Chapter 19
Design and Realization of Mobile Terminal Side Time Synchronization Based on FPGA

Qi Liu, Xiangchao Meng, and Xiaosong Cao

Abstract In order to solve the current demand of high precision time synchronization modularization of mobile terminals, the problem that mobile network standard cannot provide high precision time synchronization ability is solved. Based on the capability of the existing mobile communication system, this project will design a time synchronization module of the mobile terminal side based on field programmable gate array (FPGA) by studying the airport timing and the terminal side's airport timing process.

19.1 Introduction

With the rapid development of network technology and integrated circuits, mobile terminals have also broken free from their constraints and experienced significant growth [1]. In the face of massive interactions and collaborative cooperation among mobile devices, time synchronization has become particularly important, and its accuracy has become a goal pursued by humans. Whether its email communication, telephone billing, campus card authentication, or mobile gaming, time synchronization is essential. Without accurate time synchronization or if the precision is too low, it would lower our quality of life [2]. Nowadays, there are various time synchronization methods such as global navigation satellite system (GNSS), network time protocol (NTP), and precision time protocol (PTP), each shining in their respective domains, but with certain limitations [3]. For example, GNSS is constrained by costs, and NTP lacks high precision. Currently, absolute time synchronization requirements can be

Q. Liu (✉) · X. Cao
Beijing Polytechnic, Beijing, China
e-mail: liuqi@bpi.edu.cn

X. Cao
e-mail: Caoxiaosong@bpi.edu.cn

X. Meng
Beijing Rongzhi Huixin Engineering Co., Ltd, Beijing, China

R. Kountchev et al. (eds.), *New Approaches for Multidimensional Signal Processing*,
Smart Innovation, Systems and Technologies 385,
https://doi.org/10.1007/978-981-97-0109-4_19

achieved in scenarios like wired symmetric link networks. However, in scenarios with long-distance wireless link networks, due to the instability and asymmetry of the wireless links, most cannot achieve high-precision wireless time synchronization [4], which would negatively impact people's daily lives due to poor user experience.

This paper investigates the timing process on the air interface of mobile terminals and based on the interaction between field programmable gate array (FPGA) and the terminal designs an FPGA-based timing synchronization module for mobile terminals built upon existing mobile network devices and air interface links. The air interface timing not only ensures good measurement of uplink and downlink latency and channel delay in the existing air interface protocol stack (physical layer: PHY, media access control: MAC, etc.) but also achieves higher timing accuracy compared to technologies like NTP and PTP. This simple design enables high-precision time synchronization while ensuring good user experience and accommodating a large number of users. It also reduces costs by utilizing existing or planned network devices such as mobile edge computing (MEC) [5], base stations, and terminals. With the development of 5G, high-speed, low-latency, and high-capacity communication is experiencing vigorous growth. This time synchronization technology can be applied in scenarios such as the Internet of Things and connected vehicles, ushering in a new era with the advent of 5G and allowing everyone to participate in the wireless network world.

19.2 Related Theories

19.2.1 End-to-End Delay Mechanism

The basic principle of PTP synchronization is that the master and slave clocks exchange synchronization messages and record the transmission and reception time of these messages. The round-trip delay between the master and slave clocks is calculated by measuring the time difference between transmitting and receiving the messages. If the network is symmetrical, meaning that the transmission delays in both directions are the same, then the one-way delay is half of the round-trip delay. The clock offset between the master and slave clocks is equal to this one-way delay. By adjusting the local time based on the calculated offset, the slave clock can achieve synchronization with the master clock. The following Fig. 19.1 illustrates the implementation process of the end-to-end delay mechanism in the two-step mode [6].

In the two-step mode of the end-to-end delay mechanism, the timestamp t_1 of the Sync message is carried by the *Follow_up* message, and t_4 and t_5 are carried by the *Pdelay_Req* and *Pdelay_Resp_Follow_UP* messages, respectively.

The implementation process of the end-to-end delay mechanism in the two-step mode is as follows:

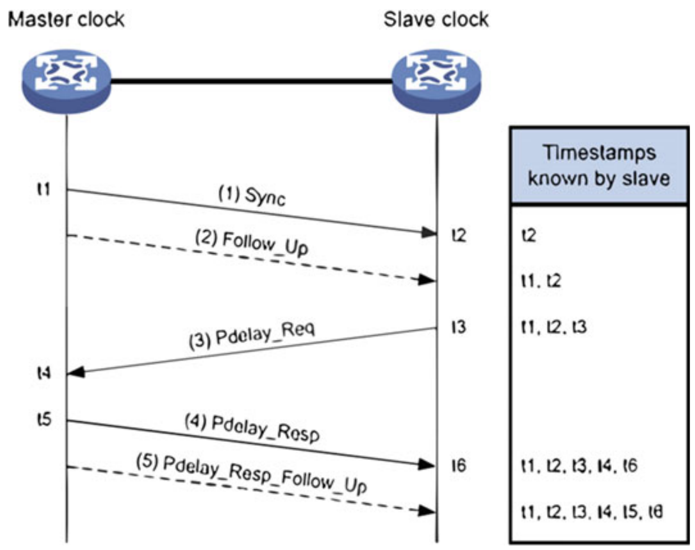


Fig. 19.1 Implementation process of the end-to-end delay mechanism in the two-step mode

- The master clock sends a Sync message to the slave clock and records the transmission time t_1. The slave clock, upon receiving the message, records the reception time t_2.
- After sending the Sync message, the master clock immediately sends the *Follow_up* message containing t_1.
- The slave clock sends a *Pdelay_Req* message to the master clock to initiate the calculation of the reverse transmission delay. The slave clock records the transmission time t_3, and upon receiving the message, the master clock records the reception time t_4.
- Upon receiving the *Pdelay_Req* message, the master clock replies by sending a *Pdelay_Resp* message with t_5 as the transmission time. The slave clock records the reception time t_6 upon receiving this message.
- After replying to the *Pdelay_Req* message, the master clock sends a *Pdelay_Resp_Follow_UP* message containing t_5.

At this point, the slave clock has obtained six timestamps: t_1 to t_6. Using these timestamps, the round-trip delay between the master and slave clocks can be calculated as $[(t_4 - t_3) + (t_6 - t_5)]$, and assuming a symmetrical network, the one-way delay between the master and slave clocks is $[(t_4 - t_3) + (t_6 - t_5)]/2$. Therefore, the clock offset between the slave and master clocks is Offset $= (t_2 - t_1) - [(t_4 - t_3) + (t_6 - t_5)]/2$.

19.2.2 Air Interface Timing Overview

At the network level, air interface timing has already had mobile network air links, and the physical layer and data link layer of the air interface protocol stack can ensure relatively good measurement of delay variations and channel delays for uplink and downlink. In future applications, due to the large number of users in air interface timing, there will be significant demand and user volume in areas such as connected vehicles, IoT, and artificial intelligence. Moreover, network equipment such as base stations, terminals, MEC, etc., can utilize existing or upcoming equipment from operators to save costs. Air interface timing can achieve higher timing accuracy using air interface wireless signals, albeit compared to technologies like NTP and PTP which have relatively lower accuracy. Figure 19.2 shows the timing network structure:

In this timing network structure, atomic clock hardware and timing services interact and influence each other. The core network and base stations in the figure achieve time synchronization monitoring and reduce dependence on satellites, thereby achieving higher precision timing using the network. In the future, it can be applied to industries such as autonomous driving in 5G networks and timing for IoT devices.

For the network timing system, the following design plan is proposed, as shown in Fig. 19.3.

Master Clock → Core Network → Base Station/MEC: Using optical network and PTPv2 technology.

Base Station → UE/Gateway: Air interface physical layer for signal timing.

Clock Hierarchy:

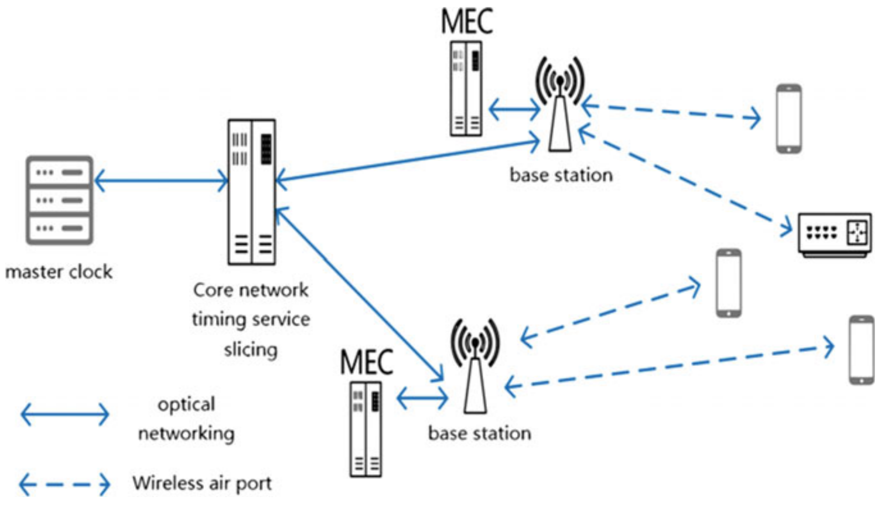


Fig. 19.2 Timing network structure

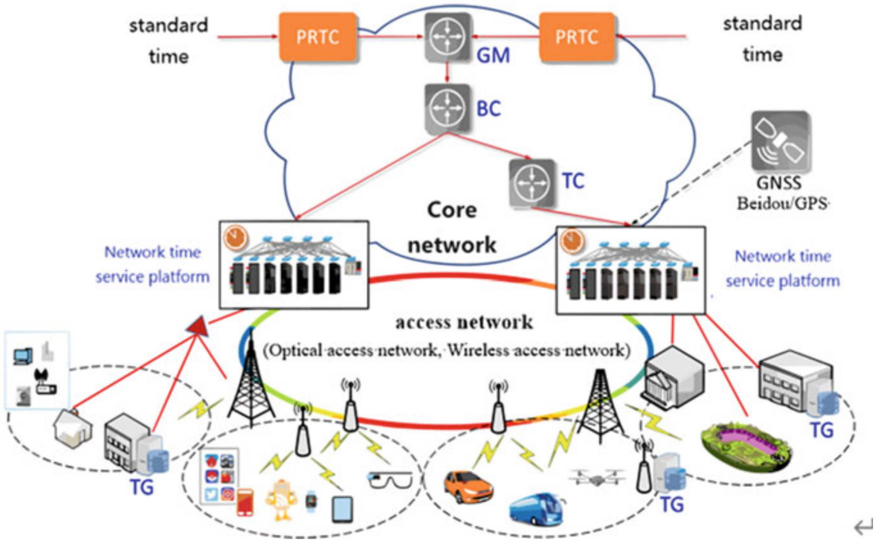

standard time
PRTC
GM
PRTC
standard time
BC
TC
GNSS
Beidou/GPS
Core network
Network time service platform
Network time service platform
access network
(Optical access network, Wireless access network)
TG
TG
TG

Fig. 19.3 Timing system overview

- GM (Grandmaster Clock): Has the highest stability, precision, and reliability.
- BC (Boundary Clock): Can establish various branch points in different layers of the structure.
- TC (Transparent Clock): Can measure and compensate for the time consumed by PTP event information in the device.

19.2.3 Air Interface Timing Principles and Process

Basic principle: Air interface timing is based on periodic, low-latency physical layer synchronization signals. The base station records the time of the signal transmission and sends that time through a physical broadcast channel or a physical downlink shared channel (PDSCH) channel. The terminal blindly detects the signal and records the received time. Afterwards, the base station provides feedback on the uplink delay measurement through a RandomAccessResponse. The terminal predicts and modifies the timing error.

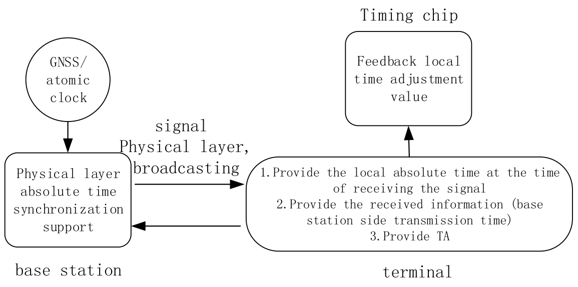

Timing chip
Feedback local time adjustment value
GNSS/ atomic clock
signal
Physical layer, broadcasting
Physical layer absolute time synchronization support
1. Provide the local absolute time at the time of receiving the signal
2. Provide the received information (base station side transmission time)
3. Provide TA
base station
terminal

Fig. 19.4 Flowchart of time synchronization process

19.3 System Design

19.3.1 Design of Time Synchronization System

Overall scheme of Time Synchronization System based on the studied air interface timing process on the mobile terminal side, further design of the time synchronization system is carried out. Figure 19.4 shows the flowchart of the time synchronization process.

The FPGA in the terminal needs to possess the following three capabilities:

- The FPGA needs to retrieve the terminal's local time when detecting radio resource control (RRC) signals, as well as the sending time of the broadcasted signals from the base station and the value of *n*_TA signal in random access response (RAR) signals or directly obtain the TimeAdvance value.
- Based on the above data, calculate the adjustment value for the terminal's local time.
- Provide feedback of the adjustment value to the terminal.

On the other hand, the terminal needs to open interfaces on the current basis to provide the aforementioned sending time, local time, and TimeAdvance value. This enables the terminal to correct its local time based on the adjustment value provided by the chip. Furthermore, the Verilog [7] language is used to implement the required functionalities of the FPGA.

19.3.2 FPGA-Based Time Synchronization Module for Mobile Terminals

Regarding the FPGA-based time synchronization module on the mobile terminal side, a rough overall design including the terminal's timing system is developed based on existing resources.

The total TimeAdvance value is recorded to represent the accumulated results from the timing chip based on all the baseband signals. This result is obtained through the TimeAdvanceCommand calculation according to the 3GPP standard process, denoted as:

$$\mathrm{TA}_{\mathrm{total}} = \mathrm{TA}_{\mathrm{RA}} + \sum_i \mathrm{TA}_i \tag{19.1}$$

Equation (19.1) serves as an estimation of uplink delay. $\mathrm{TA}_{\mathrm{total}}$ is indicated the total TimeAdvance value. $\mathrm{TA}_{\mathrm{RA}}$ is indicated the resource allocation of TimeAdvance. TA_i is indicated the Time Advance.

t_0 represents the delay of hardware data transmission and processing, such as the baseband module and timing chip. Since it is relatively fixed and may vary across different hardware, it needs to be determined in advance. If the current channel delay condition is unstable and the Nb timing signals sent by the base station cannot meet the timing conditions set by the chip, the module sends a termination command to the timing chip. The timing chip judges and calculates based on the input system frame number (SFN) and system information block (SIB_t) signals. If the SFN does not match the rSFN in the signal, the current signal is discarded, and the next timing signal is received and processed. For valid timing signals, t_{offset} is calculated based on the formula shown in the figure. The timing chip outputs absolute time information and PPS second pulse based on t_{offset} and ends the current timing process, waiting for the start of the next timing process. The clock adjustment amount for the timing terminal is t_{offset}. Figure 19.5 shows a rough flowchart of the FPGA-based timing process on the mobile terminal side.

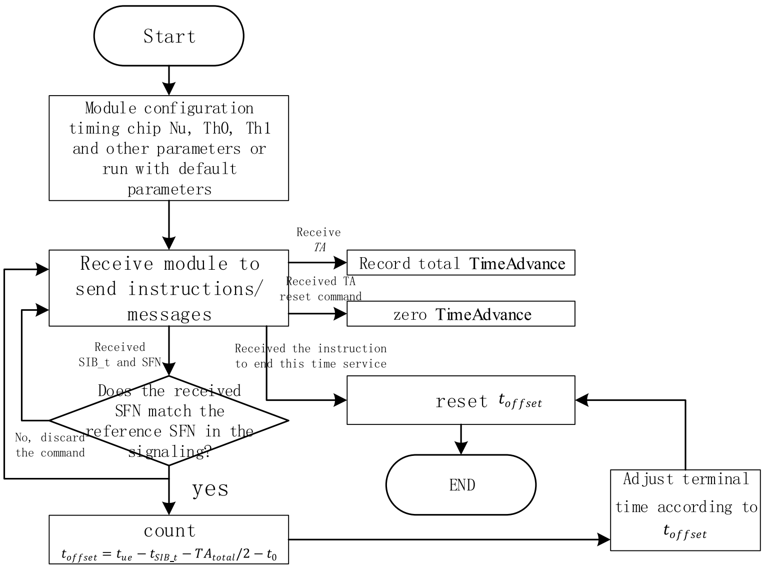

Start
Module configuration timing chip Nu, Th0, Th1 and other parameters or run with default parameters
Receive module to send instructions/ messages
Receive TA
Record total TimeAdvance
Received TA reset command
zero TimeAdvance
Received SIB_t and SFN
Received the instruction to end this time service
Does the received SFN match the reference SFN in the signaling?
No, discard the command
reset t_{offset}
END
yes
count
$t_{offset} = t_{ue} - t_{SIB\_t} - TA_{total}/2 - t_0$
Adjust terminal time according to t_{offset}

Fig. 19.5 Rough flowchart of FPGA-based timing process on the mobile terminal side

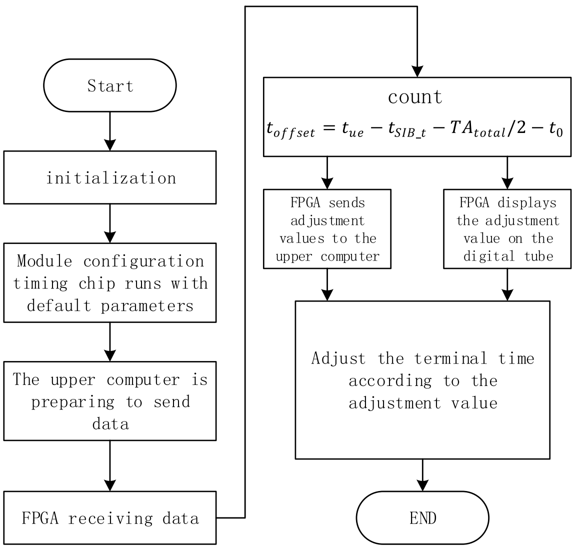

Start
initialization
Module configuration timing chip runs with default parameters
The upper computer is preparing to send data
FPGA receiving data
count
$t_{offset} = t_{ue} - t_{SIB\_t} - TA_{total}/2 - t_0$
FPGA sends adjustment values to the upper computer
FPGA displays the adjustment value on the digital tube
Adjust the terminal time according to the adjustment value
END

Fig. 19.6 FPGA terminal timing process

It involves measuring the hardware transmission and processing delays of the baseband module and timing chip, estimating the transmission delay TA caused by distance, recording the local absolute time of the receiving end, and responding with the sending time of the transmitting end. Due to network symmetry, the required adjustment for the terminal's time is calculated. The specific design between the FPGA and the terminal is shown in Fig. 19.6: FPGA Terminal Timing Process.

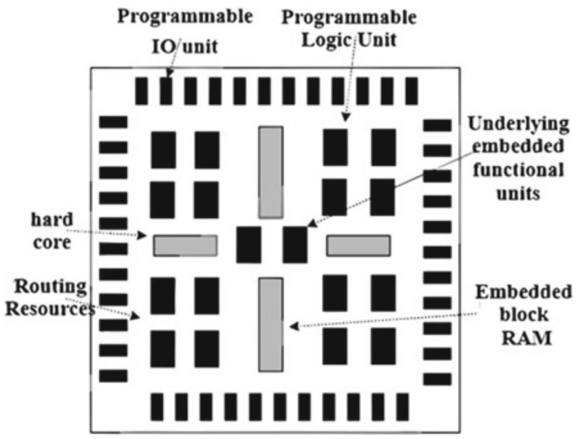
$$t_{\text{offset}} = t_{ue} - t_{\text{SIB}\_t} - \text{TA}_{\text{total}}/2 - t_0 \tag{19.2}$$

Through the interaction between the FPGA and the terminal, the FPGA obtains TimeAdvance value, value, local reception time, and signal transmission time from the interface. After calculating the adjustment value using Eq. 19.2, the result is displayed on the digital display and returned to the terminal.

19.4 Hardware and Software Resources

The FPGA [8], short for Field Programmable Gate Array, is a type of semiconductor digital integrated circuit that is also a field-programmable gate array, allowing many of its internal circuit functions to be modified based on specific requirements. In the past, PLD (Programmable Logic Device) and ASIC (Application-Specific Integrated Circuit) could only perform their respective roles without fully utilizing their advantages. However, with the emergence of FPGA, it combines the strengths of PLD and ASIC. It can have thousands or even millions of logic gates while still

Fig. 19.7 FPGA structure



being programmable on the field, enabling larger and more complex functionalities. Additionally, FPGA design costs are relatively lower compared to ASIC, and making changes to previous designs in FPGA is relatively easier. FPGA's main market used to be in the communication and networking fields, but with the passage of time and technological advancements, FPGA applications have explosively grown in consumer electronics, automotive, electrical, and other sectors.

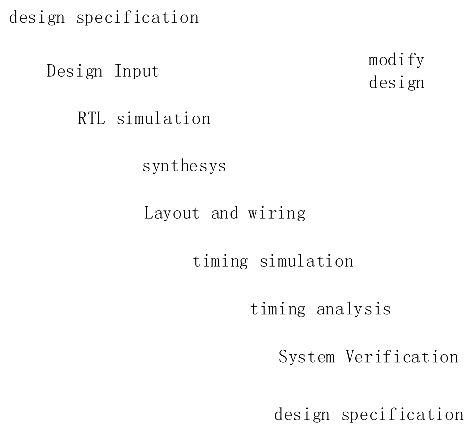
The simplified structure of an FPGA consists of six main components: the FPGA chip itself, programmable Input/Output Modules (IOB) for interfacing with the external world, programmable Logic Modules (CLB), embedded Block RAM (BRAM), interconnect resources for connecting all parts of the FPGA internally, underlying logic elements, and dedicated hard cores. The figure below shows the structure of an FPGA in Fig. 19.7.

In general, the design flow of an FPGA includes several steps, such as defining design inputs according to design specifications, RTL simulation, synthesis of the design, performing placement and routing, conducting timing simulation, analyzing timing, and performing board-level verification. The FPGA design process is shown in Fig. 19.8.

19.5 Interaction Between FPGA and Terminal

In FPGA applications such as traffic lights and digital displays, the system clock cannot be used directly, and sometimes multiple clock pulses of different frequencies are required as driving sources in a digital system. One of FPGA's features is its ability to process clock signals. Therefore, the concept of frequency division comes into being, and the frequency divider has a fundamental and important role in FPGA design. Generally, there is an integrated Phase-Locked Loop (PLL) in FPGA to achieve clock multiplication, frequency division, phase shift, and programmable

Fig. 19.8 FPGA design process



duty cycle functions. However, a PLL or IP core resource can also be excluded in designs that do not demand high clock accuracy.

Frequency dividers are mainly divided into even and odd divisions, and relatively speaking, even division principles are simple and easier to master. It can be achieved by counting with a counter. When using N times even frequency division, the clock triggers the counter for counting, and only needs to flip the output clock when counting to N/2 − 1, and so on. Figure 19.9 shows the simulation diagram of a four-fold frequency divider.

For odd frequency division, if you want to implement odd-frequency division with a 50% duty cycle, you cannot flip the clock at the midway point of the counter as with even division, as this method cannot obtain a 50% duty cycle clock. If you use a method similar to even frequency division to trigger the rising edge of a frequency-divided clock, you will get a clock signal with a duty cycle that is not 50%. If the

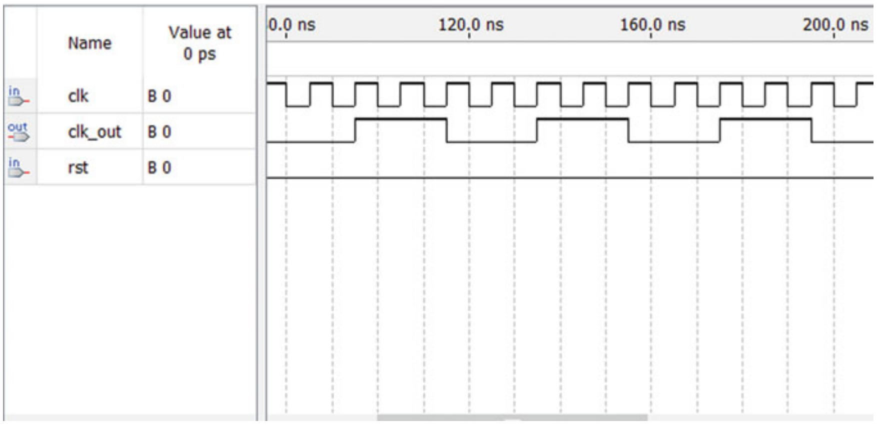


Fig. 19.9 Simulation diagram of four-fold frequency divider

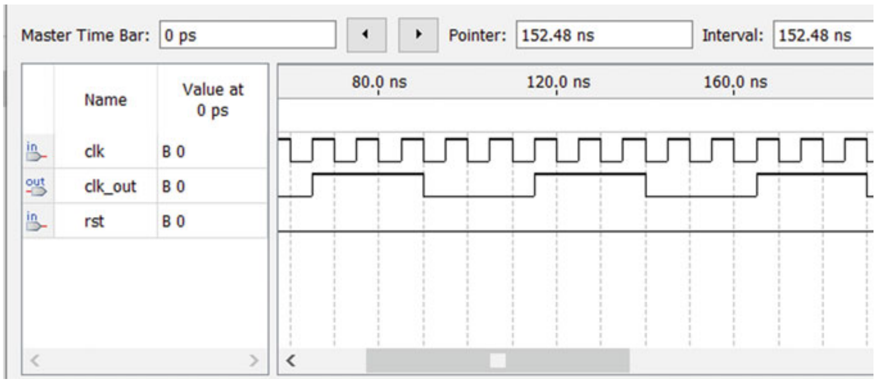


Fig. 19.10 Five-fold frequency divider

clock is triggered again on the falling edge, another clock signal with a duty cycle that is not 50% will be obtained. By performing logical operations on these two clock signals, a clock with a 50% duty cycle can be flexibly obtained. That is, to perform M-fold odd-frequency division on a clock with a duty cycle of 50%, first perform modulo M counting on the rising edge trigger, select a value for clock inversion output, and then perform another clock inversion output on (M − 1)/2, obtaining an odd-m frequency clock with a duty cycle that is not 50%. Next, at the same time, perform modulo M counting on the falling edge trigger, and when the clock inversion output value is the same as that selected for the rising edge trigger, start performing clock inversion output on the output clock, and then perform another clock inversion output on (M − 1)/2, obtaining an odd-m frequency clock with a duty cycle that is not 50%. Perform logical operations on two m frequency clocks (with a duty cycle not equal to 50%), taking the AND operation for positive periods and the OR operation for negative periods, to obtain an odd-m frequency clock with a duty cycle of 50%. Figure 19.10 shows the simulation diagram of a five-fold frequency divider.

19.6 Software Design for Time Synchronization

Schematic Diagram. In this design, the upper computer (Friendly Serial Debugging Assistant) sends four data to the FPGA in hexadecimal format. These data include the local reception time of the terminal for the signaling, the transmission time of the base station for sending the signaling [9], the TimeAdvance value, and the hardware data transmission and processing delay t_0 of the baseband module, timing chip, and other components. In this design, they are respectively named as A, B, C, and D.

After receiving these four data, the FPGA calculates the adjustment value based on the time synchronization principles and procedures stated in Sect. 19.3, using the formula $A - B - C/2 - D$ [10]. The calculated adjustment value will be displayed

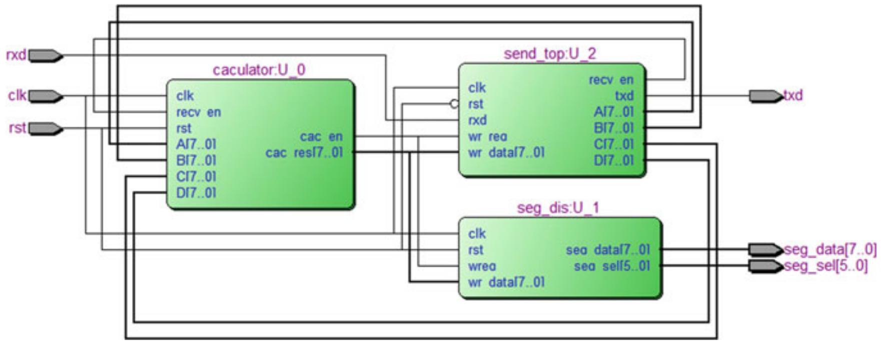


Fig. 19.11 RTL schematic diagram

on the digital display (in decimal format) and sent back to the upper computer (in hexadecimal format). Figure 19.11 shows the RTL schematic diagram of this design.

19.7 Simulation Testing and On-Board Debugging Results

Based on the principles and specific implementation of the above schematic and code, the upper computer in this design sends interface data to the FPGA and completes the calculation of the adjustment value. The adjustment value is then sent back to the upper computer and displayed on the digital display.

The upper computer sends 200, 12, 50, 100 to the FPGA, and after the adjustment value calculation, it is displayed on the digital display [11]. According to the previous formula, the adjustment value should be 63. The hexadecimal representation of 200, 12, 50, 100 is C8, C, 32, 64. The simulation result is shown in Fig. 19.12.

The interface of the Serial Debugging Assistant is shown in Fig. 19.13.

It can be seen that when the upper computer sends the four data in hexadecimal format as C8, 0C, 32, 64 to the FPGA, the FPGA calculates and returns the hexadecimal data 3F, which is equivalent to decimal 63, to the upper computer. Figure 19.14 shows the display on the digital display.

It can be seen that the displayed number on the digital display is 63.

19.8 Conclusions

The upper computer serves as a simulated terminal to provide and send four data to the FPGA, including the local reception time of the received signal at the terminal, the base station sending time of the signal, the delay of hardware data transmission and processing such as baseband module and timing chip, and the TimeAdvance value (estimated value of uplink delay). The FPGA processes and calculates the four data

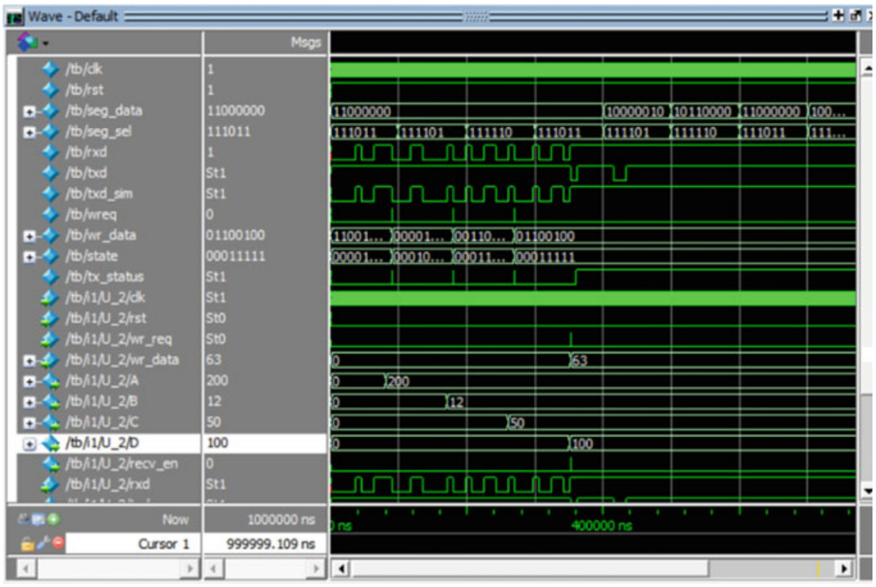
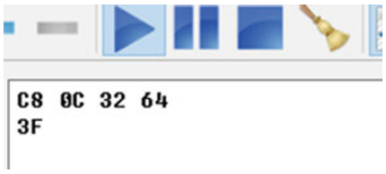


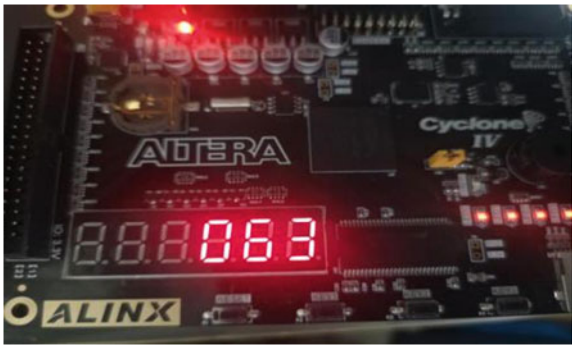
Fig. 19.12 Simulation result

Fig. 19.13 Debugging assistant interface



```
C8 0C 32 64
3F
```

Fig. 19.14 Digital display



based on the aforementioned time synchronization principles and sends the calculation result as an adjustment value to the upper computer, which is then displayed on the digital display. This completes the basic process of time synchronization.

Looking ahead, with the continuous advancement of mobile networks, it is expected that more efficient and accurate time synchronization methods will be developed to meet the increasing demands of information exchange and terminal interaction.

Acknowledgements The authors would like to thank the associate professor Zhaoming Lu who is working at Beijing University of Posts and Telecommunications (P.R. China) in particularly for the support in thesis examination and guidance. The authors are also grateful to the teachers of OAI WORKSHOP (http://www.opensource5g.org/) for their invaluable support in deploying the system and in providing experimental data and technical documentation. This work was supported by the R&D Program of Beijing Municipal Education Commission (No. KM202110858001).

References

1. LNCS Homepage, http://www.springer.com/lncs. Accessed 21 Nov 2016
2. Zhang, H., Xu, J., Liu, X., Long, K., Leung, V.C.M.: Joint optimization of caching placement and power allocation in virtualized satellite-terrestrial network. IEEE Trans. Wirel. Commun. (2023). https://doi.org/10.1109/TWC.2023.3257178
3. Chang, Y., Yang, H.X., Zhu, Q.H.: Optimize the deployment and integration for multicast-oriented virtual network function tree. J. Beijing Inst. Technol. **31**(05), 513–523 (2022). https://doi.org/10.15918/j.jbit1004-0579.2022.106
4. Chen, Y., Han, W., Zhu, Q., et al.: Target-driven obstacle avoidance algorithm based on DDPG for connected autonomous vehicles. EURASIP J. Adv. Signal Process. **2022**, 61 (2022)
5. Wu, D.P., Liu, Q.R., Wang, H.G. Wu, D.L., Wang, R.Y.: Socially aware energy-efficient mobile edge collaboration for video distribution. IEEE Trans. Multimed. **19**(10), 2197–2209 (2017)
6. Wu, D.P., Shi, H., Wang, H.G., Wang, R.Y., Fang, H.: A feature based learning system for internet of things applications. IEEE Internet Things J. **6**(2), 1928–1937 (2019)
7. Farooq, U., Baig, I., Alzahrani, B.A: An efficient inter-FPGA routing exploration environment for multi-FPGA systems. IEEE Access **6**, 56301–56310 (2018)
8. Keten, U.: GPS/GNSS independent Time Transfer over Telco IP Core Networks using DTM overlay. In: 2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS), NA, FL, USA, pp. 1–4 (2021). https://doi.org/10.1109/ISPCS49990.2021.9615348
9. Payal, R., Saxena, A., Chanda, B.: Implementation of smart home through FPGA using Verilog hardware descriptive language. In: 2020 IEEE International Conference on Advent Trends in Multidisciplinary Research and Innovation (ICATMRI), Buldhana, India, pp. 1–6 (2020). https://doi.org/10.1109/ICATMRI51801.2020.9398499
10. Liu, Y., Zhang, W., Zhou, J., Wang, Y., Wang, S., Zhang, L.: A design of encoding arbitration and interrupt request for dynamic reconfigurable high-speed serial bus in cyber physical system. In: 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Atlanta, GA, USA, pp. 627–634 (2019). https://doi.org/10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00122
11. Sun, C., He, Z., Niu, K.: An efficient underwater clock synchronization algorithm based on the principle of firefly synchronization. In: OCEANS 2021: San Diego, Porto, CA, USA. pp. 1–5 (2021). https://doi.org/10.23919/OCEANS44145.2021.9705903