t errors in $T_1(x)$ and $T_2(x)$, we use non-negative integers $p_1, p_2, ..., p_t$ to indicate positions of the identified errors.

Example: There is a 4-bit payload message “100” that is encoded with CRC of generator polynomial $G(x) = x^3 + x + 1$. The encoded message is "1000101" and the corresponding polynomial is $S(x) = x^6 + x^2 + 1$. The two received messages are “110010” and “1010101”. There are two bits with different values at positions corresponding to x^5 and x^4 in polynomials $T_1(x)$ and $T_2(x)$, respectively. Therefore, $p_1 = 5$ and $p_2 = 4$.

The second step of DUCER is to correct the bit errors whose positions have been identified in step 1. We describe two correction methods, DUCER1 and DUCER2. DUCER1 is a relatively naïve approach mostly to help ease understanding. DUCER2 is more efficient and the recommended scheme.

DUCER1: We enumerate all subsets of $\{p_1, p_2, ..., p_t\}$ to find one subset $\Phi \subseteq \{p_1, p_2, ..., p_t\}$ such that $(T_1(x) + \sum_{p_i \in \Phi} x^{p_i}) \mod G(x) = 0$. Then, the corrected message corresponds to $T_1(x) + \sum_{p_i \in \Phi} x^{p_i}$. Alternatively, one can obtain the corrected message based on $T_2(x)$ in the same way.

Example: We continue the example that has been used for illustrating how to identify error positions. In this example, the original encoded message is "1000101", while the received messages are "1100101" and "1010101" so that $p_1 = 5$ and $p_2 = 4$. Then, all four combinations $\{5\}, \emptyset, \{5, 4\}, \{4\}$ are enumerated to have four speculated error messages "0100000", "0000000", "0110000" and "0010000", respectively. These error messages are added with "1100101" corresponding $T_1(x)$ and obtain four speculated corrected messages "1000101", "1100101", "1010101" and "1110101". Polynomial division by $G(x)$ is performed for these four messages to obtain remainders as "000", "111", "110" and "001". Among them, only the first message "1000101" leads to zero remainder and therefore is the truly corrected message. For this message, $\Phi = \{5\}$ and $1100101 + 0100000 = 1000101$.

The main drawback for the primary scheme is that the time consumption grows exponentially when the number of errors increases. The enumeration in DUCER1 implies 2^t trial CRC decoding operations on the entire message, although t usually is a small number. In DUCER2, we reduce the complexity to t CRC decoding operations on entire message and 2^t summation operations of CRC bitwidth, which is substantially shorter than entire message.

Before introducing DUCER2, we describe some terms and an important conclusion as preparation. In CRC, an encoded message to be transmitted is n-bit payload followed by k CRC bits. In a decoded message at receiver, the lower k bits that correspond to the CRC bits are called *CRC check*, which can be represented by a polynomial $R(x) = T(x) \; mod \; G(x)$, called *residual polynomial*. DUCER2 is based on the following Lemma.

Lemma 2.2.1. Error Decomposition: *If a received message T has t-bit errors at positions $p_1, p_2, ..., p_t$, its CRC check value is equal to the sum of CRC values of t messages based on the same payload and CRC coding as T, each of which has single-bit error at positions $p_1, p_2, .., p_t$, respectively.*

Proof. If a received message $T(x)$ has t errors at positions $p_1, p_2, ..., p_t$, its error polynomial can be denoted as

$$E(x) = x^{p_t} + x^{p_{t-1}} + ... + x^{p_1}$$

and

$$T(x) = S(x) + E(x)$$

where $S(x)$ corresponds to the original encoded message. Its CRC check can be obtained as

$$R(x) = T(x) \; mod \; G(x) = (S(x) + E(x)) \; mod \; G(x)$$

where $G(x)$ is the generator polynomial for the CRC. By CRC encoding, we know $S(x) \; mod \; G(x) = 0$. Therefore,

$$R(x) = E(x) \; mod \; G(x).$$

Evidently,

$$(x^{p_t} + x^{p_{t-1}} + ... + x^{p_1}) \; mod \; G(x) = \sum_{i=1}^{t} x^{p_i} \; mod \; G(x)$$

and then,

$$R(x) = \sum_{i=1}^{t} (T(s) + x^{p_i}) \; mod \; G(x)$$

Thus, $R(x)$ is equal to the sum of check values of t messages, each of which has 1-bit error at $p_1, p_2, ..., p_t$, respectively. $\square$

DUCER2: We compute $R_1(x) = T_1(x)\ mod\ G(x)$ and $\hat{R}_i(x) = x^{p_i}\ mod\ G(x)$ for $p_1, p_2, ..., p_t$ identified in step 1. Please note $\hat{R}_i, i = 1, 2, ..., t$ are the CRC check values of t single-bit error messages mentioned in Lemma 2.2.1, and this computation takes t decoding operations of entire message length as p_i can be as large as the most significant bit of message. Then, according to Lemma 2.2.1, all subsets of $\{p_1, p_2, ..., p_t\}$ are enumerated to find one subset $\Phi \subseteq \{p_1, p_2, ..., p_t\}$ such that $R_1(x) + \sum_{p_i \in \Phi} \hat{R}_i = 0$. Please note the enumeration here is only for summation operations on CRC check values, which is substantially shorter than entire message. For instance, in FlexRay, a CRC check value has only 24 bits while its encoded message has 2056 bits. The corrected message corresponds to $T_1(x) + \sum_{p_i \in \Phi} x^{p_i}$. Alternatively, one can obtain the corrected message based on $T_2(x)$ in the same way.

Example: We use the same running example as in earlier descriptions. In this example, the original encoded message is "1000101", while the received messages are "1100101" and "1010101" so that $p_1 = 5$ and $p_2 = 4$. The CRC check value for the first received message is "111". The CRC check values corresponding to x^5 and x^4 are "111" and "110", respectively. The enumeration of all four subsets of $\{p_1, p_2\}$ leads to

$$\begin{aligned} \{5,4\}: &\quad 111 + 111 + 110 = 110 \\ \{5\}: &\quad 111 + 111 = 000 \\ \{4\}: &\quad 111 + 110 = 001 \\ \emptyset: &\quad 111 + 000 = 111 \end{aligned}$$

Therefore, $\Phi = \{5\}$ and the corrected message is "1100101+0100000=1000101". Please note addition in Galois field is actually logic exclusive-or operation.

2.3 Error Correction Capability of DUCER

We discuss the probability P_f that DUCER fails to correct an error. There are two cases contributing to such failures: (i) an error cannot be detected by CRC, characterized by residual error probability P_{rs} described in Section 2.1.2; (ii) two errors in two received messages are at the same position, with probability P_{same}. DUCER1 and DUCER2 share the same two cases and the same P_f. It is likely that two errors at the same position are not detected by CRC. Thus, there is overlap between cases for P_{rs} and P_{same}. As such, P_f is upper bounded by $P_{rs} + P_{same}$. Residual error probability P_{rs} has been studied in the previous work [47] and briefly reviewed in Section 2.1.2. For FlexRay with BER 10^{-6}, an upper bound of P_{rs} is 7.07×10^{-13}.

For P_{same}, we reach the following conclusion.

Lemma 2.3.1. *If channel BER is b, for two n-bit messages, the probability that any two errors of the two messages are at the same location is given by*

$$P_{same} = 1 - (1 - b^2)^n \tag{2.10}$$

Proof. The probability that two messages have error at the same specific position is b^2. Then, the probability that a bit does not have errors in both messages is $1 - b^2$. Hence, the probability that no bit having errors in both messages is $(1 - b^2)^n$. □

Based on Lemma 2.3.1, we obtain that P_{same} for FlexRay with BER 10^{-6} is 2.03×10^{-9}. When BER of FlexRay is 10^{-7}, P_{same} becomes 2.01×10^{-11}. Compared to upper bound of $P_{rs} = 7.07 \times 10^{-13}$ under the same condition of BER 10^{-6} in FlexRay, we can tell that P_f is dominated by P_{same}.

It is not difficult to find that the probability of having any errors $P_{error} = 2.029 \times 10^{-3}$ for FlexRay with BER 10^{-6}. Approximately, the probability that DUCER can correct errors detected by CRC is given by

$$\frac{P_{error} - P_{same}}{P_{error} - P_{rs}}$$

and it is 99.9998% for FlexRay with BER 10^{-6}.

2.4 Decoding Circuit Implementation

We first describe CRC decoder circuit, which is for CRC error detection and also an important component in DUCER. Consider a message encoded with k-bit CRC with generator polynomial

$G(x) = g_k x^k + g_{k-1} x^{k-1} + ... + g_1 x + g_0$. The decoding or polynomial division is realized by LFSR (Linear Feedback Shift Register) as shown in Figure 2.4. The received message $T(x)$ is serial input, 1 bit per clock cycle. The binary multiplication and addition are realized by logic AND and XOR, respectively. One operand to each multiplication is $g_i, i = 0, 1, ..., k$, which is the coefficient in $G(x)$. The output at right is the remainder of the polynomial division. If the remainder is not zero, an error is detected.

LFSR (Linear Feedback Shift Register) is used. LFSR is a type of register which its current input is achieved though a linear function of its previous output. The function we used here is binary addition (XOR). A delay flip-flop and a adder can build a one degree LFSR. A k degree LFSR with k delay flip-flops and adders can compute XOR between two 24 bits code in a clock circle and is used to calculate the k-bit CRC check value. The structure is shown in Fig 2.4 with one bit input every clock circle.

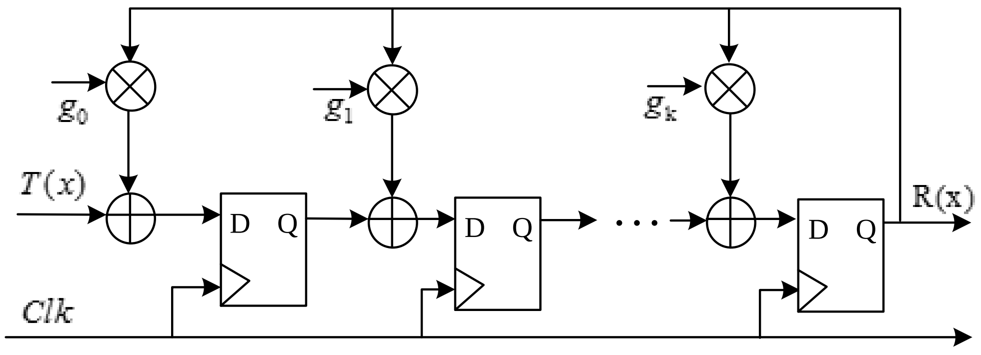


Figure 2.4: Linear Feedback Shift Register (LFSR) for realizing decoding of k-bit CRC.

The circuit architecture shared by DUCER1 and DUCER2 is shown in Figure 2.5. Two received messages $T_1(x)$ and $T_2(x)$ are first XORed to find the bits where they are different. They are indicated by the positions of bits with different values $\{p_1, p_2, ..., p_t\}$, which are stored in registers. The control unit takes $T_1(x)$ and $\{p_1, p_2, ..., p_t\}$ as input, and generates output $T_c(x)$. For

DUCER1, $T_c(x)$ indicates messages generated by flipping the value at positions $\{p_1, p_2, ..., p_t\}$. For DUCER2, $T_c(x)$ means messages of $x^{p_i}, i = 1, 2, ..., t$. Then, all messages of $T_c(x)$ are fed into the LFSR to obtain the remainders, or CRC check values $R(x)$. The error locator is to decide the Φ corresponding to the zero CRC check and then obtain $E(x) = \sum_{p_i \in \Phi} x^{p_i}$. The last component, error corrector, performs $T_1(x) + E(x)$ to reach the corrected message $S(x)$.

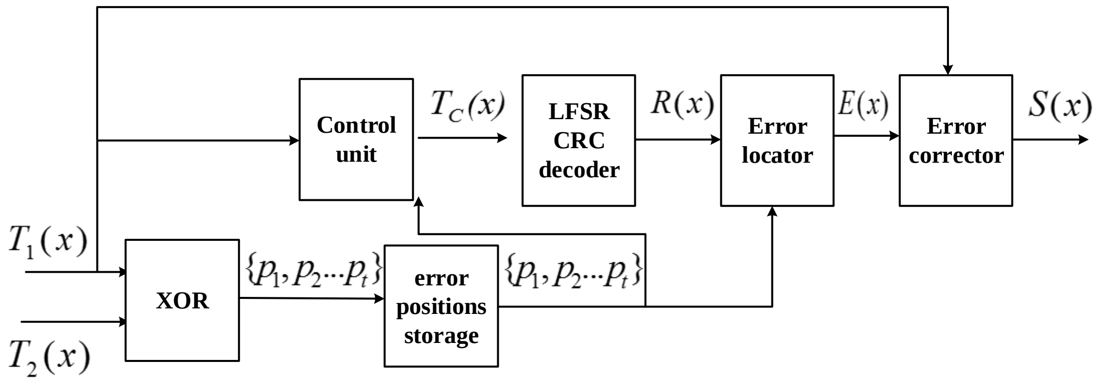


Figure 2.5: Circuit architecture of DUCER1 and DUCER2.

The circuit for RS code is more complex than DUCER. A diagram for RS code implementation is shown in Fig 2.6. Registers are needed to record the rules of computations in GF Field. For a (p,q,c), Syndrome calculator block implements matrix multiplication to design the coefficients of Syndrome $Y(x)$. Euclidean algorithm block is designed to fix the coefficients for error locater polynomial $\mu(x)$. Chien-search block is the circuit designed to find the error locations. A circuit realizing Forney algorithm calculate the error magntitudes. where $p_1, P_2...P_c$ in the diagram denotes error positions and $m_1, m_2...m_c$ denotes the error magnitudes. $S(x)$ is the original message.

2.5 Experimental Evaluations

In this section, we present evaluations of the proposed DUCER technique in terms of computation runtime and hardware implementation cost. Please note DUCER does not change encoding

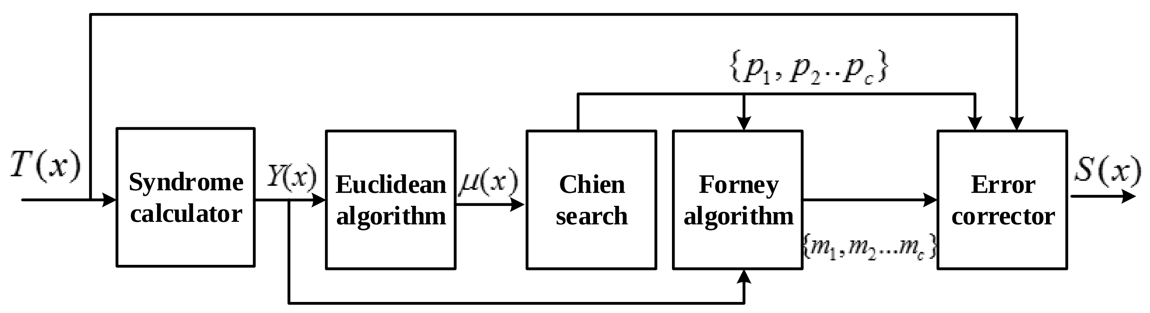


Figure 2.6: Circuit diagram for RS code

of existing protocols. Thus, the computation here is only for decoding. In contrast, ECC like RS would cause non-trivial extra encoding overhead. Since we do not have access to actual in-vehicle networks, the experiments are performed with software implementation on microprocessor and circuit realization on FPGA. With comparisons on both software and hardware platforms, the results can clearly show the differences among different methods.

2.5.1 Decoding Runtime

Decoding runtime is an important issue and is a key reason why full-fledged error correction code like RS has not been adopted for in-vehicle networks. The expectation is that the error correction runtime should not be greater than message retransmission time, and much shorter than that of classic ECC like RS code. The evaluations here are mostly based on FlexRay protocol with 254-byte payload, 24-bit CRC and BER 10^{-6}. The experiment is performed for at most 5 errors. The probability of more than 5 errors is as low as 5.10×10^{-14} for FlexRay.

2.5.1.1 Software Decoding Time

DUCER and $(254, 244, 5)$ RS code are implemented in C language, and executed on Intel Core Xeon 2.8GHz CPU with Linux system. One million messages are randomly generated for each case of 2, 3, 4 and 5 errors. The average decoding times of each case for different schemes are listed in Table 2.3. Please note that all the errors in this experiment are successfully corrected by DUCER and the time for RS is the average time when dealing with 2 to 5 bits errors. RS runtime is quite large, and about one order of magnitude higher than DUCER2. When the number of errors is 2, DUCER2 is $14\times$ faster than RS. If the probability of having t-bit errors is P_t, then the probability of having $t+1$ errors is $P_{t+1} = \frac{b}{1-b}P_t$, where b is BER. For BER 10^{-6}, $1-b$ is nearly 1 and P_{t+1} is several orders of magnitude less than P_t. If we estimate the expected decoding time considering different numbers of errors, the runtime of decoding 2 errors immensely dominates those beyond 2 errors. Therefore, we can say DUCER2 is one order of magnitude faster RS in general. DUCER1 starts with very short runtime when the number of errors is low, but has poor scalability. Our proposed DUCER2 is much more scalable than DUCER1.

Table 2.3: Average software decoding runtime of 1 million runs in μs.

# errors	DUCER1	DUCER2	RS
2	307	287	4158
3	998	484	4158
4	2226	609	4158
5	4648	736	4158

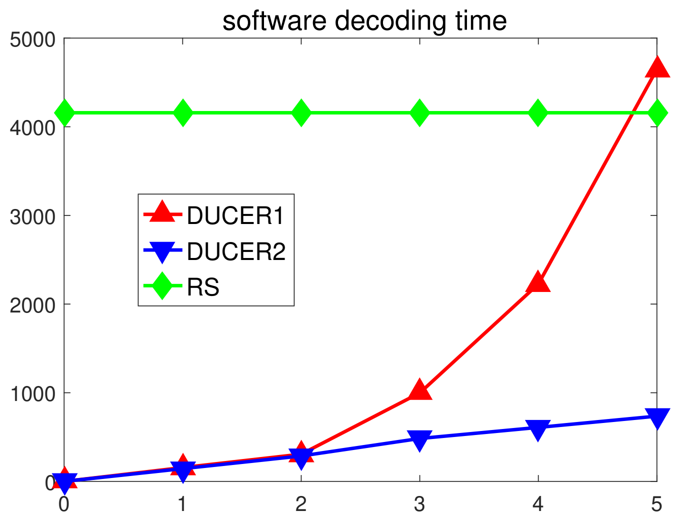


Figure 2.7: Runtime comparison in C to correct different number of errors

We also evaluated software decoding runtime for cases with random errors and the results are shown in Table 2.4. Each result here is the average of 1 million randomly generated messages. The table also includes runtime overhead with respect to CRC decoding time, which is $9.5\mu s$. When BER is 10^{-7}, which is still quite noisy channel, the overhead from our DUCER correction is only 0.11%.

Table 2.4: Average software decoding runtime in μs for cases with random errors. The overhead is versus CRC decoding time.

	DUCER1		DUCER2		RS	
BER	Time	Overhead	Time	Overhead	Time	Overhead
10^{-6}	9.603	1.08%	9.603	1.08%	17.797	87.34%
10^{-7}	9.510	0.11%	9.510	0.11%	10.330	8.73%

In Table 2.5, we compare software decoding runtime for 24-bit CRC, which is used in FlexRay, and 32-bit CRC, which is used in TTE. With the about 30% CRC bitwidth increase, the runtime increase of DUCER2 is about 20%. Therefore, the impact of CRC bitwidth increase is sub-linear.

Table 2.5: Software decoding runtime in μs for 24-bit CRC and 32-bit CRC.

	DUCER1		DUCER2	
# errors	24-bit CRC	32-bit CRC	24-bit CRC	32-bit CRC
2	307	396	287	359
3	998	1163	484	595
4	2226	2517	609	715
5	4648	5509	736	872

2.5.1.2 Hardware Decoding Runtime Cost

All of CRC, DUCER and RS decodings are also implemented with hardware circuits using Xilinx Zynq-7000 FPGA in Vivado, which operates at $50MHz$. The hardware runtime results are showed in Figure 2.8 and Table 2.6. DUCER2 is $9.6\times$ faster than RS even when there are 5 errors. If the encoding time is also counted, the overall runtime advantage of DUCER2 versus RS would be even greater. The runtime trends of these schemes are clear in the plot of Figure 2.8. Compared to RS, the DUCER2 runtime is much closer to that of CRC error detection. Also, DUCER2 has much better scalability than DUCER1. From Section 2.1.3, we know that message retransmission time is at least $4\times$ error detection time. The results in Table 2.6 indicate that the DUCER decoding time is at the same order of magnitude as message retransmission time.

Table 2.6: FPGA decoding runtime in μs.

# errors	CRC	DUCER1	DUCER2	RS
2	40.67	162.65	122.05	2352.05
3	40.67	366.11	162.79	2352.05
4	40.67	691.55	203.61	2352.05
5	40.67	1342.43	244.59	2352.05

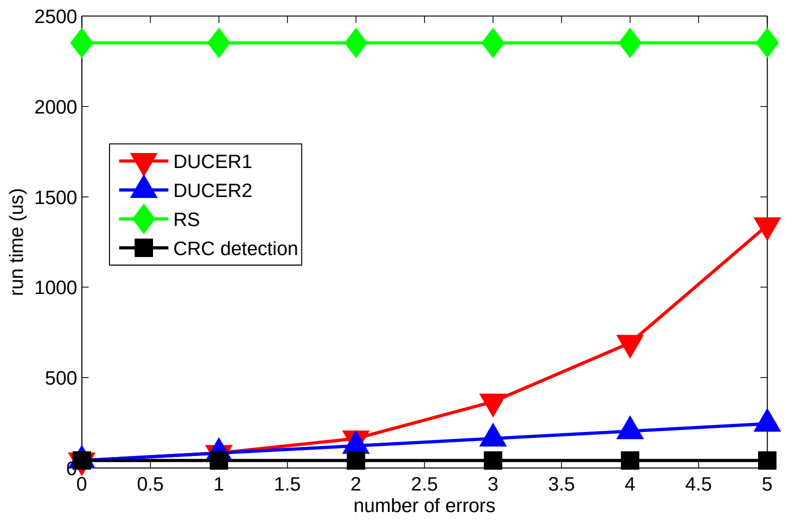


Figure 2.8: FPGA decoding time comparison.

2.5.2 Hardware Implementation Cost

In hardware implementation, circuit area cost is another important concern. In Table 2.7, the utilization of major hardware resources is summarized for the different methods. In this table, cells include all kinds of circuit elements, such as memory, lookup tables (LUTs), which do the major logic and datapath computation, and flip-flops (FFs). The number of nets largely affects both wire routing and circuit timing. In our design, DUCER1 and DUCER2 share the same architecture, whose area cost is 11% of RS in term of the number of cells. Considering that DUCER uses the same encoding as CRC while RS encoding would incur extra cost, the overall area of DUCER is about one order of magnitude less than RS.

Table 2.7: Hardware cost in FPGA implementation.

	# Cells	# I/O pins	# Nets	# LUTs	# FFs
CRC detection	80	7	106	25	39
DUCER	482	8	562	170	230
RS code	4576	8	5893	2834	867

2.6 Conclusions

In-vehicle network communication requires fast error handling for safety-critical messages. In this work, we propose DUCER, a fast and lightweight error correction scheme exploiting the redundant channels in recent protocols. It avoids message retransmission, which causes information delay uncertainty. DUCER can correct 99.9998% errors that are detected by CRC without changing encoding of existing protocols. It can correct 5-bit error with 254 byte payload within $1ms$. Its software runtime and hardware cost are both one order of magnitude less than that of Reed Solomon code.

Although DUCER is a general technique for in-vehicle network with redundant communication channels, its evaluation here is mostly based on FlexRay, which requires that both channels use the same given CRC generator polynomial. In future research, we will further improve the error correction efficiency by using different CRC polynomials for the redundant channels.

3. INFORMATION PROCESSING APPLICATION IN SMART IRRIGATION

In this chapter, we will introduce the application of information processing techniques to improve the precision and reliability for a smart irrigation system. Our work can be divided into three parts: high-accuracy positioning system and voltage monitoring system.

3.1 Background

3.1.1 Hardware in Smart Irrigation System

3.1.1.1 Microcontrollers

Several microcontrollers are employed in the irrigation system for sensor data acquisition, data communication and central control, including Arudino Uno board, Intel Edison board, Intel Up-Square board and Arduino Mega board.

Arduino Uno Board, which is shown in Figure 3.1(a), is an microcontroller designed based on the Microchip ATmega328P processor core. It is small, easy to use and able to be interfaced with various expansion boards and circuits. Uno can be powered by a relatively wide range of voltage (from 7V to 20V). Software and control code can be easily uploaded though integrated development environment (IDE).An Uno board has 20 pins, 14 digital pins and 6 analog pins. Differnt pins works for specific functions:

- Universal Asynchronous Receiver Transmitter (UART): Two pins, pin 0 and pin 1. They are for receiving and transmitting serial data individually.
- External interrupts: Two pins, pin 2 and 3, can be used to trigger an interrupt when their voltage changes.
- PWM (Pulse-Width Modulation): Some pins provide a 8-bit PWM output.
- AREF (Analog REFerence): Some pins can identify analog voltage inputs.

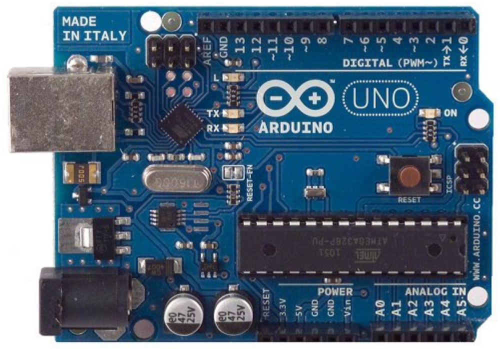
- General Purpose Input/Output (GPIO): These pins can read high or low input or act as switches that output 5 volts when set to high and no voltage when set to low.

In our system, Arduino Uno was used for moisture sensor data acquisition, sensor data transmission and analog voltage measurement.

Another Arduino series' board used in our system is Mega, which is shown in Figure 3.1 (b). A comparison between Arduino Uno and Mega is provided in Table 3.1. Arduino Mega is adopted for irrigation nozzle control in our system.

Table 3.1: Comparison between Uno and Mega

	Uno	Mega
Processor	Atmega328p	Atmega2560
Clock Speed (MHZ)	16	16
Flash Memory (kB)	32	256
SRAM (kB)	2	8
# Digital Pins	14	54
# Analog pins	6	18


(a) Arduino Uno

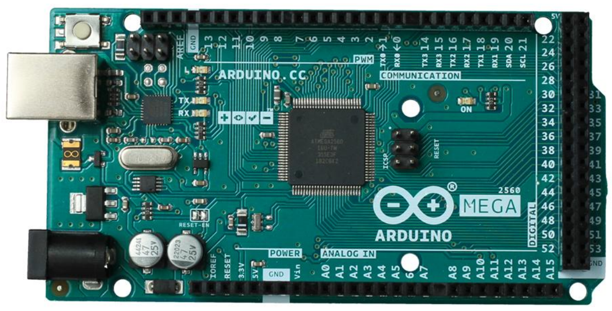

(b) Arduino Mega

Figure 3.1: Arduino Uno and Mega board

Intel Edison Board , shown in Figure 3.2, plays the role of central control in our system.

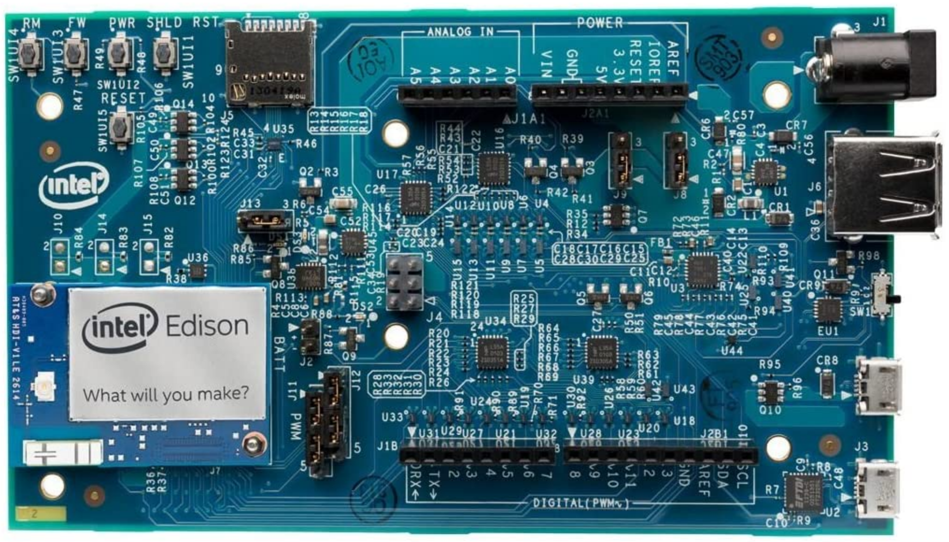


Figure 3.2: Intel Edison board

Some hardware features of Intel Edison are:

- a dual-core, dual-threaded Intel Atom CPU at 500 MHz.
- 1GB RAM and 4GB ROM.
- support to bluetooth and Wi-Fi.
- 40 GPIO.
- Clock frequency under different mode is 19.2MHZ and 32 MHZ.

Intel Edison is in small size packed with a variety of good features, which providing good service for wireless and and Internet of Things (IoT) applications. It is powered by Intel Atom 500 MHZ core and contains more than 4GB memory. A block diagram of Intel Edison is shown in Figure 3.3.

The last microcontroller we used in our system is Intel Up-squared board. It is more powerful than Edison and is employed to collect and process GPS data in our system. A monitor can be connected to Up-squared to show the real time data and make it friendly for the users to check

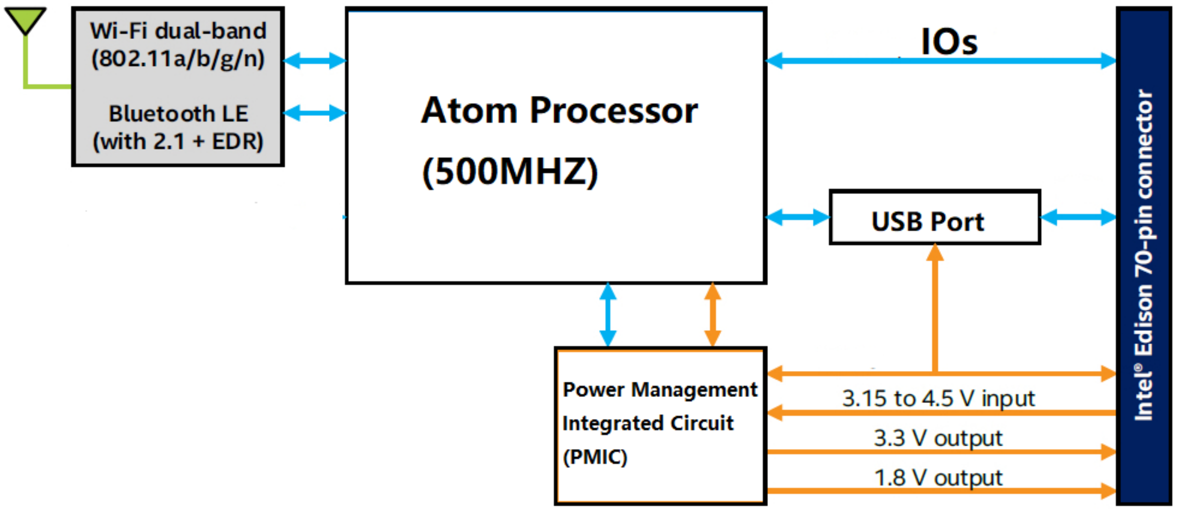
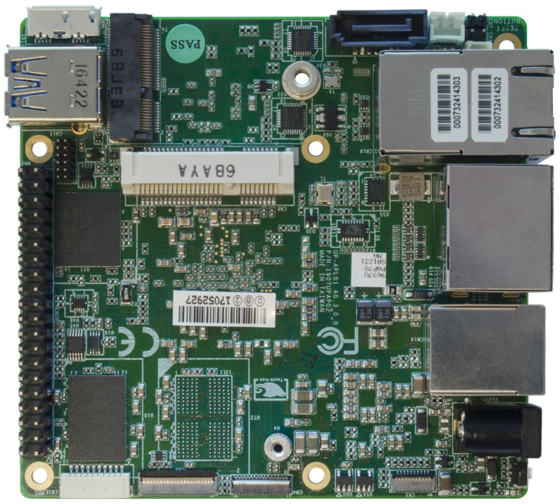
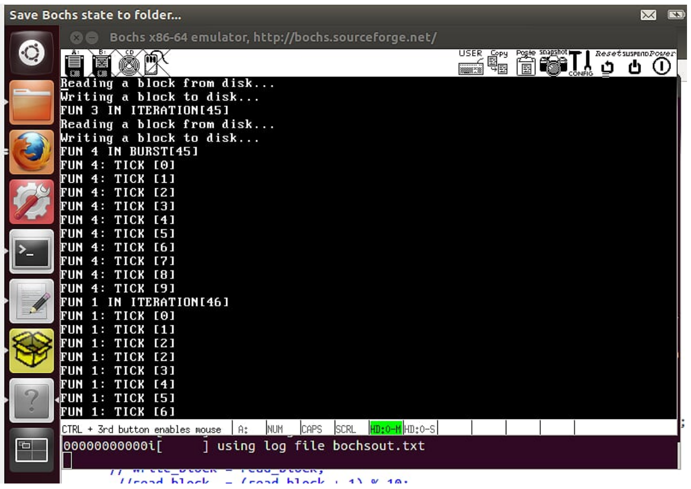

Wi-Fi dual-band (802.11a/b/g/n)
Bluetooth LE (with 2.1 + EDR)
Atom Processor (500MHZ)
IOs
USB Port
Intel® Edison 70-pin connector
Power Management Integrated Circuit (PMIC)
3.15 to 4.5 V input
3.3 V output
1.8 V output

Figure 3.3: Intel Edison block diagram, adapted from [1]

operation status from time to time. A picture of Up-squared board is shown in Fig 3.4(a). Ubuntu 16.0 is installed in this board, and its user interface is shown in Fig 3.4(b).



(a) Intel Up-squared board


Save Bochs state to folder...
Bochs x86-64 emulator, http://bochs.sourceforge.net/
USER
Copy
Paste
snapshot
CONFIG
Reset
SUSPEND
Power
CTRL + 3rd button enables mouse
A:
NUM
CAPS
SCRL
HD:0-M
HD:0-S

```
Reading a block from disk...
Writing a block to disk...
FUN 3 IN ITERATION[45]
Reading a block from disk...
Writing a block to disk...
FUN 4 IN BURST[45]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
FUN 4: TICK [6]
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
FUN 1 IN ITERATION[46]
FUN 1: TICK [0]
FUN 1: TICK [1]
FUN 1: TICK [2]
FUN 1: TICK [2]
FUN 1: TICK [3]
FUN 1: TICK [4]
FUN 1: TICK [5]
FUN 1: TICK [6]
```

```
00000000000i[     ] using log file bochsout.txt
```

(b) Ubuntu 16.0 installed on Up-Squared

Figure 3.4: Up-squared board and its operating system

3.1.1.2 Wireless Communication Module

We use XBee/Zigbee modules to realize the wireless communication between different micro-controllers. XBee/Zigbee is a popular and portable integrated circuit family to support compatible and middle range distance wireless communication [53]. They are embedded circuits providing wireless end-point connectivity between different devices, such as computers and different micro-controllers. XBee/Zigbee devices communicate with each other over the air, sending and receiving wireless messages. Xbee/Zigbee can be powered though charger or USB, as shown in Figure 3.5, and transmit or receive data from local devices. The XBee/Zigbee in our system has a 4000 ft

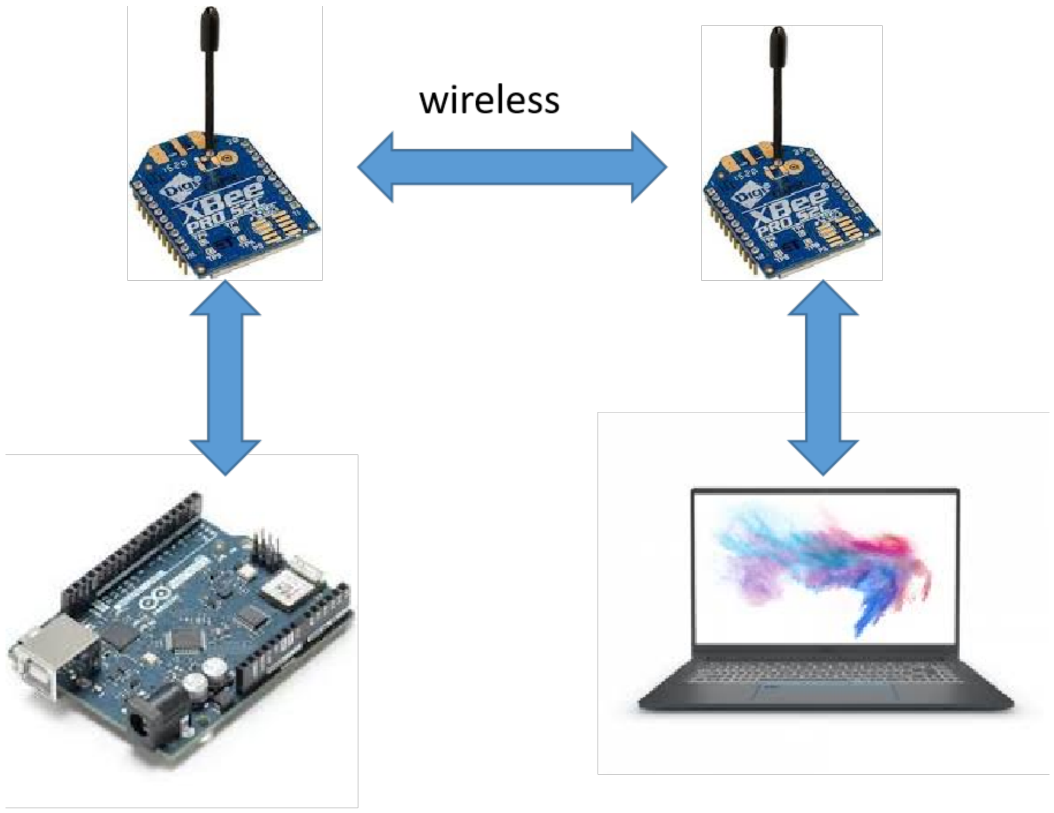
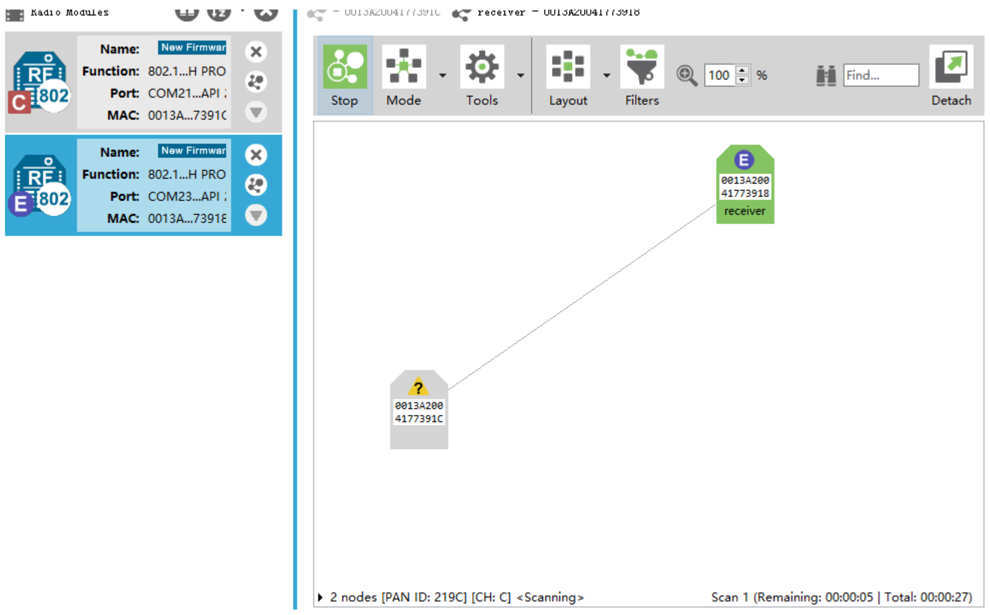


Figure 3.5: Xbee wireless communication

sight range under normal conditions. At the beginning, we need to implement a configuration for each Xbee, to make sure they are under the same network and the same channel. This process is illustrated in Figure 3.6.


Name: New Firmwar
Function: 802.1...H PRO
Port: COM21...API
MAC: 0013A...7391C

Name: New Firmwar
Function: 802.1...H PRO
Port: COM23...API
MAC: 0013A...7391E

0013A200 41773918 receiver

0013A200 4177391C

2 nodes [PAN ID: 219C] [CH: C] <Scanning>

Scan 1 (Remaining: 00:00:05 | Total: 00:00:27)

Figure 3.6: Xbee modules configuration

3.1.2 Irrigation Management

A picture of the center pivot irrigation machine is provided in Figure 3.7 and a top-down diagram is shown in Figure 3.8.

- Water is pumped through the pipe from the center point.
- The system moves on wheels, water is released along the radius.
- Structure between two adjacent towers is called span. The whole machine consists of several similar spans.
- Center controller controls movement of the outermost tower.

A center pivot irrigation machine releases water from its nozzles along its arm while it is sweeping through a field. The total amount of water irrigated is proportional to the time it spends over the

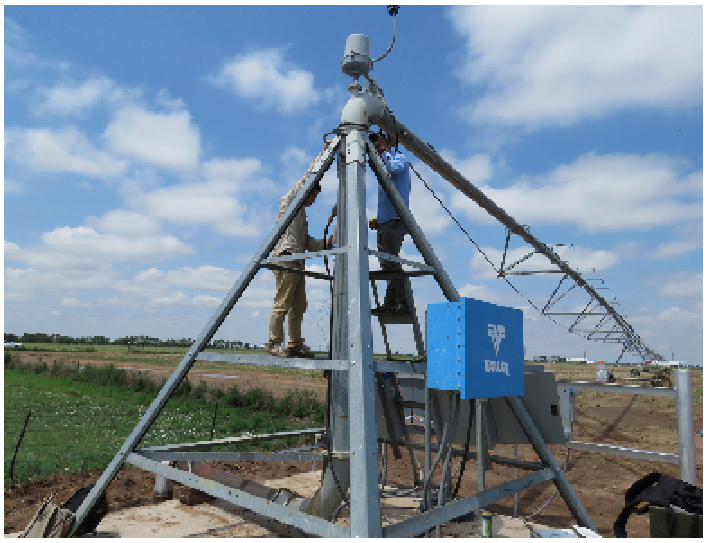


Figure 3.7: Center-pivot irrigation machine

field when the flow rate is fixed. When the speed doubles, half time is spent to pass a particular area and half amount of water is irrigated at this area. In summary, we change the movement speed of irrigation machine to adjust irrigation water. An overall architecture for control system is shown in Figure 3.9. The system consists of six parts:

- Central control unit
- Irrigation machine
- Positioning system
- Soil moisture system
- Central information unit
- User interface

Soil moisture data is collected from soil sensors and sent through Xbee/Zigbee modules to central information unit at center pivot. There are two Edison boards in the control box at the base of the center pivot machine. One central information unit, which collects a variety of information from

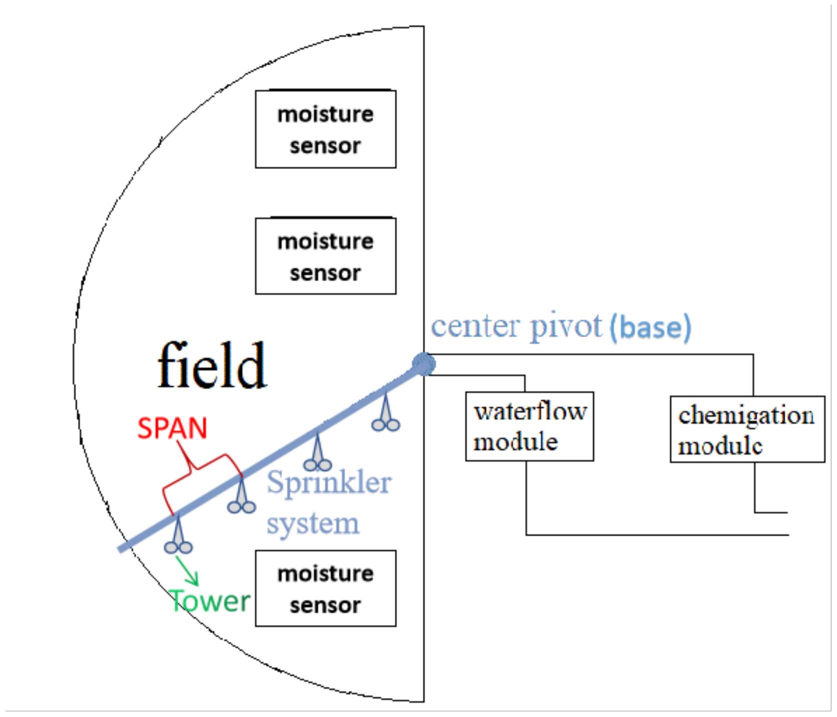


Figure 3.8: Water released along the radius and its amount decided by machine movement speed

other devices or from the internet, such as soil sensor data, weather and GPS data. Another Edison board is central control unit, which interacts with client, analyze data stored in information center and control the speed of machine to irrigate different amount of water. An authorized user can either input their commands on site by using a Linux-based control interface, or use a web based application for remote control. Positioning system will be installed at the outermost tower, which records the real-time location of irrigation machine. The location information will be continuously sent from positioning system to the information unit for speed control to adjust the amount of irrigated water. Global Positioning System (GPS) is is applied in our system to track the location and speed of the outermost tower of the center pivot machine. However, the errors of ordinary GPS can be several meters and is not accurate enough for the irrigation control. a space-based positioning system owned by United states government. Its principle is based on triangulation [54]. The coordinate of receiver is determined by calculating its distance to four different satellites using the propagation time. However, Many devices can only guarantee reliable (95% Confidence Interval)

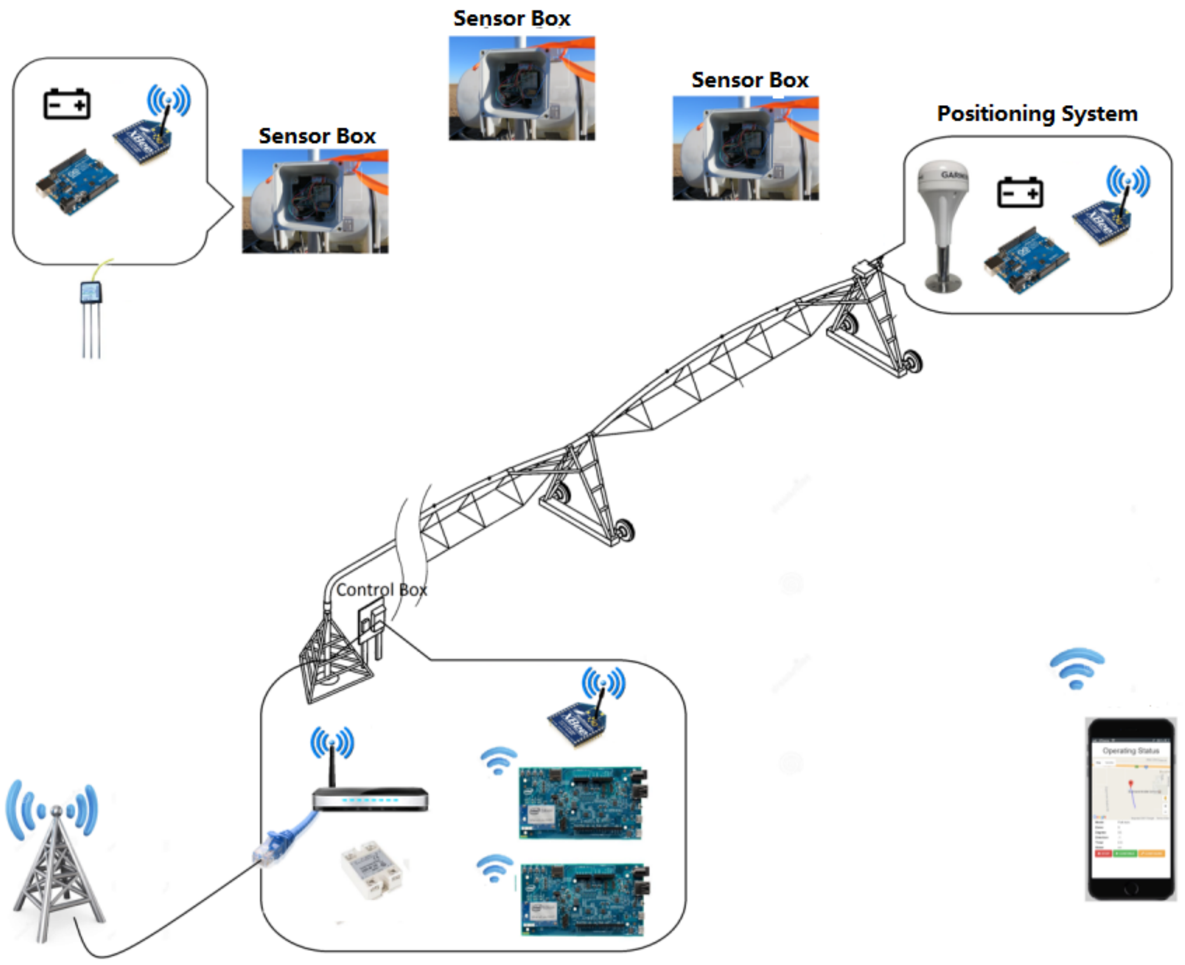


Figure 3.9: Control system architecture

7.8 meters positioning accuracy according to the United States government report. The error can even reach $10m$ level under some conditions. There are many reasons leading to the error. According to [55], the changeable atmospheric conditions, affecting the speed of the signals transmission, can add small errors in the calculated coordinates. In urban area, due to the block and reflections of tall buildings, receiver may get unclear signals which influence the distance calculation. As a result, accurate navigation in urban area is more difficult. The error of GPS will cause large deviation from GPS curved vehicle movement path to the actual one and make the navigation system send the wrong instruction, bring inconvenience to users and even lead to the traffic accidents. How to get precise GPS and actual movement path is a realistic and difficult topic where so many trials have been implemented on.

3.2 Previous Work

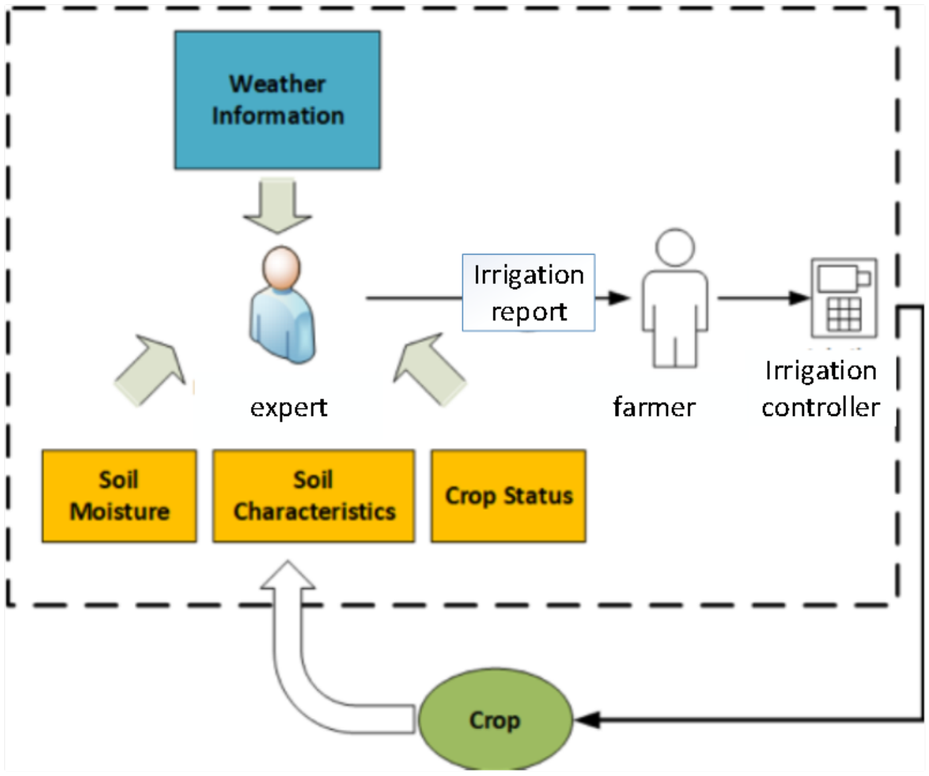


Figure 3.10: Traditional irrigation system with manual calculation

Before the development of in-field sensors, irrigation system will apply water quite uniformly to different zones in the field. It leads to a waste of water given that the variations in soil properties across most fields. At the beginning of 21st century, soil sensors became popular to monitor soil conditions at different site in a field in order to guide irrigation actions. Paper [56] proposed a wireless in-field sensor system with six soil sensors across the field. The irrigated water is decided by an experienced expert based on feedback from sensors, as shown in Figure 3.10. Irrigation decisions relying on manual calculation can be very tedious when more and more information is introduced to the sensor system. After that, some agricultural models were designed to simulated crop growth under different conditions, such as DSSAT (The Decision Support System

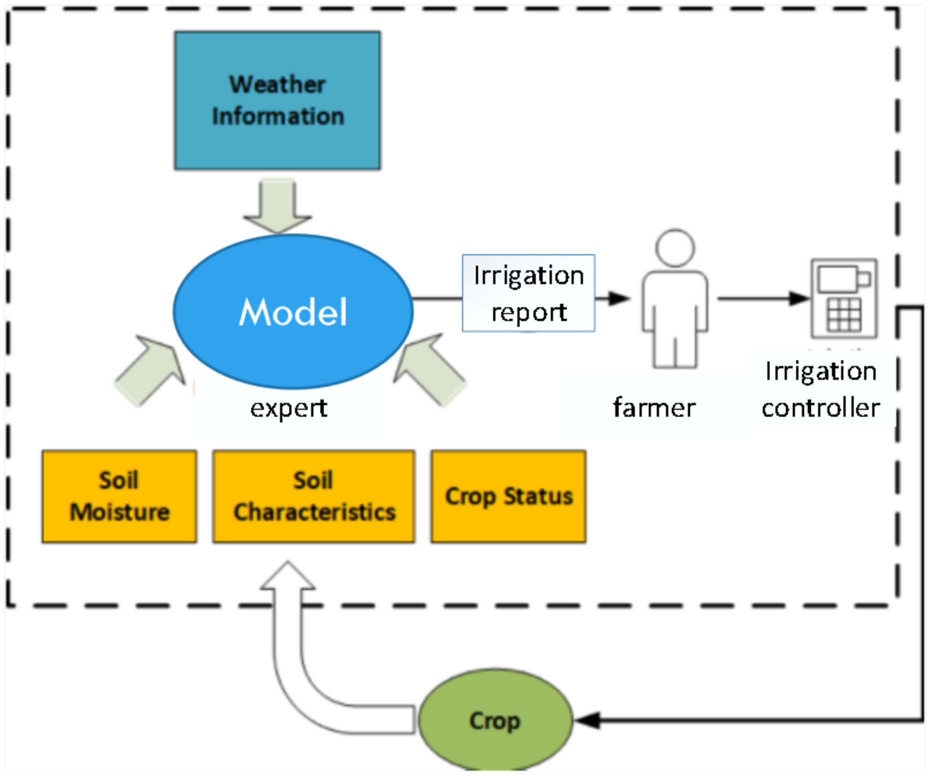


Figure 3.11: Model based irrigation system

for Agrotechnology Transfer) [57]. Model Predictive Control (MPC) wes then proposed in some works [58] [59], as described in Figure 3.11. Paper [58] built a model to establish the relationship between water, soil and crops with considering evaporation. Paper [59] took weather as one of the input for their model. However, models were all established based on historic data. Performance of MPC decreases after a long period of time.

With the development of machine learning techniques, one of my teammate proposed an irrigation decision model in [60]. Model-based reinforcement learning technique is applied to determine irrigation decisions. It can make decisions based on the information of weathers, sensors and what it learnt from the previous trails. Two neural networks (NNs) are also introduced to predict the DSSAT simulation results. One NN inputs irrigation and weather information and predicts total soil water content while the other NN predicts crop yield given daily total soil water content of an entire crop season. Subsequently, prediction of the crop yield is used as the training data to train

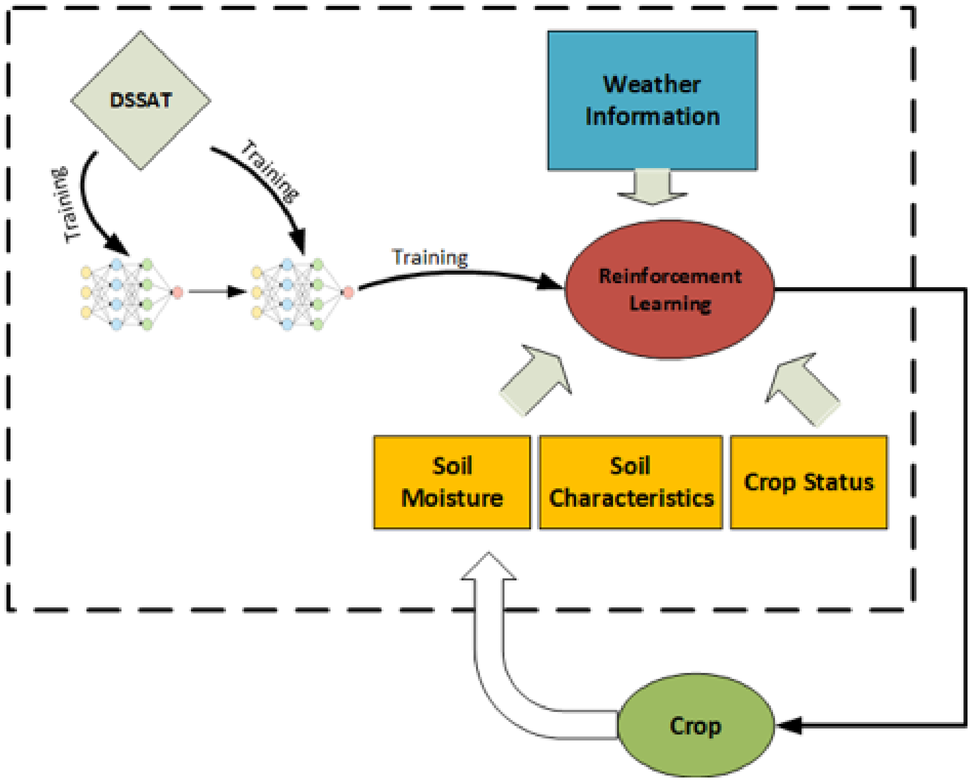


Figure 3.12: Reinforcement learning irrigation system

the reinforcement learning model. The basic structure is depicted in Figure 3.12. This approach achieved relatively precise irrigation and allows full automation of the irrigation process. However, this method is restricted to small state space size and difficult to scale to large problems. Therefore, it is difficult to accurately represent actual irrigation context, leading to loss of important information that is very much needed for optimizing irrigation decisions.

Based on work [60], another teammate proposed a deep reinforcement learning-based irrigation control method [61]. It can overcome the drawbacks of traditional reinforcement learning and other machine learning-based irrigation control. Our current irrigation decision algorithm is based on [61].

3.3 Positioning System

Our irrigation decision making is based on a lot of information, such as machine location, weather and soil moisture. The far-end (corresponding to the outermost tower) location of the center pivot machine is important, as the amount of irrigation depends on both the site and machine moving speed. In this section, we describe several techniques for obtaining high accuracy for positioning the pivot machine.

3.3.1 Three-Point Localization Method(TPL)

A traditional and widely-used localization method is three-point localization method (TPL) [62]. This method measures the distances between the target point and three different pivot points. The location of the target point is determined by the crossing point of three circles generated from three different pivot points. As shown in Figure 3.13, three different fixed pivots are installed in the field and the outermost tower of the irrigation machine is treated as the target point. Distances between the target and different pivots are then measured. The pivot point locations are known and the coordinates of the target can be calculated according to the intersection point of the three circles. Note that the track of the outermost tower of irrigation machine is also on a circle with radius of 1200 feet. The information can be used to further prune out the outliers and improve the accuracy. Sometimes, the three circles do not cross at a single point due to errors of distance measurements. As a result, the location of the target point is determined by center of a circle fitted by the three intersection points (indicated as orange points in Figure 3.14). First, we obtain two straight lines l_1 and l_2 though different pairs of intersection points, as indicated in Figure 3.14. Their perpendicular bisectors are $l_1^{'}$ and $l_2^{'}$. The position of the target point is estimated to be the crossing point between $l_1^{'}$ and $l_2^{'}$.

We investigate two different ways to measure the distances between three pivots and target, named as Timing Based Three-Point Localization Method (TTPL) and Received Signal Strength Indicator (RSSI) based Three-Point Localization Method(RTPL).

Signal propagation time multiplied by the speed of light is used to calculate the distance between a pivot and the target point. XBee/Zigbee modules are installed on the pivots and the target point individually. XBee/Zigbee on ta pivot sends a communication request to XBee/Zigbee on the target point and then receives an acknowledgement sent back by the target. XBee/Zigbee CPU timer can be used to calculate the time between sending request and receiving acknowledgement.

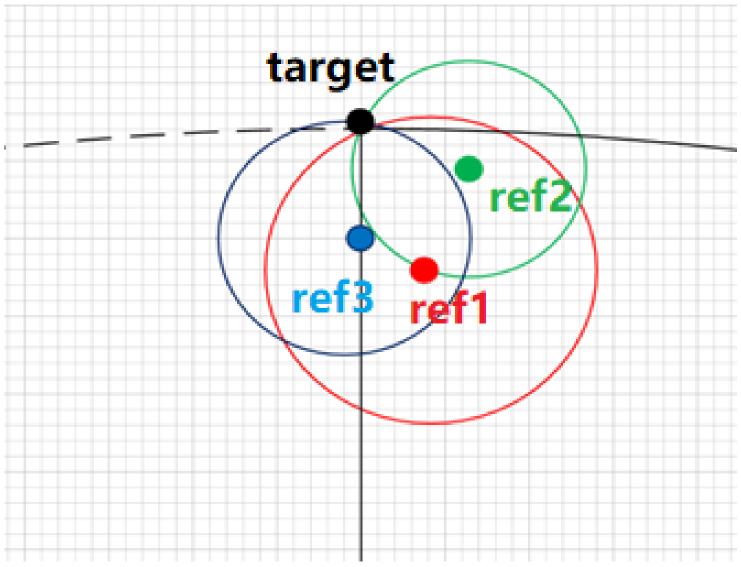


Figure 3.13: Three-point localization method (TPL)

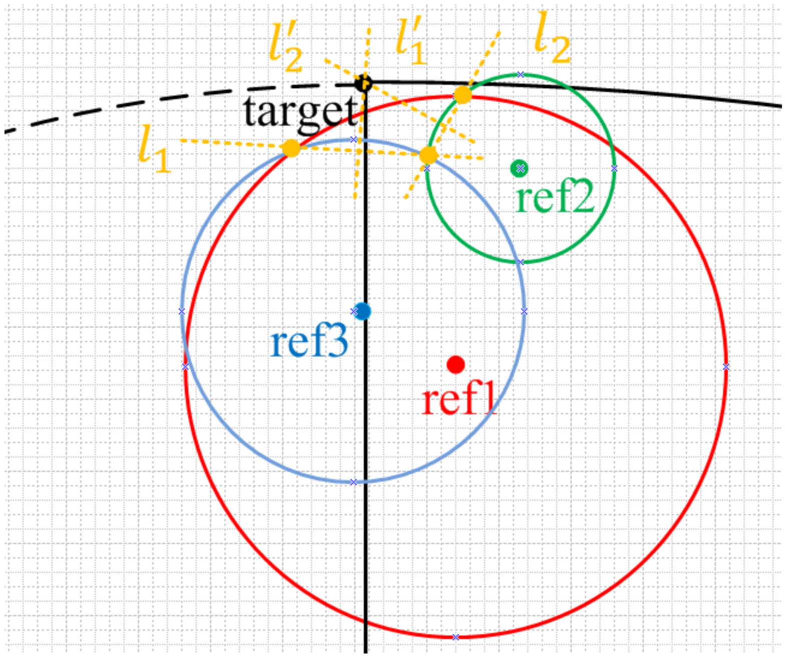


Figure 3.14: Target localization when three circles do not cross at one point

The time multiplied by light speed is equal to the double of the distance between the pivot and target point. A diagram to illustrate this idea is shown in Figure 3.15. With the distances, we can then calculate the intersection point.

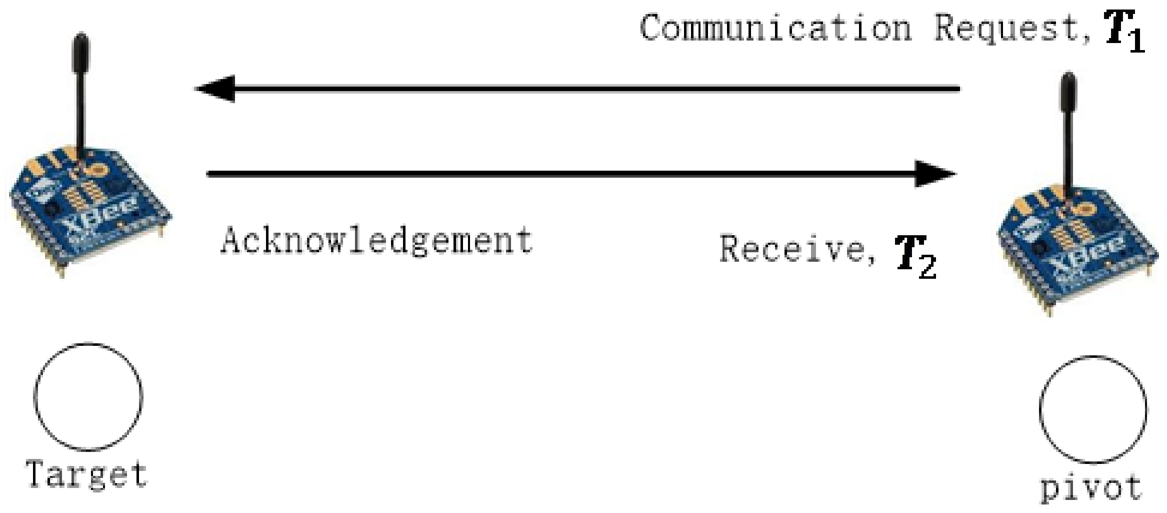


Figure 3.15: Timing based three-point localization method (TTPL) diagram

For signal strength-based method, TPL is also used to find the intersection of three circles. Instead of using signal propagation time, we use RSSI to estimate the distance, as indicated in Figure 3.16. RSSI is a measurement of how well your device can detect a signal from another device. It's a value that is useful to indicate whether you have sufficiently strong signal to build a good wireless connection. DBm and RSSI are different indicators for signal strength. The difference is that RSSI is a relative value based on the signal strength drop during the transmission [63]. The equation for RSSI is:

$$RSSI = 10 * logP_r \tag{3.1}$$

where P_r is the received power, which depends on not only the received power, but also the frequency of wireless signals. In our experiments, we fixed the signal sent from Xbee/Zigbee module to make a constant frequency. Then, the received signal strength is decided by only the distance from receiver and transmitter. A linear equation was used to establish the relationship between

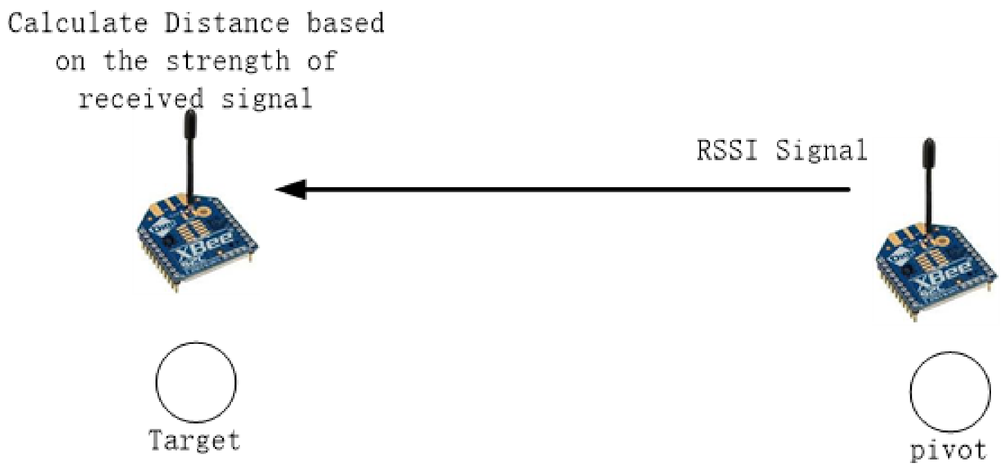


Figure 3.16: RSSI based three-point localization method (RTPL) diagram

distance and RSSI signal.

$$RSSI = a * distance + b \tag{3.2}$$

Parameters a and b in the linear equation can be calibrated by several experiments with known distance and RSSI values.

3.3.2 U-blox GPS with Gaussian Filter

In this section, we introduce U-blox GPS [2] and how to further improve its accuracy through Gaussian filter. A picture of U-blox GPS is shown in Figure 3.17. The baud rate for U-blox GPS is 9600 $bits/s$ and each GPS data contains 256 bits, which means we are able to receive 37 GPS data per second. Considering that irrigation machine moves slowly (the maximum speed is 4 $inch/s$), we decide to position in every two seconds based on 74 raw GPS data.

A Gaussian filter [10] is applied on each group of 74 GPS data to improve the accuracy. Each GPS data is composed by longitude and latitude, which are denoted by x and y, respectively. We denote the average values among a group of 74 GPS data by $(\bar{x}, \bar{y})$. In our design, we use 2-D Gaussian filter, where x-axis is $x - \bar{x}$ and y-axis indicates $y - \bar{y}$. The standard deviation of data in

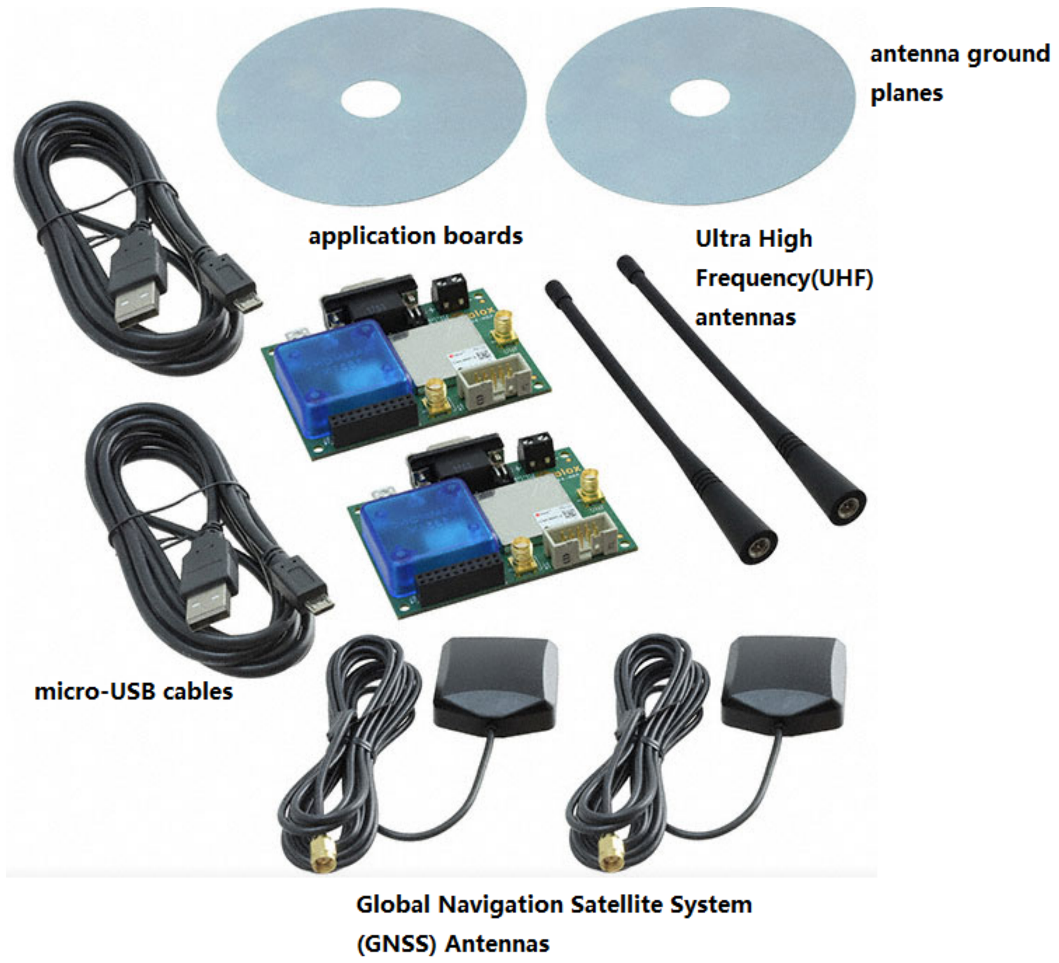

antenna ground planes
application boards
Ultra High Frequency(UHF) antennas
micro-USB cables
Global Navigation Satellite System (GNSS) Antennas

Figure 3.17: U-blox GPS package, adapted from [2]

a group is represented by σ. If the GPS data satisfies condition:

$$(x - \bar{x})^2 + (y - \bar{y})^2 > 4\sigma^2 \tag{3.3}$$

This data is treated as an outlier and pruned out.

3.3.3 Experiment Results

Some experiments were conducted to compare the accuracy of different positioning methods. For TTPL and RTPL, 3 different locations were tested, and 5000 data samples were obtained for each location. The three pivots for different locations are identical. For U-blox GPS, 7400 data for

Table 3.2: Error comparison between three different positioning techniques

	Average error (m)	Maximum error (m)	STD(m)
TTPL	11.32	19.03	1.27
RTPL	9.35	21.44	1.53
U-blox without filter	0.44	1.26	0.24
U-blox with filter	0.36	0.98	0.22

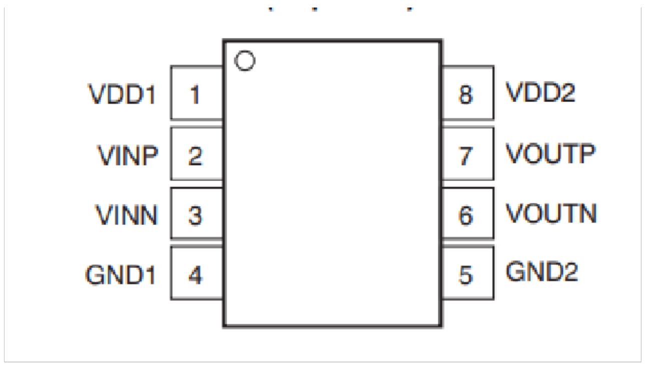
three different locations are tested and every 74 data are packed as a set (totally 100 sets for each location). The average error, maximum error and error stand deviation (STD) of TTPL, RTPL and U-blox with/without Gaussian filter are showed in Table 3.2.

For U-blox GPS with Gaussian filter, the final latitude and longitude values of irrigation machine is decided by the median, instead of average, from GPS data set after filtering noise. According to the result in Table 3.2, the error of U-blox with filters is one order of magnitude less than TTPL and RTPL methods. The Gaussian filter can reduce the average error by 18% and the maximum error can be reduced to less than 1 meter.

3.4 Voltage Monitoring System

The electric motor of the center pivot machine is powered by a 3-phase 480V AC(Alternating Current) supply. The peak voltage of each phase can be 600V. It is important to ensure that the supply voltage is within a legitimate range for the sake of safety.

We propose a AMC1100 [64] based power monitoring system which samples the real-time voltage status and sends an alarm to the central controller to shut down the whole system in emergency. AMC1100 is a AC amplifier. Its input voltage range is between -250mV and 250mV and its output voltage ranges in $0-5$V . Two VDDs, VDD1 and VDD2 indicated in Figure 3.18, are needed and both of them are 5V. We choose AMC1100 for its very low non-linearity, low offset error, low noise and relatively fixed gain. The AMC1100 based Printed Circuit Board (PCB) board was designed according to the voltage(480V AC) provided for irrigation system, as shown in Figure 3.19. The values of resistors R_1,R_2 and R_7 are then calculated to make the maximum voltage on R_7 not greater than 250mV. Resistor–Capacitor (RC) filters are used at both input and



GND1	4	Power	High-side analog ground
GND2	5	Power	Low-side analog ground
VDD1	1	Power	High-side power supply
VDD2	8	Power	Low-side power supply
VINN	3	Analog input	Inverting analog input
VINP	2	Analog input	Noninverting analog input
VOUTN	6	Analog output	Inverting analog output
VOUTP	7	Analog output	Noninverting analog output

Figure 3.18: AMC1100 pin descriptions, adapted from [3]

output sides to reduce the noise. Considering that the control board Arduino Uno has only one 5V and another 3.3V voltage supply while both VDDs on AMC1100 need 5V input, a DC(Direct current)/DC converter [65] is used to convert the 3.3V voltage supply from Arduino Uno to 5V which can be provided to VDD on AMC1100.

In a three-phased voltage system, three AMC1100 based PCB boards are embedded for voltage monitoring of each phase. They are all connected to the ADCs on Arduino Uno board. The Uno board also stores historical data. When the voltage becomes abnormal, the Arduino board can detect that from its ADCs outputs and send an alarm to the central controller. Central controller receives the alarm and shuts down the voltage supply through the control pin of SSR (Solid-State Relay). A diagram for the voltage monitoring system is shown in Figure 3.20.

3.5 Conclusions

With the advance of computer techniques, precise irrigation becomes possible and preferred. A reinforcement learning based irrigation algorithm was designed for a center-pivot irrigation system.

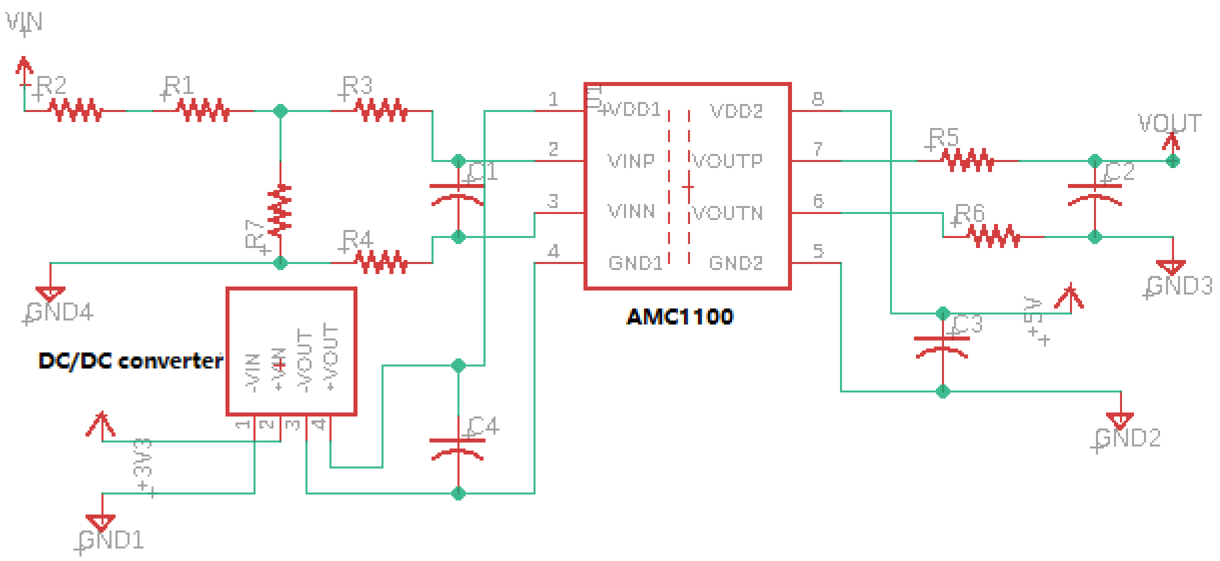


Figure 3.19: AMC1100 based PCB design

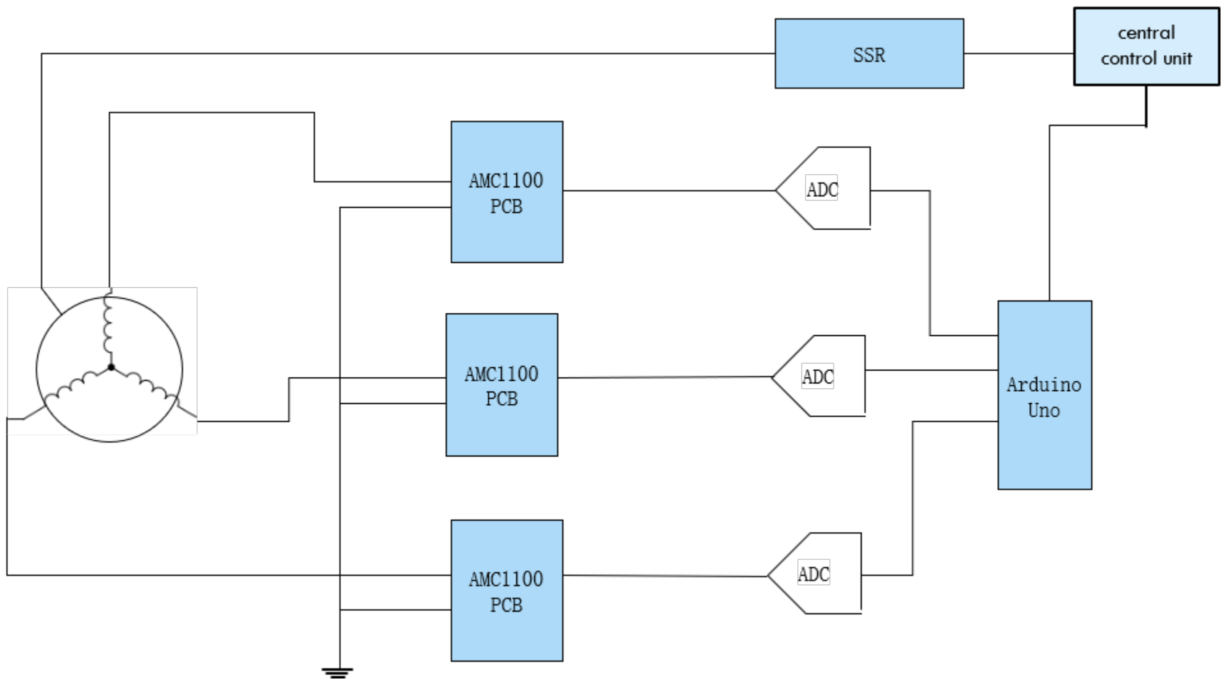


Figure 3.20: Voltage monitoring system

In order to realize this algorithm, customized hardware devices are embedded into the control part. In this work, different positioning schemes have been designed and tested, which finally provided a positioning system with the maximum error under one meter. A power monitoring system is then introduced to shut down the system from abnormal power supply and improve the safety of the system.

4. FAST PLACEMENT FOR FPGA-BASED INFORMATION PROCESSING HARDWARE

In this chapter, we will introduce fast FPGA placement techniques for designs with regularity.

4.1 Background

4.1.1 Field Programmable Gate Arrays (FPGA)

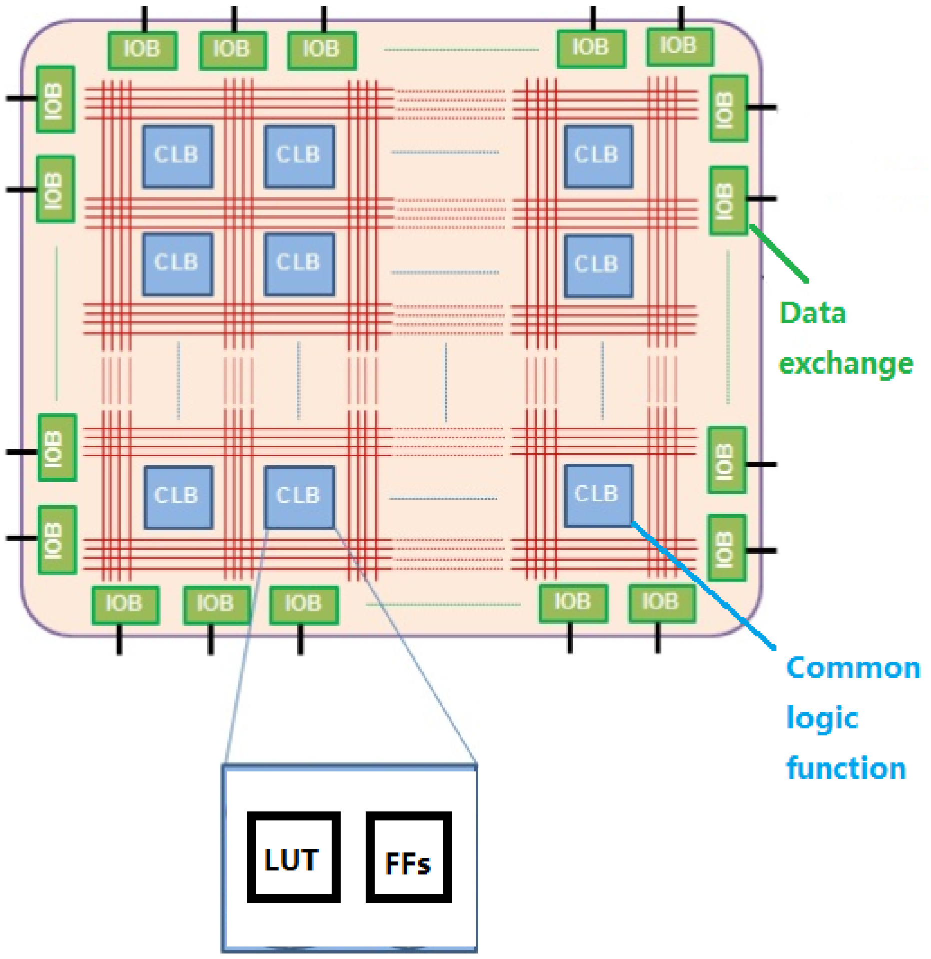


Figure 4.1: Internal architecture of a typical FPGA, adapted from [4]

FPGA [66] [67] is a type of integrated circuits, where basic circuit elements have been pre-fabricated and functionalities can be realized by programming its memory and wire connections.

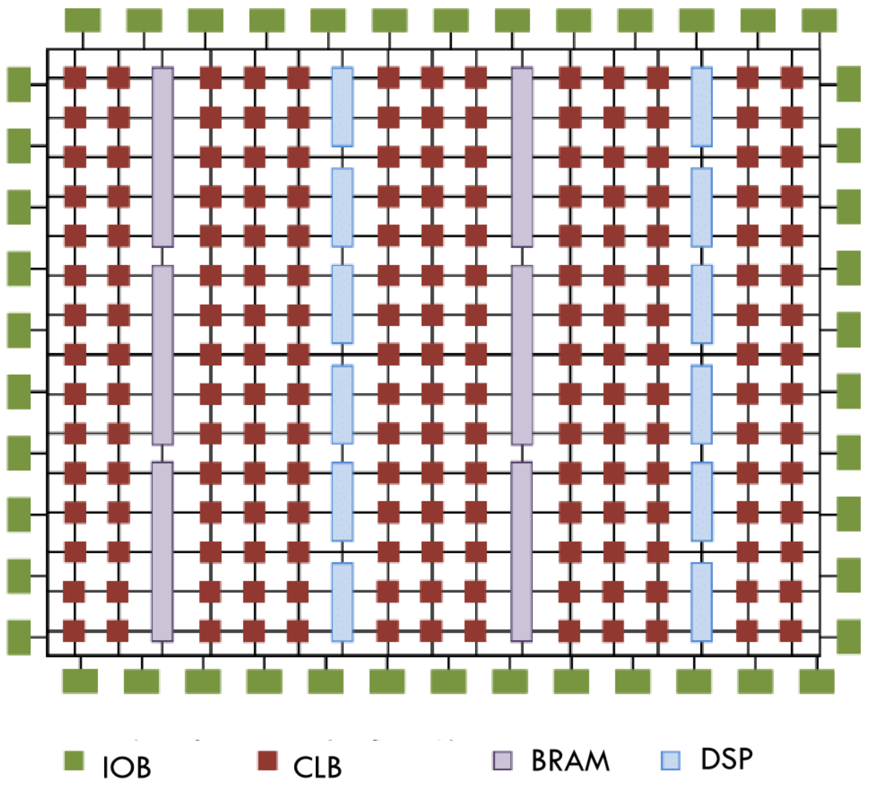


Figure 4.2: Modern FPGA architecture, adapted from [5]

A key component for FPGA is CLB (Configurable Logic Block), where the majority of logic functions are implemented. Figure 4.1 illustrates an FPGA architecture with an array of CLBs and IO blocks. Within a CLB, a small combinational logic function can be implemented by a LUT (Lookup Table). Besides CLBs, modern FPGA often contains BRAM (Block Random Access Memory) and DSP (Digital Signal Processing) blocks. An overview of such an architecture is shown in Figure 4.2.

4.1.2 FPGA Design Flow

The diagram in Figure 4.3 shows the design flow on FPGA. There are four main steps [68]: (1) Design specification and entry, (2) Synthesis, (3) Implementation and (4) Programming to FPGA.

A brief FPGA design flow is shown in Figure 4.3. Circuit description and constraint specification: In this stage, FPGA architecture is selected according to its available resources. The circuit design is described in HDL (Hardware Description Language). Design constraints are captured in

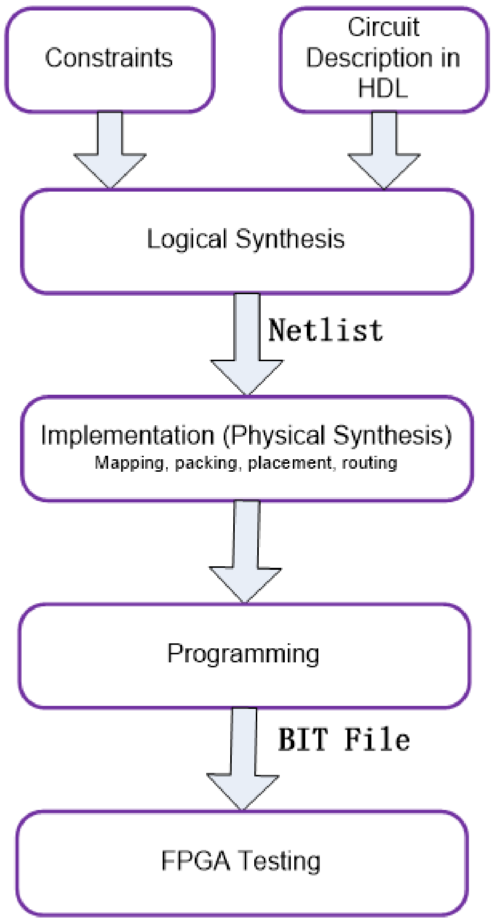

Constraints
Circuit Description in HDL
Logical Synthesis
Netlist
Implementation (Physical Synthesis)
Mapping, packing, placement, routing
Programming
BIT File
FPGA Testing

Figure 4.3: Deisgn flow on FPGA

XDC files.

Synthesis: The FPAG design tool transforms HDL code into a gate-level netlist.

Implementation: This stage contains four smaller steps, mapping, packing, placement and routing. Mapping divides the whole circuit into basic elements so that they can be fit into FPGA. Packing, also known as clustering, is to combine the LUTs and flip-flops together to form logic blocks. Placement determines which logic block within an FPGA should implement each of the logic blocks we get from packing. Routing is to connect the blocks.

Programming: When the design need to be loaded on the FPGA, it must be converted to a format that the FPGA can accept it. Some programming tools deals with the conversion from layout to bitstream.

Among all the four steps, implementation consumes most of the time. Placement and routing are the most time-consuming steps in implementation. That's why we focus on the design of regularity-aware placement methods for speed up.

4.1.3 FPGA Placement

Placement is to decide locations for circuit elements or map them to physical resources on an FPGA chip. In general, there are three categories of FPGA placement algorithms [69]: 1) min-cut-partitioning-based methods, 2) analytical techniques and 3) simulated-annealing-based approaches,

Simulated annealing starts with a random feasible solution and makes iterative moves to improve the solution quality. For FPGA placement, a move can be a swap between the locations of two circuit blocks or moving a block into an empty space. Whether or not a move is accepted is decided by the evaluation of a cost function, which is often wirelength for FPGA placement. A move that reduces the cost is always accepted. A move with small increase of the cost is accepted with a certain probability, which is inversely proportional to the cost increase. Accepting cost increases allows the algorithm to have a chance of hill climbing and therefore avoid being trapped into a local optimal solution.

4.1.4 Half-Perimeter Wirelength Model

The cost function in simulated annealing-based FPGA placement is wirelength and one of the wirelength models we use is half-perimeter wirelength (HPWL) [70]. It is one of the most popular models for FPGA placement. HPWL for one net is equal to a half of the perimeter of the smallest bounding rectangle, as shown in Figure 4.4. For a net with two or three pins, the routing cost estimated by HPWL is near accurate. However, for nets with 4 pins or more, indicated in Figure 4.4, HPWL tends to underestimate the wirelength. An factor q was introduced to compensate for the underestimate. The value of q depends on the pins of a net. It slowly increases from 1.08 for nets with 4 pins to 2.79 with 50 pins. For large fan-out nets which have more than 50 pins, q increases linearly as shown in the following equation:

$$q = 2.7933 + 0.02616 \times (\#pins - 50) \tag{4.1}$$

The detailed cost function with HPWL wirelength model is:

$$Cost = \sum_{net} q(i) \times max_{i,j \in net}(|x_i - x_j| + |y_i - y_j|) \tag{4.2}$$

Where $|x_i - x_j$ and $|y_i - y_j|$ calculates bounding box width and height and $q(i)$ indicates the correction factor.

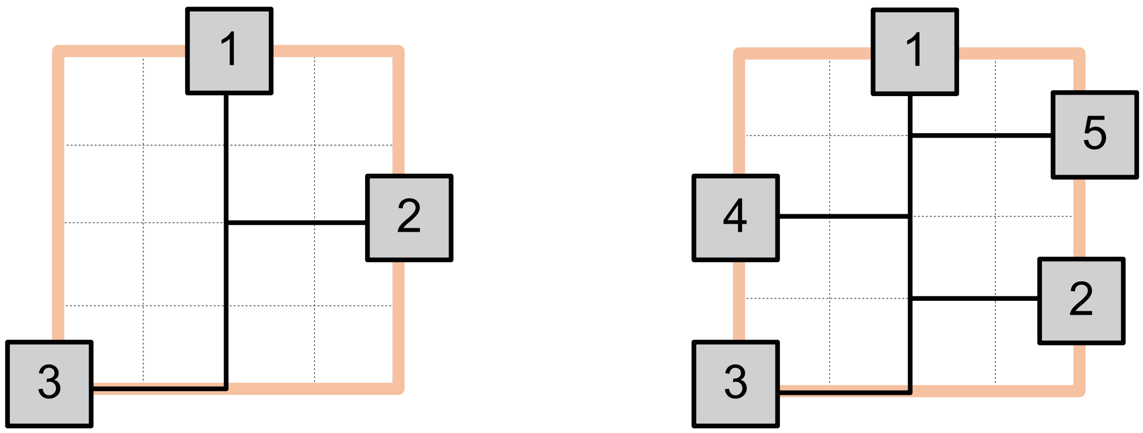


Figure 4.4: For a 3-pin net, HPWL=8 and wirelength=8. For a 5-pin net, HPWL=8 and wirelength=12

4.2 Previous Work

A classical approach to FPGA placement is simulated annealing, which is represented by the famous academic tool VPR [15, 16]. It is a very flexible framework that works for almost any objective function and does not rely on special problem structure. Its main drawback is the slow convergence for large problems. Therefore, later simulated annealing-based FPGA placement methods [19, 21, 24, 28] mostly focus on its computing acceleration. In particular, 10× speedup is achieved by GPU-based parallelism [21] with no more than 3% increase of wirelength. Placement runtime and solution tradeoff were studied in [18], where the solution degradation can be as much as 33%.

Recently, analytical approach has been gaining popularity for FPGA placement [20, 22, 23, 25–27] due to its capability of handling large designs. Packing, which is used to be a standalone step prior to placement [16, 26], is integrated with placement in [27] to improve solution quality. Placement objectives also shift from minimizing wirelength [16, 20] to additionally considering routability [25–27].

There are very few works on FPGA placement considering regularity. One example [29] is for regular datapath placement. However, the regular arrays here are at the granularity of Configurable Logic Blocks (CLBs) and thus the corresponding placement is trivially laying out in a CLB array. By contrast, a PE in a neural network usually covers multiple CLBs and such PE array placement becomes non-trivial at all. Systolic array placement on FPGA is investigated in [31]. However, its main focus is on the utilization of DSP and RAM blocks instead of regularity. High level design reuse for modular designs is introduced in RapidWright [30] and is a complement to our algorithm level regularity. For example, the work of [31] achieves reuse and replication of SLRs (Super Logic Region) through RapidWright [30]. An SLR is a complete chip die, which is a much coarser grained than our PE-level regularity, where a PE covers around 10 CLBs. While the regularity in [30, 31] is realized in a copy-and-paste manner, our middle-grained regularity is more complicated to handle.

4.3 Regularity-Aware FPGA Placement

4.3.1 Overview of the Proposed Methodology

The input is a packed netlist corresponding to CLBs, Random Access Memory (RAM) blocks, Digital Signal Processors (DSPs) and IO pads, which need to be mapped to a given FPGA architecture. We consider circuit designs that contain a regular PE array besides non-regular elements or control logic. In general, a PE consists of multiple CLBs and this is particularly true for neural network circuits. The objective is to minimize total wirelength while all elements are placed legally. In addition, circuits in two different PEs are placed in the same way, i.e., the placement solutions of two PEs are identical. Last but not the least, PEs are placed in a geometrically regular array.

The regularity-aware placement includes two phases:

- Phase I: ILP (Integer Linear Programming)-based PE array placement.
- Phase II: Accelerated simulated annealing taking the PE array placement as a part of the initial solution.

In general, there are four types of wire connections to be considered.

1. Intra-PE nets. These are nets connecting blocks within a PE.
2. Inter-PE nets. These are nets involving connections among different PEs.
3. PE-external nets. These are nets involving connections between a PE and elements outside of the PE array.
4. External nets. These are nets that are completely outside of any PEs.

Phase I is focused on minimizing wirelength of intra-PE and inter-PE nets while phase II considers all types of nets.

4.3.2 Phase I: ILP-based PE Array Placement

The input of phase I is an $m \times n$ array of PEs $\mathcal{P} = \{\pi^{1,1}, \pi^{1,2}, ..., \pi^{1,n}, \pi^{2,1}, ..., \pi^{m,n}\}$. Each PE $\pi^{i,j} \in \mathcal{P}$ consists of k blocks $\pi^{i,j} = \{b_1^{i,j}, b_2^{i,j}, ..., b_k^{i,j}\}$. Figure 4.5 shows an example for $m = n = k = 3$. In phase I, the $m \times n$ array structure is retained as layout pattern as well. The placement here is mostly to decide block locations within a PE and the same block placement pattern is shared by all PEs. For example, in Figure 4.5, the placement of the 3 blocks in a PE is shared by all the 9 PEs. The placement of a block $b_l^{i,j}$ is indicated by two decision variables, column index $x_l^{i,j}$ and row index $y_l^{i,j}$ in a given FPGA architecture. Due to the regularity, the overall number of decision variables is reduced from $2m \cdot n \cdot k$ to the order of $2k$. The large reduction makes ILP, which is computationally expensive, a practical component in the proposed methodology. For neural network circuits, the value of k is typically around a dozen.

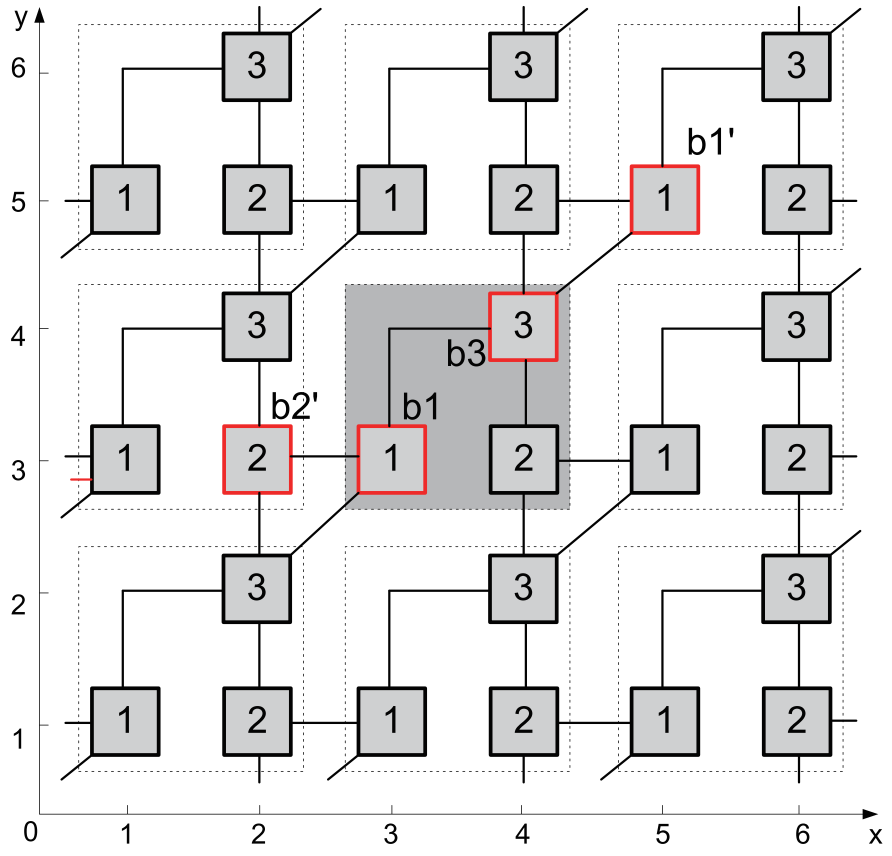


Figure 4.5: An example of PE array placement. Each dotted box is a PE and and the shaded one is the reference PE. Each small square is a block.

Since virtually only the placement of one PE needs to be decided, one PE $\pi_{ref} = \{b_1, b_2, ..., b_k\}$ that is not adjacent to the array boundary is designated as the reference PE for this mission. For the sake of brevity, the PE row/column indices are skipped for the placement variables of the reference PE and its block locations are denoted as $\mathbf{x} = (x_1, x_2, ..., x_k)$ and $\mathbf{y} = (y_1, y_2, ..., y_k)$. For example, the location of block b_3 in Figure 4.5 is at $(x_3 = 4, y_3 = 4)$. Note that these decision variables must be integers in a legitimate range. The locations of the other PEs repeat the pattern of $(\mathbf{x}, \mathbf{y})$

with horizontal and/or vertical shift in terms of PE width and height given by:

$$W = \max_{l=1}^{k} x_l - \min_{l=1}^{k} x_l + 1, \quad H = \max_{l=1}^{k} y_l - \min_{l=1}^{k} y_l + 1$$

In Figure 4.5, $W = H = 2$. Please note the placement solution of phase I is an approximation as a uniform array is assumed for FPGA while an actual FPGA architecture is a mixture of CLBs, DSPs and RAM blocks.

The objective is to minimize Half Perimeter Wire Length (HPWL) for all intra-PE and inter-PE nets, e.g., intra-PE net (b_1, b_3), inter-PE nets (b'_1, b_3) and (b_1, b'_2) in Figure 4.5. The set of intra-PE nets $\mathcal{N}_{in}$ consists all nets that are completely within the reference PE. Each net in the set of inter-PE nets $\mathcal{N}_\xi$ spans at least one block in the reference PE and at least another block in a non-reference PE. We use a generic notation $\{b'_1, b'_2, ..., b'_k\}$ for indicating blocks in non-reference PEs, and vector pair $\mathbf{x}' = (x'_1, x'_2, ..., x'_k)$ and $\mathbf{y}' = (y'_1, y'_2, ..., y'_k)$ to represent their locations. Evidently, $x'_i = x_i + p \cdot W$ and $y'_i = y_i + q \cdot H$, where p and q are integers. The x and y components of the total intra-PE and inter-PE HPWL are denoted as L^x_{in}, L^y_{in}, L^x_ξ, and L^y_ξ, respectively. Then, the objective of the ILP is described by

$$\text{Minimize } \; L^x_{in}(\mathbf{x}) + L^y_{in}(\mathbf{y}) + L^x_\xi(\mathbf{x}) + L^y_\xi(\mathbf{y}) \tag{4.3}$$

The HPWL functions are to be elaborated through their x components as the y components work in the same way. The x component of intra-PE wirelength is defined by

$$L^x_{in}(\mathbf{x}) = \sum_{N_i \in \mathcal{N}_{in}} \max_{b_j \in N_i, b_l \in N_i, j \neq l} |x_j - x_l| \tag{4.4}$$

and one example is net (b_1, b_3) in Figure 4.5. The x component of inter-PE wirelength is defined by

$$L^x_\xi(\mathbf{x}) = \sum_{N_i \in \mathcal{N}_\xi} \max_{b_j \in N_i, b'_l \in N_i} \begin{cases} x'_l - x_j, & \text{if } p > 0 \\ x_j - x'_l, & \text{if } p < 0 \end{cases} \tag{4.5}$$

where $p > 0$ $(p < 0)$ indicates block b'_l is on the right (left) of the reference PE like block b'_1 (b'_2) in Figure 4.5. The case of $p = 0$ is trivial. Although an inter-PE net may contain two blocks b_j and b_l, both of which are in the reference PE, their distance $|x_j - x_l|$ can never exceed the maximum inter-PE distance in the same net, and therefore does not need to be included in Equation (4.5). The $\max$ operations in Equations (4.4) and (4.5) are nonlinear and not supported by ILP solvers in general. This problem can be solved by introducing assist variables and additional constraints. For instance, $\max(z_1, z_2, ...)$ can be replaced by a new variable u that satisfies $u \geqslant z_1, u \geqslant z_2, ...$ As the new variables are to be minimized in the optimization, they become tight upper bounds or the maximum values. The nonlinear absolute value operation $|x_j - x_l|$ in the objective function can be replaced by $\max(x_j - x_l, x_l - x_j)$ and handled in the same manner.

The constraints in the ILP are the following.

$$|x_j - x_l| + |y_j - y_l| \geqslant 1,\ j, l = 1, 2, ..., k,\ j \neq l \tag{4.6}$$

$$x_j, y_j \in \mathbb{Z},\ j = 1, 2, ..., k \tag{4.7}$$

$$x_{min} \leqslant x_j \leqslant x_{max},\ y_{min} \leqslant y_j \leqslant y_{max},\ j = 1, 2, ..., k \tag{4.8}$$

where $x_{min},\ x_{max},\ y_{min}$ and y_{max} are given constants. Constraint (4.6) is to ensure that two different blocks do not occupy the same place. Constraints (4.7) and (4.8) make the solutions to be integers in legitimate ranges.

Although the absolute value $|x_j - x_l|$ in constraint (4.6) can be transformed to $\max(x_j - x_l, x_l - x_j)$, the $\max$ operation here cannot be simply replaced by an upper bound variable u like in handling $|x_j - x_l|$ for the objective function (4.4), which is to be minimized. Such upper bound variable would not be minimized in a constraint and hence cannot be tight as the maximum value.

We devise a technique to solve this problem and use operation $|z|$ as an example. This is a specific technique for an integer z in the context of ILP. A sufficiently large constant B is introduced as well as two additional integer variables e and $\hat{z}$. The value of $\hat{z}$ is forced to be equal to $|z|$ by adding the following constraints in the ILP formulation.

$$z + Be \geqslant 0 \tag{4.9}$$

$$-z + B(1 - e) \geqslant 0 \tag{4.10}$$

$$-z - B(1 - e) \leqslant \hat{z} \leqslant z + Be \tag{4.11}$$

$$z - Be \leqslant \hat{z} \leqslant -z + B(1 - e) \tag{4.12}$$

$$e \in \{0, 1\}$$

$$\hat{z} \in \mathbb{Z}$$

When $z > 0$, constraints (4.9) and (4.10) in the ILP solver would force $e = 0$. Then, Equations (4.11) and (4.12) become

$$-z - B \leqslant \hat{z} \leqslant z \tag{4.13}$$

$$z \leqslant \hat{z} \leqslant -z + B \tag{4.14}$$

which can be combined to $\hat{z} = z$.

When $z < 0$, constraints (4.9) and (4.10) in the ILP solver would force $e = 1$. Then, Equations (4.11) and (4.12) become

$$-z \leqslant \hat{z} \leqslant z + B \tag{4.15}$$

$$z - B \leqslant \hat{z} \leqslant -z \tag{4.16}$$

which can be combined to $\hat{z} = -z$. Therefore, $\hat{z} = |z|$.

Since the decision variables can be reduced to the positions of the blocks in the reference PE, the computational complexity of the ILP approach is decided by k, which is the number of blocks within a PE, and does not increase with the number of PEs ($m \cdot n$) in a given array.

4.3.3 Phase II: Customized Simulated Annealing

Phase II takes the PE array placement solution from phase I as part of its initial solution, and places the other blocks, which are not PEs, randomly in the initial solution. Then, simulated annealing is performed with acceleration by exploiting the regularity. The objective is to minimize total wirelength including intra-PE, inter-PE, PE-external and external wirelength. At the same time, the placement solution must be legal and does not have any overlap. In addition, the regular array structure is retained in PE placement. There are two kinds of moves in the simulated annealing: (1) moving a block to an empty legal space; (2) swapping the locations of two blocks of the same kind in terms of CLBs, RAMs and IO pads.

The simulated annealing here is considerably different from the conventional one due to the regularity and our customization. First, a move of a block in one PE must be repeated for the same corresponding blocks in the other PEs in an identical manner. Second, moves of PE blocks and non-PE blocks are searched with different levels of effort. Although the PE array placement in phase I is an approximation due to the neglection of PE-external nets and the assumption of homogeneous architecture substrate, its intra-PE and inter-PE wirelength has already been heavily optimized and the overall solution is far better than a random placement. As such, we explore much less number of moves for PE blocks than non-PE blocks. Moveover, the annealing temperature for PE block moves is set to be lower than that of non-PE blocks as only small changes are sufficient in general.

We notice that a very time consuming part of the simulated annealing is the cost function (total wirelength) estimation by VPR, which evaluates the impact of FPGA routing architectures. Based on this observation, we develop three techniques for further runtime reduction, and the tradeoff between runtime and solution quality: (1) speculative decision; (2) wirelength calculation omission; (3) hybrid wirelength model, which are to be elaborated as follows.

4.3.3.1 *Speculative Decisions for PE Block Moves*

One observation is that a block is preferred to be placed in boundary regions of a PE if it has many wire connections with circuit elements outside of the PE. Likewise, a block whose wire connections are mostly inside the PE is preferred to be placed in the middle of the PE. Therefore, we can sometimes quickly evaluate a simulated annealing move by examining the internal and external connections of PE blocks without the time consuming estimation of cost function, which is the wirelength computation considering FPGA routing architecture. Note that this does not conflict with the wirelength calculation omission, which is to be described in Section 4.3.3.2. The two techniques are applicable for different scenarios. The concept of internal/external connections has overlap with intra-PE/inter-PE nets defined in Section 4.3.1, but is different. A net with h sinks is counted as h *connections* and internal connections are all within a single PE. External connections of a block b_i include connections with non-PE blocks in addition to inter-PE connections.

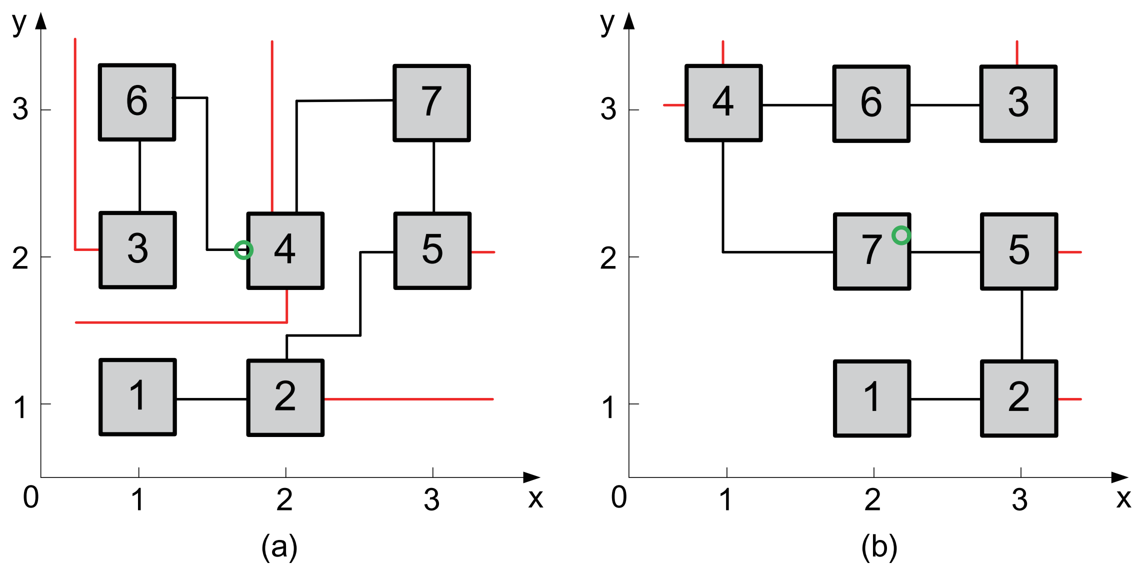


Figure 4.6: Examples for calculating favorability score. Green circles indicate center of gravity. Black and red lines indicate internal and external connections, respectively.

The key ingredient for the internal/external connection-based speculation is a *favorability score* elaborated as follows. We reuse the concept of reference PE π_{ref} defined in Section 4.3.2 and assume that there are k blocks in each PE. Given a PE placement solution, the Center Of Gravity (COG) of its blocks is represented by $(\hat{x}, \hat{y})$, and the distance from COG to each block $b_i \in \pi_{ref}$ is denoted as d_i. Then, the favorability score is defined as

$$\phi = \sum_{i=1}^{k} d_i \cdot \frac{E_i}{I_i}$$

where I_i and E_i indicate the numbers of internal and external connections for block b_i, respectively. The score is high and favorable when a block with high E_i/I_i is far from the COG.

Examples of the favorability score calculation and its effect are illustrated in Figure 4.6. In Figure 4.6(a), the COG is at $(\frac{13}{7}, 2)$. Its favorability score is obtained by $\frac{E_2}{I_2} \times d_2 + \frac{E_3}{I_3} \times d_3 + \frac{E_4}{I_4} \times d_4 + \frac{E_5}{I_5} \times d_5 = \frac{1}{2} \times 1.143 + \frac{1}{1} \times 0.857 + \frac{2}{2} \times 0.143 + \frac{1}{2} \times 1.143 = 2.143$. The favorability score for the placement in Figure 4.6(b) is estimated to be 5.071 through the same calculation procedure. One can see that placement in Figure 4.6(b) is more preferred than Figure 4.6(a).

Algorithm 1: Speculative decision for PE block move

```
   Input: PE placement solution after trial move $\mu$;
          $I_i, i = 1, 2, ..., k$: #internal connections;
          $E_i, i = 1, 2, ..., k$: #external connections;
          $\tilde{\phi}$: favorability score before the move;
          $\theta_L$ and $\theta_H$: constant thresholds and $\theta_L < \theta_H$.
1  PE center of gravity $\hat{x} = \sum_{i=1}^{k} x_i/k$, $\hat{y} = \sum_{i=1}^{k} y_i/k$;
2  for each block $b_i \in \pi_{ref}$ do
3      $d_i = |x_i - \hat{x}| + |y_i - \hat{y}|$;
4  $\phi = \sum_{i=1}^{k} d_i \cdot \frac{E_i}{I_i}$; // Favorability score
5  if $\phi < \theta_L$ then
6      Reject $\mu$;
7  else if $\phi > \theta_H$ and $\tilde{\phi} \leqslant \theta_H$ then
8      Commit $\mu$;
9  else
10     Evaluate wirelength to commit or reject $\mu$;
```

The speculative decision procedure is summarized in Algorithm 1. In step 1, the COG of the given placement solution after a trial move μ is calculated. The favorability score is obtained through steps 2-5. If the score is very low, which indicates a poor solution, move μ is directly rejected in step 7 without computing the cost function of simulated annealing. If the score is high while the solution prior to the move is not good enough, move μ is directly committed in step 10 without computing the cost function. Only in those unclear cases, the cost function is evaluated and a normal simulated annealing decision is made in step 13.

In our experience, the speculation is correct in most of time, but not always. Therefore, it can accelerate overall computing with a chance of small solution degradation.

4.3.3.2 Wirelength Calculation Omission for Repeating Patterns

This is a simple yet very effective technique that exploiting the repeating layout patterns in a PE array. It is illustrated with the example in Figure 4.7. There are 4 PEs $\pi^{1,1}$, $\pi^{1,2}$, $\pi^{2,1}$ and $\pi^{2,2}$. Due to the regularity, nets $(b_1^{1,1}, b_3^{1,1})$, $(b_1^{1,2}, b_3^{1,2})$, $(b_1^{2,1}, b_3^{2,1})$ and $(b_1^{2,2}, b_3^{2,2})$ are identical. Similarly, nets $(b_2^{1,1}, b_1^{1,2})$ $(b_2^{2,1}, b_1^{2,2})$ are the same. When block b_1 is moved upward from Figure 4.7(a) to (b), only the wirelength update for the reference PE ($\pi^{1,2}$) (red color in Figure 4.7(b)) needs to be actually calculated. The other wirelength changes (blue color in Figure 4.7(b)) can be taken from that of the reference PE and the calculation can be omitted. Although this technique is simple, it allows purely computing acceleration without any solution quality loss.

4.3.3.3 Hybrid Wirelength Model

Besides HPWL, net wirelength can be estimated more accurately by considering FPGA routing architecture and VPR provides such pre-routing wirelength estimation function. However, calling this function costs more runtime than HPWL estimation. Therefore, there is runtime-accuracy tradeoff between HPWL and FPGA-aware wirelength estimation. Our idea is to use HPWL in early iterations of simulated annealing when an approximate global solution is searched and the FPGA-aware wirelength model is employed in later iterations when the solution search is mostly to refine details.

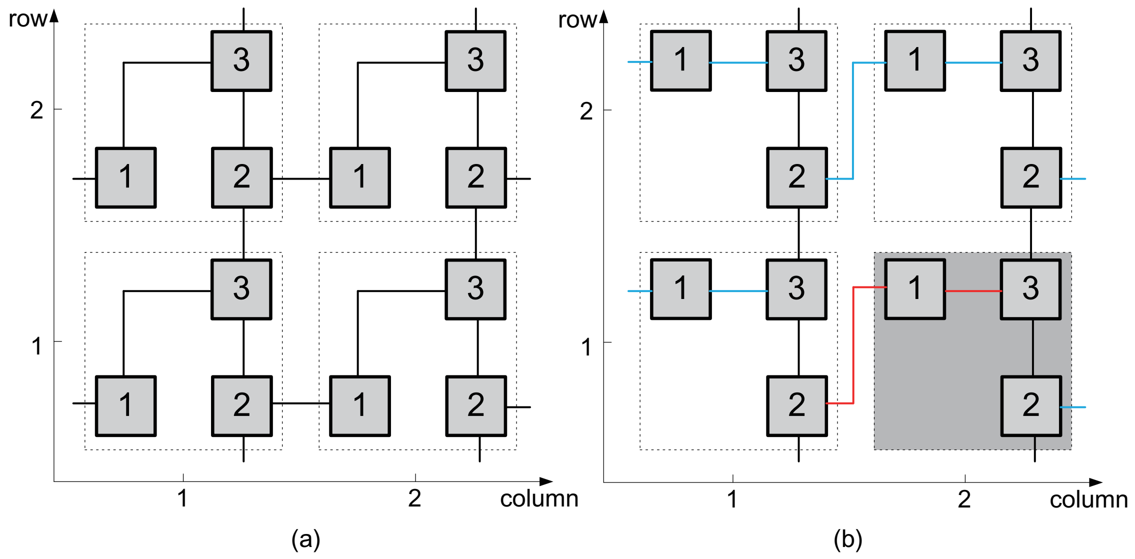


Figure 4.7: When block b_1 is moved from (a) to (b), only wirelength associated with the reference PE (in red color) needs to be calculated, as the other wire patterns (in blue color) are the same.

4.4 Experimental Evaluations

4.4.1 Testcase Design

Conventional benchmark circuits for FPGA placement are not oriented toward designs with regularity. Hence, we designed a set of circuits with regularity as the testcases in the experiment. Here our choice on the regularity format is the systolic array, which is a widely used architecture topology in many important application domains, such as machine learning and digital signal processing. More specifically, considering that neural network circuit is currently the most representative and popular systolic array-based hardware application, we prepare testcases with focus on different types of systolic array-based neural network circuits, including Convolutional Neural Network (**CNN**) for computer vision, Multi-Layer Perceptron (**MLP**) for speech recognition, and Recurrent Neural Network (**RNN**) for natural language processing.

For each testcase, the systolic array consists of m-by-m PEs, where m is set as 4, 8, 16, 32, and 64 for each neural network type. Figure 4.8 is an example of 4×4 systolic array, where the top and left Random Access Memories (RAMs) are input RAMs, and the right RAMs are output RAMs. As a reference, the systolic array in Google TPU [37] design is 256×256. Each PE consists

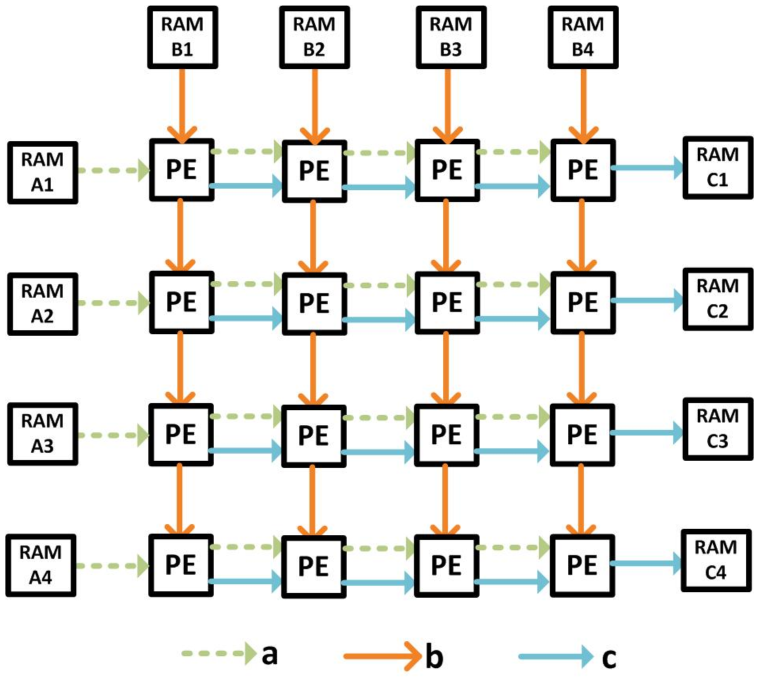


Figure 4.8: An example design of 4×4 systolic array.

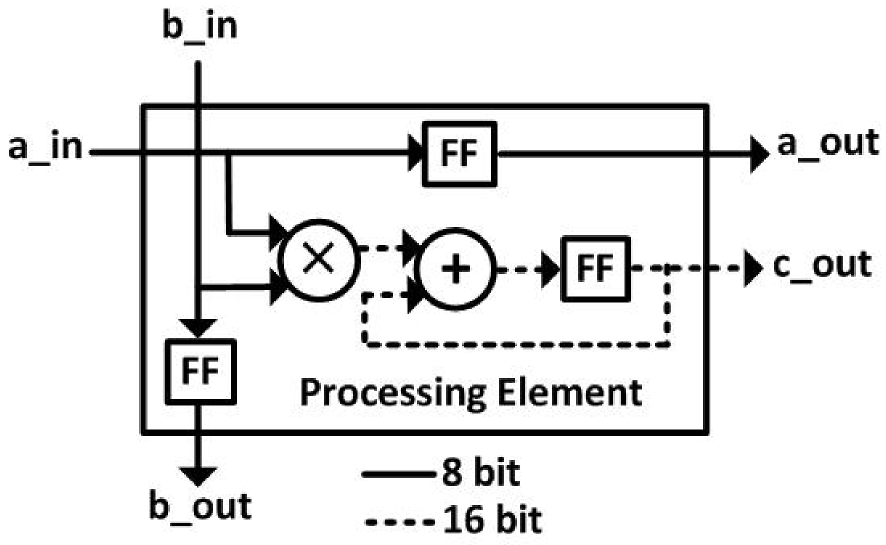


Figure 4.9: One example design of a PE.

of one 16-bit multiplier, one 16-bit adder, two 8-bit registers and one 16-bit register (as seen in Figure 4.9). Notice that as the size of systolic array scales, the overall circuit complexity increases rapidly. For instance, a 64×64 systolic array testcase contains 4096 multipliers, 4096 adders and 12288 registers, which is a relatively large-size design from the perspective of FPGA placement. Our testcase designs include 15 different neural network circuits with different PE array sizes.

4.4.2 Execution Flow

Given a Verilog file, logic synthesis is performed and verified in ODIN II [71]. VQM [72] and Quartus [40] are used to instantiate RAM blocks described in Verilog. The FPGA architecture is based on "$stratixiv\_arch.timing.xml$", which is used in VPR $Titan$ benchmarks. This architecture is close to a realistic one in the Altera Stratix FPGA Family, which has been applied for large circuit designs. The packing is performed and adjusted in VPR [16], after which our placement performed. The ILP solver used in our method is Gurobi 9.01 [73]. Finally, VPR takes our placement results for its routing and then evaluates post-routing wirelength as well as circuit timing.

The utilization for RNN circuits after packing is summarized in Table 4.1, where LUT represents look-up table, and FF indicates flip-flop. Note that VPR supports customizing the size of an FPGA, so we use the absolute number of LUTs and FFs as the metric for FPGA utilization instead of the percentage. The utilizations of CNN and MLP circuits are similar. The circuit size for 64×64 systolic array is pretty large, but is still practical considering that high-end industrial FPGA product may contain more than three millions of LUTs [74].

Table 4.1: FPGA utilization of RNN circuits.

Array Size	#LUT	#FF
4×4	7874	464
8×8	35087	1920
16×16	160160	8192
32×32	576450	33280
64×64	2031106	135509

4.4.3 Runtime Improvement

Our placement method is implemented in C language. The experiments are performed on a Linux work station with an Intel 3.70GHz i5-9600K core and 16GB RAM. Our placement is compared with VPR 8.0.0 [16], which is perhaps the most influential academic tool. The results are showed in Table 4.2. The speedup from our method increases with PE array size and more than $25\times$ acceleration is achieved for neural networks with 64×64 systolic array. This speedup implies that the placement runtime for the large cases is reduced from about 13 hours to a little more than a half hour. It is significantly more than the $7\times$ speedup obtained by an analytical approach [23]. Since runtime reduction is mostly needed for large designs, the increasingly large speedup is very appealing.

Table 4.2: Results on placement runtime. (RNN: recurrent neural network; CNN: convolutional neural network; MLP: multilayer perceptron.)

Cases	Array Size	RAM Size	Runtime (s)		
			VPR	Ours	Speedup
RNN	4×4	256KB	318	127	2.5×
RNN	8×8	256KB	995	194	5.1×
RNN	16×16	256KB	2938	288	10.2×
RNN	32×32	256KB	9867	514	19.2×
RNN	64×64	256KB	46828	1653	28.3×
CNN	4×4	256KB	347	162	2.1×
CNN	8×8	256KB	1087	226	4.8×
CNN	16×16	256KB	3233	327	9.9×
CNN	32×32	256KB	9894	481	20.6×
CNN	64×64	256KB	50327	1792	28.1×
MLP	4×4	512KB	582	343	1.7×
MLP	8×8	512KB	1662	404	4.1×
MLP	16×16	512KB	4214	487	8.3×
MLP	32×32	512KB	10253	585	17.5×
MLP	64×64	512KB	51197	2014	25.4×

4.4.4 Solution Quality and Correlations

The routing for all the placement solutions can be easily completed by VPR. The post-routing wirelength and circuit frequency results are shown in Figure 4.10. These results are normalized using VPR results as the baseline. On average, our method results in 6.9% wirelength increase and 6.8% circuit frequency decrease. A main reason for the degradation is from the dead space caused by maintaining regularity. Without the regularity constraint, VPR is able to find more compact solutions. Therefore, this degradation is a price paid for the regularity-based acceleration and difficult to be avoided.

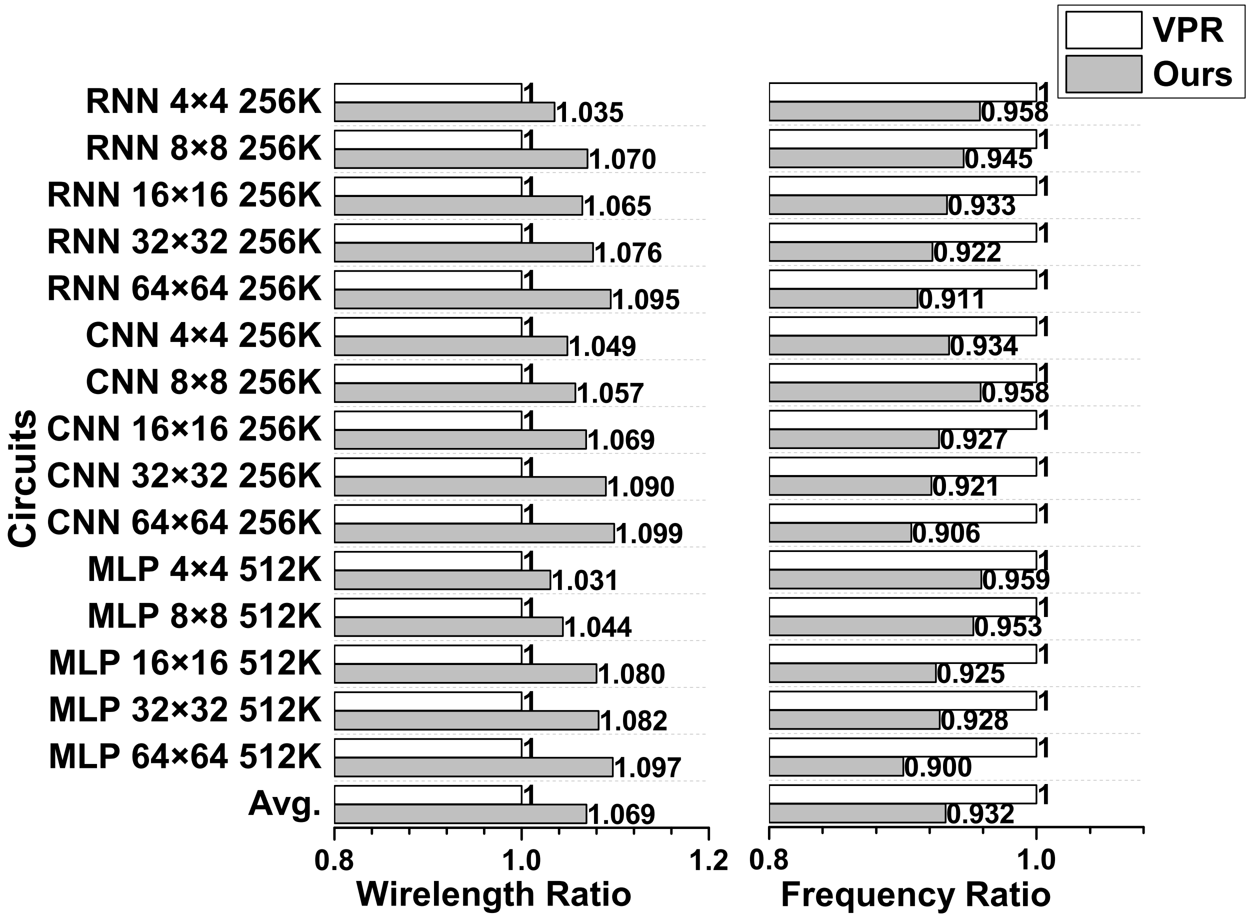


Figure 4.10: Normalized wirelength (left) and frequency (right) from solutions of our placer and VPR.

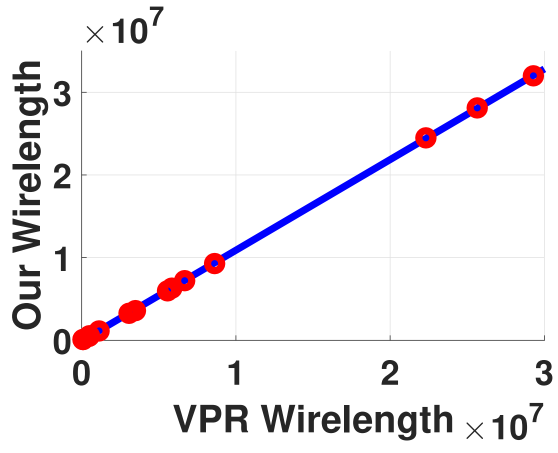


Figure 4.11: Wirelength correlation between our solutions and VPR's, correlation coefficient 1.0000.

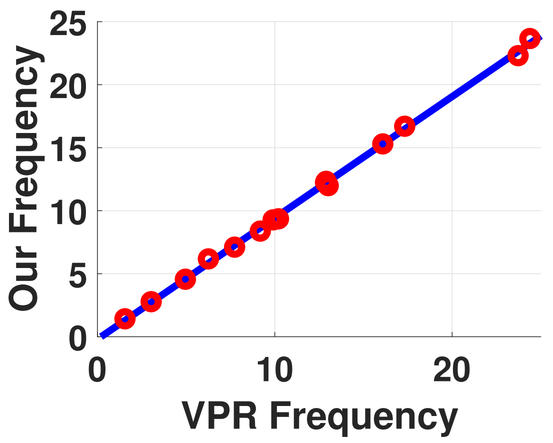


Figure 4.12: Frequency correlation between our solutions and VPR's, correlation coefficient 0.9994.

We also investigate the correlation between our placement and VPR. The wirelength and frequency correlation results are shown in Figure 4.11 and Figure 4.12, respectively. In both figures, each red circle indicates a data point, of which the x and y coordinates are the results from VPR and our placement for the same testcase, respectively. The blue lines are the linear regression results of the data points. For both wirelength and frequency, the results from our placement and VPR are nearly perfectly correlated, with the correlation coefficients being 1.0000 for wirelength and 0.9994 for frequency. The near perfect correlation allows the proposed method to serve for fast early prototyping.

4.4.5 Placement Solution Demonstration

A couple of placement solutions are demonstrated in Figure 4.13. Each figure shows the middle region within a 64×64 systolic array layout. Four structurally adjacent PEs are highlighted with colors and all blocks of the same PE have the same color. Blocks of the other PEs are represented by gray squares. The regularity from our placement is evident in Figure 4.13(b) while VPR cannot retain the regularity as shown in Figure 4.13(a). On the other hand, the regularity results in vacant places, which can be seen in Figure 4.13(b). Therefore, our placement is not as compact as VPR and this explains the wirelength increase from our placement.

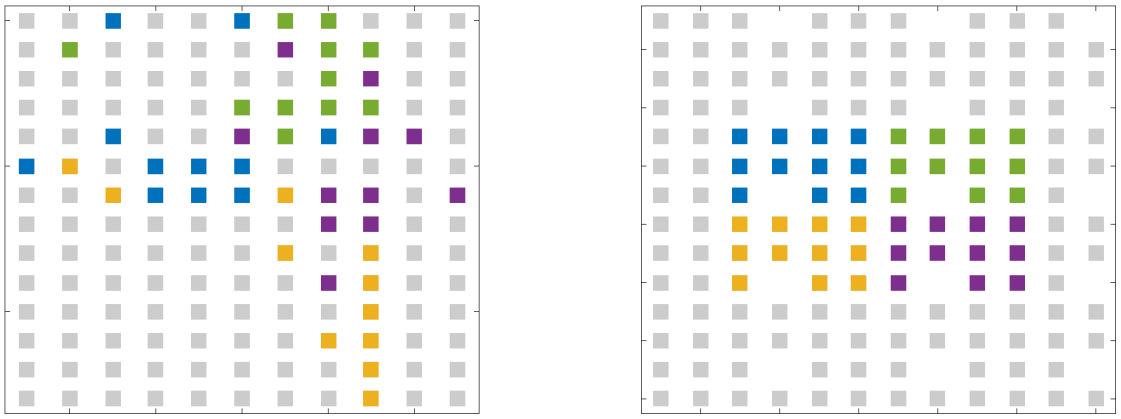

(a) VPR (b) Ours

Figure 4.13: Placement of 4 PEs in a 64×64 array.

4.4.6 Analysis of Individual Techniques

Besides maintaining regularity and thus reducing decision variables in simulated annealing, four other techniques are developed and included in our proposed method.

- **ILP**: PE array placement by ILP (Section 4.3.2) as a part of the initial solution for simulated annealing.
- **Speculation**: Speculative decisions for PE block moves described in Section 4.3.3.1.
- **Omission**: The wirelength calculation omission proposed in Section 4.3.3.2.
- **Hybrid**: Using hybrid HPWL and VPR wirelength model as described in Section 4.3.3.3.

For ILP, one particularly appealing feature is that its complexity is decided by PE size and independent of the number of PEs. There are hundreds of variables in the ILP formulation for the testcases and most of them are associated with the transformations in handling the absolute values.

The number of constraints is over 1000. The measured runtime versus PE array size is plotted in Figure 4.14, which confirms its independence of the number of PEs.

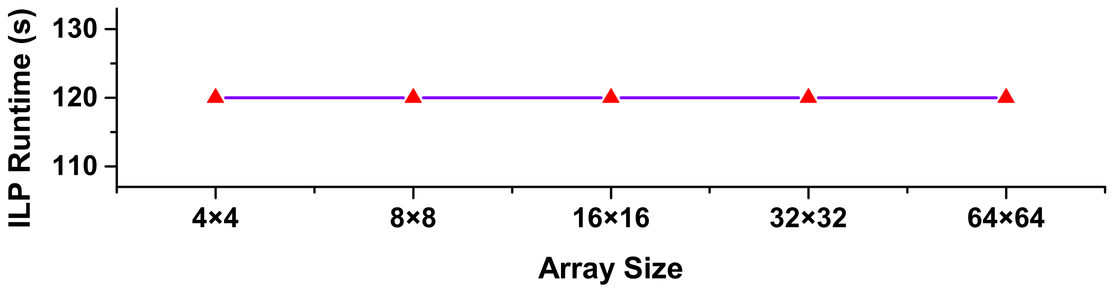


Figure 4.14: ILP runtime versus PE array size.

The effect of each individual techniques are further analyzed through experiments with results shown in Figures 4.15, 4.16 and 4.17. The effects are demonstrated through comparisons with the following.

- **Baseline1**: None of the 4 techniques is used and wirelength is completely evaluated by HPWL model in simulated annealing.
- **Baseline2**: None of the 4 techniques is used and wirelength is completely evaluated by VPR model in simulated annealing.
- **All**: All of the 4 techniques are applied.

In Figures 4.15, 4.16 and 4.17, the results for one technique means only this technique is applied on top of Baseline1.

The runtime speedup comparisons are shown in Figure 4.15. One can see that ILP and Omission contribute to most of the speedup among the 4 techniques and the speedup improvement from Speculation is also close. Hybrid is intended to tradeoff between speedup and solution quality and therefore its speedup is lower than Baseline1 while it is still higher than Baseline2.

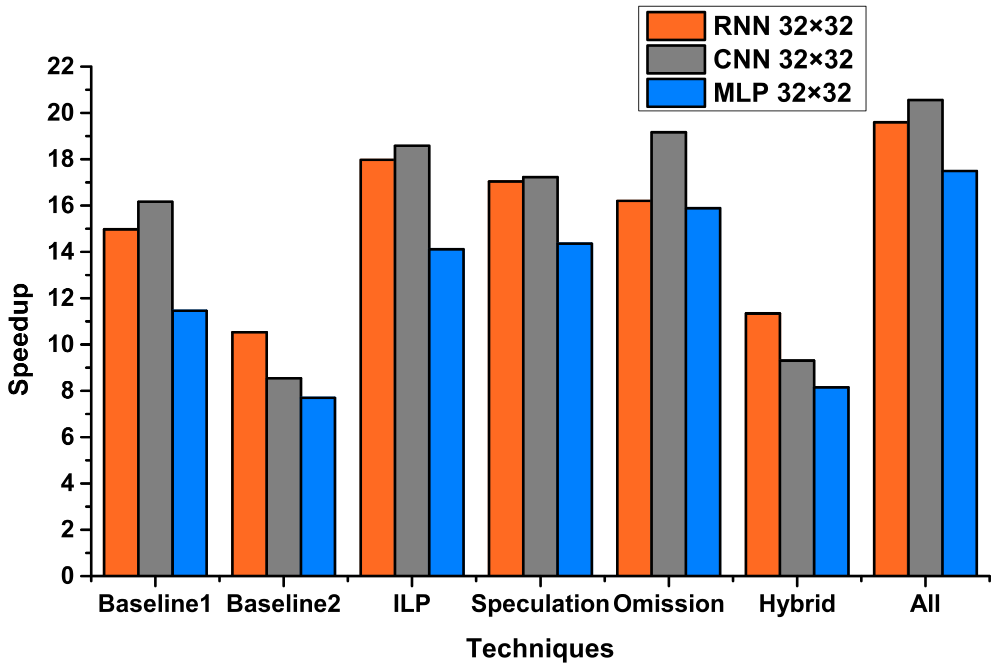


Figure 4.15: The effect of each technique on runtime speedup.

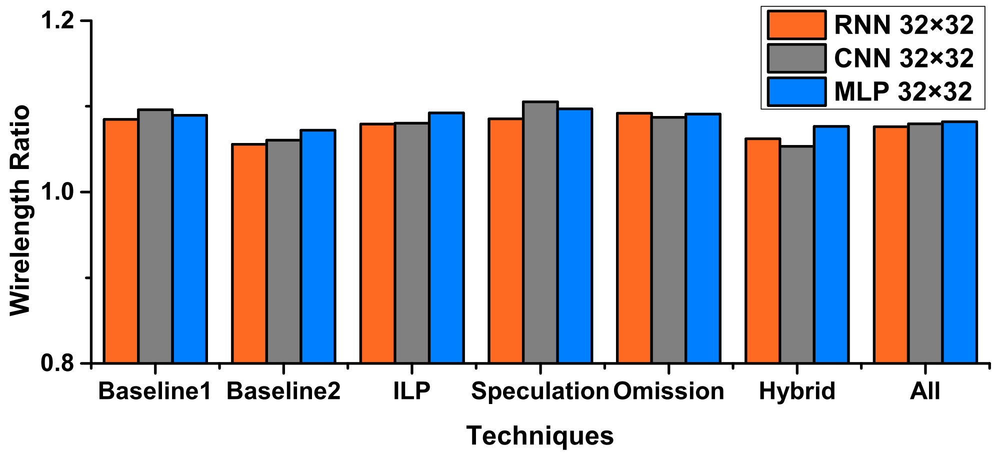


Figure 4.16: The effect of each technique on wirelength.

The effects on wirelength are compared in Figure 4.16 where the vertical axis corresponds to normalized wirelength with respect to VPR solutions. ILP and Hybrid can reduce wirelength overhead a little compared to Baseline1. The Omission wirelength is supposed to be the same as that of Baseline1 in theory. The slight difference is due to the randomness in simulated annealing. Overall, the impact to wirelength from these techniques vary little. This observation implies that the wirelength increase from our method is mostly due to the regularity constraints instead of algorithm deficiency.

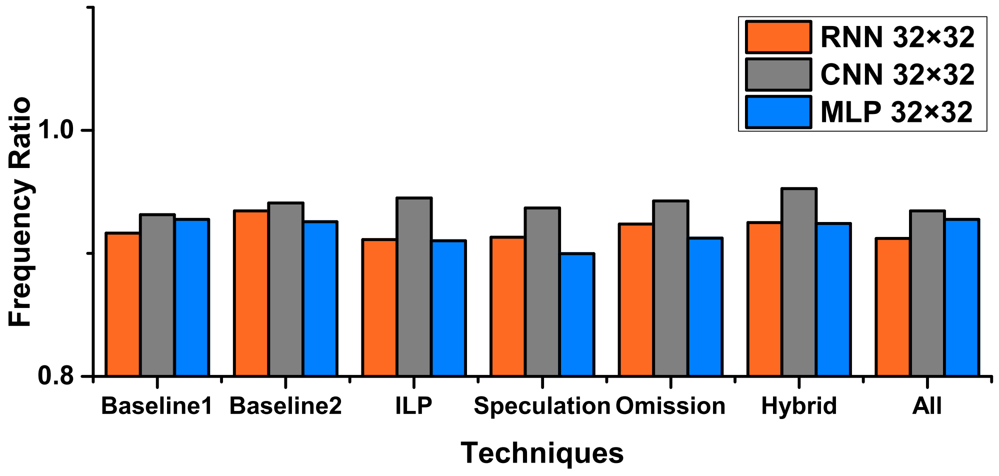


Figure 4.17: The effect of each technique on frequency.

The frequency results are depicted in Figure 4.17, where the vertical axis indicates normalized frequency with respect to VPR results. Among these techniques, Hybrid has the minimum frequency degradation while the frequencies from the other techniques are close to each other. This trend also indicates that the frequency degradation is more from regularity constraints than the algorithms.

In the Hybrid technique, HPWL model is used in the first $1 - \rho$ portion of simulated annealing iterations, where $\rho \in [0, 1]$ is a parameter, and VPR wirelength model is employed in the rest ρ portion. The rate ρ of using VPR model affects the tradeoff between runtime and solution quality.

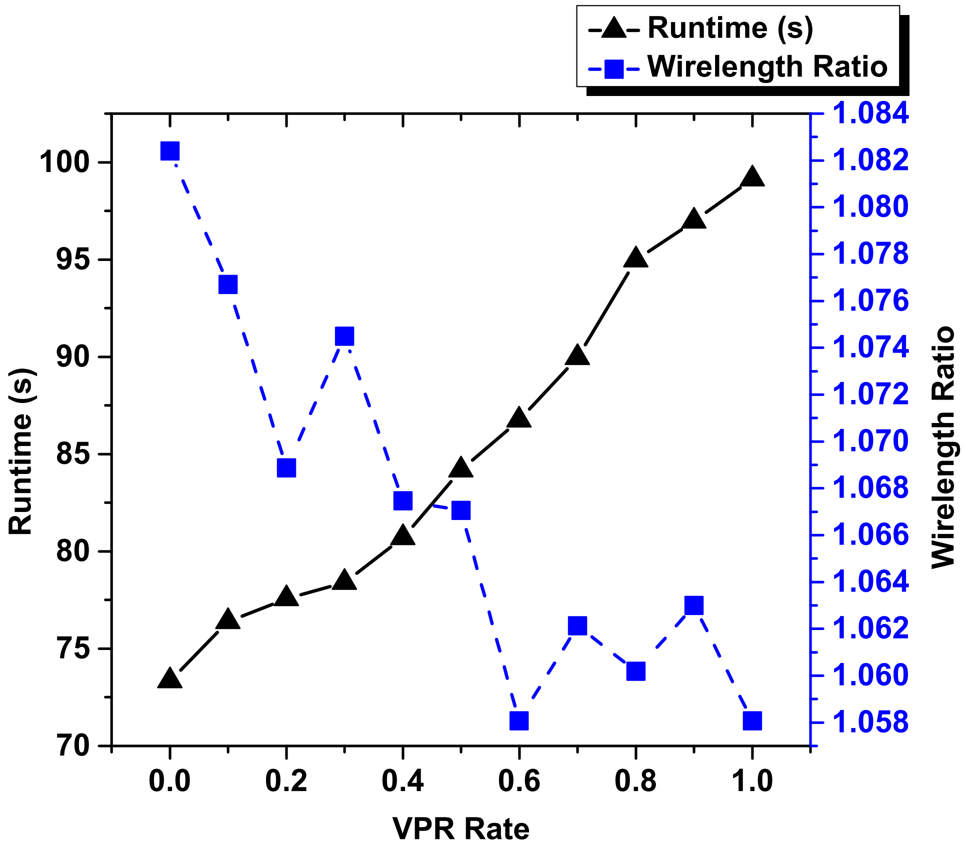


Figure 4.18: Runtime and wirelength tradeoff by different VPR wirelength model use rate on an 8×8 systolic array.

Experiments are performed on an 8×8 systolic array for studying this tradeoff and the results are plotted in Figure 4.18. As the use of VPR model increases, the simulated annealing runtime increases monotonically. At the same time, wirelength decreases in general with some kinks due to the heuristic nature of the algorithm as well as model inaccuracy.

4.5 Conclusions

There is a strong need for fast FPGA placement and increasing popularity for designs with regularity. This work is an effort to leverage the latter for helping the former. At the same time, the pros and cons of this approach are evaluated. A hybrid ILP and simulated annealing-based FPGA placement method is developed for design regularity. It achieves more than $25\times$ speedup versus a classical academic tool on relatively large neural network designs on FPGA. Meanwhile, it pays a price of near 10% wirelength and circuit frequency degradation. On the other hand, it has almost perfect correlation with the classical method. The proposed method will be useful for designs with tight turn-around time and early design prototyping.

5. SUMMARY AND CONCLUSIONS*

This dissertation research is focused on information processing techniques in different fields, including protocol study for vehicle system, applications for smart irrigation and FPGA based design for neural networks.

For the protocol study on in-vechicle communication system, we proposed DUCER scheme to implement error correction on networks with CRC and redundant channels. It can correct at most 5 errors with a high reliability. DUCER makes use of existing structure in networks and realizes error correction by using a little more memory. It avoids message retransmission, which causes information delay uncertainty. Compared to existing error correction methods, such as Reed Solomon code, DUCER shows one order of magnitude speed up.

Different information processing techniques have been applied to a smart irrigation system to improve the data accuracy and the system safety. Several customized filters have been designed and applied for the positioning system on irrigation machine. The error is finally reduced to $0.5m$ on average and the maximum $0.9m$. Power monitoring system is embedded into the control part to ensure operating safety.

Hardware neural network accelerator design suffers from its long design time. We proposed a fast regularity-aware placement method to reduce the design time. A hybrid ILP and simulated annealing-based FPGA placement method is developed for achieving design regularity and acceleration. Compared with a classical academic tool, it achieves more than $25\times$ speedup with less than 10% wirelength increase and circuit frequency degradation. It provides a convenient way for early prototyping and parameter tuning.

In the conclusion, this dissertation successfully explores different fields related to information processing techniques, and proposes solutions for various problems. Experiment results prove the

*©2018 IEEE. Part of this chapter is Reprinted, with permission, from Hongxin Kong, Jun Cheng, Krishna Narayanan, Jiang Hu, "DUCER: a Fast and Lightweight Error Correction Scheme for In-Vehicle Network Communication", IEEE International Conference on Vehicular Electronics and Safety, 09/2018.

effectiveness of the solutions.

REFERENCES

[1] Intel Edison Hardware Guide. https://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison-module_HG_331189.pdf.

[2] U-blox GPS. https://www.u-blox.com/en/product/c94-m8p.

[3] AMC1100 Datasheet. https://www.ti.com/lit/ds/symlink/amc1100.pdf?ts=1602212268781&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FAMC1100.

[4] Purpose and Internal Functionality of FPGA Look-Up Tables. https://www.allaboutcircuits.com/technical-articles/purpose-and-internal-functionality-of-fpga-look-up-tables/.

[5] B. Ronak and S. A. Fahmy, "Mapping for Maximum Performance on FPGA DSP Blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 573–585, 2015.

[6] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-Vehicle Networks: A Review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 534–545, 2015.

[7] B. Liu, W. Bai, and G. Zhen, "A Prompt Retransmission Method for In-vehicle Network FlexRay," in *Chinese Control Conference*, pp. 7841–7846, 2017.

[8] H. Kopetz, "Fault Containment and Error Detection in the Time-triggered Architecture," in *Proceedings of The International Symposium on Autonomous Decentralized Systems*, pp. 139–146, 2003.

[9] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The Time-Triggered Ethernet (TTE) Design," in *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 22–33, 2005.

[10] G. Deng and L. Cahill, "An Adaptive Gaussian Filter for Noise Reduction and Edge Detection," in *IEEE Conference Record Nuclear Science Symposium and Medical Imaging Conference*, pp. 1615–1619, 1993.

[11] S. Eberhardt, T. Duong, and A. Thakoor, "Design of Parallel Hardware Neural Network Systems from Custom Analog VLSI 'Building Block' Chips," *Transactions of the NEURON*, vol. 3, p. 2, 1989.

[12] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *IEEE Transactions on Micro*, vol. 35, no. 3, pp. 10–22, 2015.

[13] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design Space Exploration of FPGA-Based Deep Convolutional Neural Networks," *ACM/IEEE Asia and South Pacific Design Automation Conference*, pp. 575–580, 2016.

[14] H. Zheng, S. T. Gurumani, K. Rupnow, and D. Chen, "Fast and Effective Placement and Routing Directed High-Level Synthesis for FPGAs," *ACM International Symposium on Field-Programmable Gate Arrays*, p. 1–10, 2014.

[15] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *International Workshop on Field-Programmable Logic and Applications*, pp. 213–222, 1997.

[16] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 2, pp. 1–30, 2014.

[17] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, "Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning," *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 129–132, 2018.

[18] Y. Sankar and J. Rose, "Trading Quality for Compile Time: Ultra-fast Placement for FPGAs," *ACM International Symposium on FPGA*, pp. 157–166, 1999.

[19] M. G. Wrighton and A. M. DeHon, "Hardware-Assisted Simulated Annealing with Application for Fast FPGA Placement," *ACM International Symposium on Field Programmable Gate Arrays*, pp. 33–42, 2003.

[20] Y. Xu and M. A. Khalid, "QPF: Efficient Quadratic Placement for FPGAs," *International Conference on Field Programmable Logic and Applications*, pp. 555–558, 2005.

[21] A. Choong, R. Beidas, and J. Zhu, "Parallelizing Simulated Annealing-Based Placement Using GPGPU," *International Conference on Field Programmable Logic and Applications*, pp. 31–34, 2010.

[22] M. Gort and J. H. Anderson, "Analytical Placement for Heterogeneous FPGAs," *International Conference on Field Programmable Logic and Applications*, pp. 143–150, 2012.

[23] T.-H. Lin, P. Banerjee, and Y.-W. Chang, "An Efficient and Effective Analytical Placer for FPGAs," *ACM/IEEE Design Automation Conference*, pp. 1–6, 2013.

[24] M. An, J. G. Steffan, and V. Betz, "Speeding up FPGA Placement: Parallel Algorithms and Methods," *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 178–185, 2014.

[25] R. Pattison, Z. Abuowaimer, S. Areibi, G. Gréwal, and A. Vannelli, "GPlace: A Congestion-Aware Placement Tool for UltraScale FPGAs," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–7, 2016.

[26] G. Chen, C. Pui, W. Chow, K. Lam, J. Kuang, E. F. Y. Young, and B. Yu, "RippleFPGA: Routability-Driven Simultaneous Packing and Placement for Modern FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 2022–2035, 2018.

[27] W. Li and D. Z. Pan, "A New Paradigm for FPGA Placement Without Explicit Packing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2113–2126, 2019.

[28] J. Yuan, J. Chen, L. Wang, X. Zhou, Y. Xia, and J. Hu, "ARBSA: Adaptive Range-Based Simulated Annealing for FPGA Placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2330–2342, 2019.

[29] A. Koch, "Structured Design Implementation - A Strategy for Implementing Regular Datapaths on FPGAs," *ACM International Symposium on Field-Programmable Gate Arrays*, pp. 151–157, 1996.

[30] C. Lavin and A. Kaviani, "RapidWright: Enabling Custom Crafted Implementations for FPGAs," *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 133–140, 2018.

[31] N. Zhang, X. Chen, and N. Kapre, "RapidLayout: Fast Hard Block Placement of FPGA-Optimized Systolic Arrays Using Evolutionary Algorithms," *IEEE International Conference on Field Programmable Logic and Applications*, 2020.

[32] P. S. Kumar and Z. David, *Neural Networks and Systolic Array Design*, vol. 49. World Scientific, 2002.

[33] C. R. Castro-Pareja, J. M. Jagadeesh, S. Venugopal, and R. Shekhar, "FPGA-Based 3D Median Filtering Using Word-Parallel Systolic Arrays," *IEEE International Symposium on Circuits and Systems*, vol. 3, pp. III–157, 2004.

[34] Y. Yang, W. Zhao, and Y. Inoue, "High-Performance Systolic Arrays for Band Matrix Multiplication," *IEEE International Symposium on Circuits and Systems*, pp. 1130–1133, 2005.

[35] P. K. Meher, "Efficient Systolic Implementation of DFT Using a Low-Complexity Convolution-Like Formulation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 8, pp. 702–706, 2006.

[36] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Transactions on Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[37] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary,

Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," *ACM/IEEE International Symposium on Computer Architecture*, pp. 1–12, 2017.

[38] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs," *ACM/IEEE Design Automation Conference*, pp. 1–6, 2017.

[39] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, "Frequency Improvement of Systolic Array-Based CNNs on FPGAs," *IEEE International Symposium on Circuits and Systems*, pp. 1–4, 2019.

[40] Intel Quartus. `https://www.intel.com/content/www/us/en/programmable/documentation/zpr1513988353912.html`.

[41] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[42] W. Li, M. Di Natale, W. Zheng, P. Giusto, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Optimizations of An Application-level Protocol for Enhanced Dependability in FlexRay," in *Conference on Design, Automation and Test in Europe*, pp. 1076–1081, 2009.

[43] J. Dvořák and Z. Hanzálek, "Using Two Independent Channels With Gateway for FlexRay Static Segment Scheduling," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1887–1895, 2016.

[44] M. Rahmani, B. Muller-Rathgeber, and E. Steinbach, "Error Detection Capabilities of Automotive Network Technologies and Ethernet-A Comparative Study," in *Intelligent Vehicles Symposium*, pp. 674–679, 2007.

[45] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai, "IEEE 802.11 Wireless Local Area Networks," *IEEE Communications Magazine*, vol. 35, no. 9, pp. 116–126, 1997.

[46] P. Koopman, "32-bit Cyclic Redundancy Codes for Internet Applications," in *Dependable Systems and Networks International Conference*, pp. 459–468, 2002.

[47] F. Schiller and T. Mattes, "An Efficient Method to Evaluate CRC-polynomials for Safety-critical Industrial Communication," *Transactions of Applied Computer Science*, vol. 14, no. 1, pp. 57–78, 2006.

[48] FlexRay Protocol. `https://www.fujitsu.com/downloads/CN/fmc/lsi/FlexRay-EN.pdf`.

[49] P. Dayal and R. K. Patial, “Implementation of Reed-Solomon CODEC for IEEE 802.16 Network Using VHDL Code,” in *Optimization, Reliabilty, and Information Technology International Conference*, pp. 452–455, 2014.

[50] A. Al Azad, M. Huq, and I. R. Rokon, “Efficient Hardware Implementation of Reed Solomon Encoder and Decoder in FPGA Using Verilog,” in *International Conference on Advancements in Electronics and Power Engineering*, pp. 117–121, 2011.

[51] E. I. A. El Habti and R. El Gouri, “FPGA Implementation of A New Chien Search Block for Reed-Solomon Codes RS (255, 239) Used in Digital Video Broadcasting DVB-T,” *International Transactions of Engineering Research and Applications*, vol. 4, no. 8, pp. 82–86, 2014.

[52] A. Raghupathy and K. Liu, “Algorithm-based Low-power/High-speed Reed-Solomon Decoder Design,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 11, pp. 1254–1270, 2000.

[53] R. Faludi, *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*. O’Reilly Media Inc., 2010.

[54] E. Kaplan and C. Hegarty, *Understanding GPS: Principles and Applications*. Artech house, 2005.

[55] V. Di Lecce, A. Amato, and V. Piuri, “Neural Technologies for Increasing the GPS Position Accuracy,” *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, pp. 4–8, 2008.

[56] Y. Kim, R. G. Evans, and W. M. Iversen, “Remote Sensing and Control of an Irrigation System Using a Distributed Wireless Sensor Network,” *IEEE transactions on Instrumentation and Measurement*, vol. 57, no. 7, pp. 1379–1387, 2008.

[57] J. W. Jones, G. Hoogenboom, C. H. Porter, K. J. Boote, W. D. Batchelor, L. Hunt, P. W. Wilkens, U. Singh, A. J. Gijsman, and J. T. Ritchie, “The DSSAT Cropping System Model,” *European Transactions of Agronomy*, vol. 18, no. 3-4, pp. 235–265, 2003.

[58] A. C. McCarthy, N. H. Hancock, and S. R. Raine, “Simulation of Irrigation Control Strategies for Cotton Using Model Predictive Control Within the VARIwise Simulation Framework,” *Transactions of Computers and Electronics in Agriculture*, vol. 101, pp. 135–147, 2014.

[59] C. Lozoya, C. Mendoza, L. Mejía, J. Quintana, G. Mendoza, M. Bustillos, O. Arras, and L. Solís, “Model Predictive Control for Closed-loop Irrigation,” *Transactions of IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 4429–4434, 2014.

[60] L. Sun, Y. Yang, J. Hu, D. Porter, T. Marek, and C. Hillyer, “Reinforcement Learning Control for Water-efficient Agricultural Irrigation,” in *IEEE International Conference on Ubiquitous Computing and Communications*, pp. 1334–1341, 2017.

[61] Y. Yang, J. Hu, D. Porter, T. Marek, K. Heflin, H. Kong, and L. Sun, "Deep Reinforcement Learning-Based Planning and Automated Scheduling for Multi-zone Fixed Irrigation," *ASABE Annual International Meeting*, 2020.

[62] J. Zhu, H. Zhao, P.-G. Sun, and Y.-G. Bi, "Equilateral Triangle Localization Algorithm Based on Average RSSI," *Transactions of Northeastern University Natural Science*, vol. 28, no. 8, p. 1094, 2007.

[63] J. Xu, W. Liu, F. Lang, Y. Zhang, and C. Wang, "Distance measurement model based on RSSI in WSN," *Transactions of Wireless Sensor Network*, vol. 2, no. 8, p. 606, 2010.

[64] T. Instruments, "AMC1100: Replacement of Input Main Sensing Transformer in Inverters with Isolated Amplifier," *Application Report*, 2012.

[65] ROE DC/DC Converter. `https://recom-power.com/pdf/Econoline/ROE.pdf`.

[66] J. Serrano, *Introduction to FPGA Design.* Cern, 2008.

[67] R. C. Seals and G. Whapshott, *Programmable Logic: PLDs and FPGAs.* Macmillan International Higher Education, 1997.

[68] A. Kumar, A. Hansson, J. Huisken, and H. Corporaal, "An FPGA Design Flow for Reconfigurable Network-based Multi-processor Systems on Chip," in *Design, Automation & Test in Europe Conference & Exhibition*, pp. 1–6, 2007.

[69] W. Wang, Q. Meng, and Z. Zhang, "A Survey of FPGA Placement Algorithm Research," in *IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pp. 498–502, 2017.

[70] M. Xu, G. Gréwal, S. Areibi, C. Obimbo, and D. Banerji, "Near-Linear Wirelength Estimation for FPGA Placement," *Canadian Transactions of Electrical and Computer Engineering*, vol. 34, no. 3, pp. 125–132, 2009.

[71] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin II - An Open-Source Verilog HDL Synthesis Tool for CAD Research," *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 149–156, 2010.

[72] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "From Quartus to VPR: Converting HDL to BLIF with the Titan Flow," *International Conference on Field Programmable Logic and Applications*, pp. 1–4, 2013.

[73] Gurobi Optimization. `http://www.gurobi.com`.

[74] Xilinx Versal Architecture and Product Data Sheet. `https://www.xilinx.com/support/documentation/data_sheets/ds950-versal-overview.pdf`.