

Implementing a Seven-Person Voting Display Circuit Using Field-Programmable Logic Devices

Jianwu Zhu^{1,*}, Ying Zhang²

¹*School of Computer Science, Wuhan City Vocational College, Wuhan, Hubei, China*

²*Dongfanghong Primary School, Wuhan, Hubei, China*

**Corresponding Author*

Abstract: In scenarios such as conference voting and competition judging, simple and efficient voting devices are indispensable. Traditional discrete-circuit voting systems suffer from high hardware redundancy and poor flexibility. To optimize the design approach of voting circuits, this paper conducts a comparative study on seven-person voting circuit solutions. Two design methods are employed: one based on traditional digital discrete circuits to build hardware, and the other using an FPGA platform combined with LPM modules for programmable logic design. The results show that the discrete-circuit solution features intuitive structure and low cost but lacks scalability; in contrast, the FPGA-based solution offers high integration, convenient debugging, and reprogrammability. By comparing the strengths and weaknesses of both approaches, this study identifies suitable design strategies for different application scenarios, providing valuable references for lightweight and intelligent design of small-scale digital logic voting systems, thus demonstrating practical application value.

Keywords: FPGA; Data Selector; Adder Counter; Seven-Segment Display; Verilog; Integrated Circuit

1. Design Requirements

A voting system refers to a process in which multiple individuals cast votes on a resolution or action. The number of voters must be odd, and if more than half of the votes are in favor, the resolution is considered passed or the action deemed valid; otherwise, if more than half vote against it, the resolution is rejected or the action invalidated. A seven-person voting system, literally speaking, involves seven individuals casting separate votes. If four or more votes are in favor, the resolution is considered passed;

conversely, if four or more votes are against it, the resolution is considered rejected.

2. Digital Circuit Design

2.1 Digital Design

Digital design, also known as digital circuit design or digital system design, is an important field in electronics engineering and computer science. It primarily involves using logic gates (such as AND, OR, NOT, etc.) to design and implement complex digital systems.

2.2 Discrete-Circuit Implementation

The design based on digital circuits requires an 8-to-1 data selector 74LS151, a binary up counter 74LS161, a two-input NAND gate 74LS00, and a seven-segment display. Seven switches J1, J2, J3, J4, J5, J6, and J7 represent the voting status of seven individuals, where switch KEY being closed indicates agreement and open indicates disagreement. The voting result is indicated by LED X1: when the number of votes in favor is four or more, the outcome is considered as approval and LED X1 lights up; if fewer than four votes are in favor, the outcome is considered as disapproval and LED X1 remains off. Meanwhile, the seven-segment display shows the number of votes in favor. The functional table of the circuit diagram refers to Table 1, and the simulation circuit diagram refers to Figure 1

Table 1. Functional Table of the Seven-Person Voting Circuit

J1	J2	J3	J4	J5	J6	J7	S2	S1	S0	X1	Y (Votes)
J1	J2	J3	J4	J5	J6	J7	0	0	0	0	0
J1	J2	J3	J4	J5	J6	J7	0	0	1	0	1
J1	J2	J3	J4	J5	J6	J7	0	1	0	0	2
J1	J2	J3	J4	J5	J6	J7	0	1	1	0	3
J1	J2	J3	J4	J5	J6	J7	1	0	0	1	4
J1	J2	J3	J4	J5	J6	J7	1	0	1	1	5
J1	J2	J3	J4	J5	J6	J7	1	1	0	1	6
J1	J2	J3	J4	J5	J6	J7	1	1	1	1	7

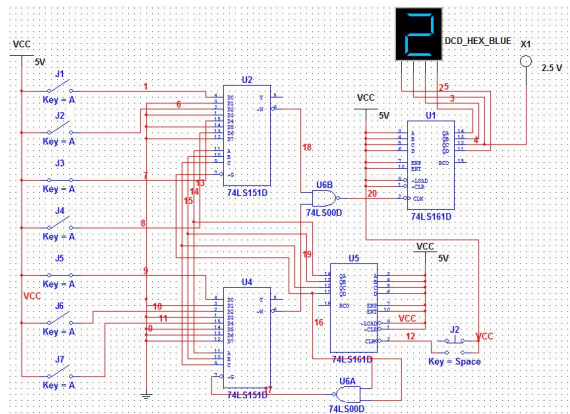


Figure 1. Voting Circuit

3. Design of Large-Scale Programmable Logic Devices

FPGA (Field Programmable Gate Array), also known as field programmable gate array, is a programmable digital integrated circuit (IC) that integrates millions of logic gates internally. Functionally, it can replace traditional combinations of IC chips to implement digital systems. FPGA evolved from a series of programmable devices such as PAL and CPLD. As a semi-custom circuit in the application-specific integrated circuit (ASIC) domain, FPGA not only addresses the shortcomings of custom circuits but also overcomes the limitation of early programmable logic devices, which had a limited number of logic gates.

More importantly, the field-programmable nature of FPGAs allows designers to configure logic blocks and interconnections on-site, enabling them to implement or modify the functionality of complex electronic systems. All FPGA architectures consist of three key components: configurable logic blocks, configurable I/O modules, and programmable interconnect resources.

3.1 Main FPGA Development Process

(1) First, create a project and establish a new project file. Development tools use project files to manage different projects [1].
 (2) Second, establish the top-level module. The top-level module acts as a large container that, using a top-down design approach, incorporates all sub-modules of the entire project and integrates them into a single top-level diagram for simulation and compilation. Similarly, the top-level model can be viewed as a circuit diagram containing various smaller sub-circuits, where simulation and compilation involve emulating the operation of this complete circuit [2].

(3) This training development utilizes the parameterized macro function IP library built into Quartus software, which is an industry-standard EDIF parameterized module. It allows direct configuration of bit width, ports, and operating modes without manually writing low-level gate circuits. The compiler automatically optimizes resources for Altera FPGA hardware, improving timing performance and resource utilization. The Quartus software environment includes a wide range of commonly used functional modules, such as counters, accumulators, encoders, comparators, and decoders [3].

(4) If the required functional module is not available in the component library, you can create and design it yourself, then store it in the library for future use. Functional modules can be generated either by using hardware description language (Verilog) or by drawing schematics. The process of creating a module is similar to generating an engineering file: first, create a new project, generate the module's graphical symbol (Symbol), then invoke this symbol in the top-level diagram. Next, copy the Verilog source code file into the project directory containing the top-level model. Alternatively, you can also copy the completed schematic file into the same project directory. Either method allows you to easily create a custom component [4].

(5) Next, from bottom to top, connect the various functional submodules of the top-level container using wiring. Similar to the schematic design process, first arrange the components in the design window, then use the connection tool to link each subcomponent together. Construct the simulation design circuit [5].

(6) After completing the schematic design, an FPGA chip compatible with the simulation software must be selected to enable the circuit system to run on hardware. The main steps are: (1) selecting an appropriate FPGA chip model; (2) assigning pins by connecting the inputs, outputs, and individual signals of the top-level model to the corresponding hardware pins on the chip; (3) configuring compilation parameters, which vary depending on the type of circuit [6].

(7) The next step is compilation. The compilation process resembles simulation in EDA software, but differs from traditional EDA tool simulations by also involving hardware programming of the FPGA chip. The entire compilation process - including circuit simulation, pin assignment, and automatic routing - is automatically handled by the Quartus system [7].

(8) After a successful compilation, Quartus generates files with the extensions .sof or .pof. The .sof file can be downloaded to the FPGA chip via the JTAG debug interface on the FPGA development board. Using the various logic gates built into the FPGA, all functions of the designed circuit can be perfectly implemented. However, once power is cut off, the circuit configuration stored in the FPGA chip will be lost. In contrast, the .pof file can be downloaded to the AS programming interface (EEPROM chip) of the FPGA. Even after power loss, the circuit configuration information remains intact, and the full functionality of the circuit can be restored upon re-powering [8].

(9) In addition to the FPGA simulation tool Quartus, there is Xilinx's Vivado Design Suite, which includes a highly integrated design environment and a new generation of tools spanning from system to IC level, enabling functional development for Xilinx FPGA devices. Model Technology's ModelSim is an HDL simulation software that can be used to simulate VHDL, Verilog HDL, or mixed-language code designs, and also supports simulation of various common IEEE hardware description languages [9].

3.2 Design Steps

- 1). Identify the required hardware components: First, determine the hardware components needed to implement the seven-person voting system, including FPGA chips, LED displays, buttons, and serial communication modules.
- 2). Design the power circuit: To supply power to all hardware components, a stable power circuit must be designed to ensure the system operates properly.
- 3). Design the FPGA control circuit: The FPGA chip is the core of the entire system, and a control circuit must be designed using an appropriate FPGA chip according to the design requirements.
- 4). Design the button input circuit: The seven-person voting system requires seven buttons, each representing a voter's vote. A button input circuit must be designed to connect these buttons to the FPGA chip.
- 5). Design the serial communication circuit: To enable data transmission between the FPGA chip and the host computer, a serial communication circuit must be designed, with an appropriate communication protocol and baud rate selected.
- 6). Design the LED display circuit: To display voting results in real time, an LED display circuit

must be designed, an appropriate LED display selected, and a corresponding display program written according to design requirements.

7). Consider system stability and reliability: To ensure the system's stability and reliability, thorough testing and verification of the circuit are required, along with appropriate measures such as adding decoupling capacitors and filters to enhance the system's anti-interference capability [10].

4. Verilog Implementation

Two Verilog programs are designed to realize the seven-person voting logic. Program One explicitly sums the individual votes and drives both the seven-segment display and the result LED; Program Two uses a loop to count the votes and determine the pass/fail outcome.

Program 1. Seven-Person Voting Program One

```
module exp10(K1, K2, K3, K4, K5, K6, K7,
ledag, mResult);
  input K1, K2, K3, K4, K5, K6, K7; // 7 people
  output [3:0] ledag; // seven-segment
display
  output mResult; // voting result
  reg [3:0] ledag; reg mResult;
  reg [2:0] KNum, K1Num, K2Num, K3Num,
K4Num, K5Num, K6Num, K7Num;
  always @(K1 or K2 or K3 or K4 or K5 or K6
or K7) begin
    K1Num = {1'b0, 1'b0, K1}; // count votes in
favor
    K2Num = {1'b0, 1'b0, K2}; K3Num = {1'b0,
1'b0, K3};
    K4Num = {1'b0, 1'b0, K4}; K5Num = {1'b0,
1'b0, K5};
    K6Num = {1'b0, 1'b0, K6}; K7Num = {1'b0,
1'b0, K7};
  end
  always @(K1Num or K2Num or K3Num or
K4Num or K5Num or K6Num or K7Num)
    KNum <= K1Num + K2Num + K3Num +
K4Num + K5Num + K6Num + K7Num;
  always @(KNum) // LED on when result low,
pass if >= 4
    if (KNum > 3) mResult <= 0; else mResult <=
1;
  always @(KNum) // show number of votes on
seven-segment display
    case (KNum)
      3'b000: ledag = 4'b0000; 3'b001: ledag =
4'b0001;
      3'b010: ledag = 4'b0010; 3'b011: ledag =
```

```

4'b0011;
  3'b100: ledag = 4'b0100; 3'b101: ledag =
4'b0101;
  3'b110: ledag = 4'b0110; 3'b111: ledag =
4'b0111;
  default: ledag = 4'b0000;
endcase
endmodule

```

Program 2. Seven-Person Voting Program Two

```

module voter7(pass, vote);
  output pass;
  input [6:0] vote;
  reg [2:0] sum; integer i; reg pass;
  always @(vote) begin
    sum = 0;
    for (i = 0; i <= 6; i = i + 1)
      if (vote[i]) sum = sum + 1; // sum + 1 when
vote is true
    if (sum[2]) pass = 1; // high bit set => >= 4
    else pass = 0;
  end
endmodule

```

5. Summary

Traditional digital design and FPGA-based logic design adopt completely different data processing mechanisms, which fundamentally determines their differences in operational performance and application flexibility. Conventional digital circuit design primarily relies on a serial processing architecture. In such systems, data signals are transmitted sequentially through a series of discrete logic gate circuits from the input port to the output port. Each logic unit processes data one by one, and the subsequent module can only start working after the previous calculation is completed. This serial working mode limits the data processing speed and real-time response capability of the entire circuit system. In contrast, FPGA devices are built on a parallel hardware architecture that supports simultaneous multi-channel data stream processing. Combined with the parallel execution characteristics of Verilog hardware description language, multiple functional modules inside the FPGA can operate independently and synchronously. This unique working mechanism enables FPGAs to realize high-speed, high-efficiency and real-time data processing, which is far superior to traditional serial digital circuits in complex logic scenarios. In terms of functional flexibility and reconfigurability, the two design methods also show obvious differences. Traditional discrete

digital circuits are fixed hardware systems. Once the circuit wiring and component layout are solidified on the hardware board, the logical functions of the system cannot be changed arbitrarily. If users need to adjust functions or modify logical algorithms, they have to redesign and rebuild the hardware circuit, which brings great limitations to system iteration. However, FPGA has prominent field programmability. Its internal logical functions can be repeatedly configured and updated by rewriting Verilog programs without changing any physical hardware structures. This feature makes FPGA systems extremely flexible, which can adapt to diverse algorithm requirements and multi-task switching scenarios.

From the perspective of cost, maintainability and scalability, FPGA design also has comprehensive advantages. Traditional custom digital circuits require independent hardware design, component selection and circuit board production according to specific functional requirements, resulting in high development costs and long development cycles. Moreover, the solidified hardware circuit is difficult to modify and expand. Any function upgrade needs physical operations such as rewiring and component replacement, leading to poor maintainability and scalability. As a general-purpose programmable hardware platform, FPGA can be reused for multiple different design projects through program reprogramming, which effectively reduces development costs. Meanwhile, function modification, upgrade and expansion can be completed only by software programming, greatly improving system maintainability and scalability. In the practical design scenario of the seven-person voting circuit, FPGA-based design shows more comprehensive and prominent advantages than traditional digital design.

References

- [1] Rose J, Brown S. Flexibility of Interconnection Structures for Field-Programmable Gate Arrays. *IEEE Journal of Solid-State Circuits*, 1991, 26(3): 277-282.
- [2] Chow P, Seo S O, Au D, et al. A 1.2 um CMOS FPGA Using Cascaded Logic Blocks and Segmented Routing. *Custom Integrated Circuits Conference (CICC)*, 1991: 137-140.
- [3] Betz V, Rose J. VPR: A New Packing, Placement and Routing Tool for FPGA Research. *Field-Programmable Logic and Applications (FPL)*, 1997: 213-222.

- [4] Chen D, Cong J, Pan P. FPGA Design Automation: A Survey. *Foundations and Trends in Electronic Design Automation*, 2006, 1(3): 195-330.
- [5] Ludwin A, Betz V, Padalia K. High-Quality, Deterministic Parallel Placement for FPGAs on Commodity Hardware. *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2008: 14-23.
- [6] Manohararajah V, Chiu G R, Singh D P, et al. Difficulty of Predicting Interconnect Delay in a Timing Driven FPGA CAD Flow. *Proceedings of the System-Level Interconnect Prediction Workshop (SLIP)*, 2006: 3-8.
- [7] Rose J, El Gamal A, Sangiovanni-Vincentelli A. Architecture of Field-Programmable Gate Arrays. *Proceedings of the IEEE*, 1990, 81(7): 1013-1029.
- [8] Brown S, Rose J. FPGA and CPLD Architectures: A Tutorial. *IEEE Design & Test of Computers*, 1996, 13(2): 42-57.
- [9] Wang C, Zhang J. Based on FPGA Design and Simulation of Function Signal Generator. *Applied Mechanics and Materials*, 2013, 380-384: 3292-3295.
- [10] Coskun O, Avci K. FPGA Schematic Implementations and Comparison of FIR Digital Filter Structures. *Balkan Journal of Electrical and Computer Engineering*, 2018, 6(1): 20-28.