Australian National University

ASSIGNMENT COVER SHEET

ANU College of Engineering and Computer Science
Australian National University
Canberra ACT 0200 Australia
www.anu.edu.au

+61 2 6125 5254

Submission and assessment is anonymous where appropriate and possible. Please do not write your name on this coversheet.

This coversheet must be attached to the front of your assessment when submitted in hard copy. If you have elected to submit in hard copy rather than Turnitin, you must provide copies of all references included in the assessment item.

All assessment items submitted in hard copy are due at 5pm unless otherwise specified in the course outline.

Student ID	Gong Chaoyun u6329142		
For group assignments, list each student's ID	Ju Yang u6338572 Xiaochen Liu u6312197		
Course Code	ENGN8537		
Course Name	Embedded Systems and Real Time Digital Signal Processing		
Assignment number	Major Project 6		
Assignment Topic	Real Time FPGA-based Ethernet Control Communication for a Servo Motor		
Lecturer	Dr. Aline I. Maalouf		
Tutor	Mr. Ehab Salahat, Mr. Siddharth Pethe, Dr. Nicolo Malagutti		
Tutorial (day and time)	Thursday 11.00 am – 1.00 pm		
Word count	11052	Due Date	28th October 2019
Date Submitted	28th October 2019	Extension Granted	

I declare that this work:

- ☐ upholds the principles of academic integrity, as defined in the ANU Policy: Code of Practice for Student Academic Integrity;
- ☐ is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the course outline and/or Wattle site;
- ☐ is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- ☐ gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- ☐ in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

Initials Chaoyun Gong, Yang Ju, Xiaochen Liu.

For group assignments, each student must initial.

Real Time FPGA-based Ethernet Control Communication for a Servo Motor

The Australian National University

ENGN 8537 Embedded Systems and Real Time Digital Signal Processing

Group 6

Group Members

Gong Chaoyun	u6329142
Ju Yang	u6338572
Xiaochen Liu	u6312197

Date: 28 October 2019

List of Contents

List of Figures *4*

List of Tables *6*

1.Introduction ***7***

1.2 Background **7**

1.3 Scope **9**

2.Methology ***10***

2.1 Ethernet technology **10**

2.1.1 Introduction to ethernet. 10

2.1.2 Ethernet protocols 10

2.1.3 Ethernet data unit 12

2.1.4 TCP/IP Protocol and Socket 14

2.2 Servo motor control **16**

2.2.1 Servo motor 16

2.2.2 PWM Control 20

2.3 Introduction to Qsys **21**

2.3.1 Software and hardware requirements [20] 22

2.3.2 Qsys advantages 23

2.3.3 Signal settings for HPS and FPGA interfaces [22] 24

2.3.4 Peripheral pin multiplexing 25

2.4 Introduction to AXI Bus **25**

2.4.1 AXI architecture 26

2.4.2 Signal description 28

2.4.3 Signal interface requirements 30

2.4.4 basic read and write transmission 30

2.4.5 Read and write response structure 31

2.4.6 Accessing HPS Devices from the FPGA 32

2.4.7 Connecting an FPGA Master to the HPS Interconnect 32

2.4.8 Enabling the FPGA-to-HPS Bridge 33

3 Implementation ***34***

3.1 Design Overview **34**

3.2 Communication function **35**

3.3 Motor control function **37**

3.4 Building connection between HPS and FPGA **39**

4.Results ***48***

5.Discussion ***51***

6.Conclusion ***52***

7. Future Work ***53***

7.1 More servo motors **53**

7.2 Control other devices **53**

8.Reference ***54***

9.Appedix 56

Server Program 56

Client Program 60

Hps_0 64

Modified GHRD 67

List of Figures

Fig. 1 The top view of DE10-nano board 8
Fig. 2 The bottom view of DE10-nano board 8
Fig. 3 The block diagram of DE10-nano board 9
Fig. 4 OSI model 11
Fig. 5 Data message format for different layers 13
Fig. 6 The relationship between OSI 7 layers model and TCP/IP network model 14
Fig. 7 Three ways handshake 16
Fig. 8 Socket connection 16
Fig. 9 Closed-loop system of servo motor 17
Fig. 10 Compare the industrial servo motors and smaller RC or hobby servos 17
Fig. 11 MG996R Servo Motor 18
Fig. 12 Three lines of MG996R Servo Motor 19
Fig. 13 PWM control of MG 996R 19
Fig. 14 The schematic graph of hobby servo motor 19
Fig. 15 PWM control of servo motor 20
Fig. 16 PWM control in this project 21
Fig. 17 Top level system structure 23
Fig. 18 Peripheral pin multiplexing 25
Fig. 19 Read architecture 26
Fig. 20 Write architecture 26
Fig. 21 Interface and interconnection 27
Fig. 22 VALID before READY handshake 30
Fig. 23 READY before VALID handshake 31
Fig. 24 VALID with READY handshake 31
Fig. 25 Qsys GUI 33
Fig. 26 Flow chart of the process 34
Fig. 27 Flow chart of Socket communication function 36
Fig. 28 Flow chart of PWM control using timer 38
Fig. 29 Platform Designer icon 39
Fig. 30 Qsys interface 39
Fig. 31 Set the clock source 40
Fig. 32 Set the parameters of HPS 40
Fig. 33 Set the parameters of AXI Bridge 41
Fig. 34 Set the peripheral pins 41
Fig. 35 Set the external clock sources 41
Fig. 36 Set the memory clock frequency 42
Fig. 37 Set memory timing 42
Fig. 38 Set the PIO component 43
Fig. 39 Set the SDRAM 43
Fig. 40 Assigning the addresses of components 43
Fig. 41 Addresses of system peripherals 44
Fig. 42 Building instantiation template 44
Fig. 43 Generating HDL 45
Fig. 44 Start compilation 46
Fig. 45 Information in header file 47
Fig. 46 The input command in client and the corresponding position of servo motor in FPGA board 48

Fig. 47 The output of current angle of servo motor in PuTTY 49
Fig. 48 Some continuous input commands in client 50

List of Tables

Table 1 The specifications of MG996R Servo Motor 18
Table 2 Global signal 28
Table 3 Write address channel signal 28
Table 4 Read address channel signal 29
Table 5 Base address of built-in devices.....32

1.Introduction

The aim of the project is to implement real time FPGA-based Ethernet control communication for a servo motor. It means that the position and speed of servo motor which is connected to FPGA board can be controlled precisely through Ethernet in real time.

To achieve the purpose of the project, those devices are included in this project: DE10-nano board, Servo Motor Kit (SMK), MG996R Servo Motor, window computer, Ethernet cable, USB cable and 40 pins wire jumper.

Several key points should be considered. Firstly, using TCP to transport the messages at the network level in TCP/IP model on the Ethernet platform. Secondly, the HPS part in DE10-nano board will be used to received messages from Ethernet and the Linux operating system will be run in HPS. Thirdly, the FPGA part in DE10-nano board will be connected to Servo Motor Kit and control the servo motor through the GPIO port. Fourthly, in the DE10-nano board, the HPS part and FPGA part should be connected together. Fifthly, the PWM will be used to control the position and speed of servo motor.

This project will focus on the communication between different devices in Ethernet (the control computer is window and HPS is Linux) and the connection between HPS and FPGA in DE10-nano board by using Qsys.

When the project has been implemented, more servo motors or other devices can be controlled through Ethernet. This project can be used in Internet of Things (IoT). In life, people can control the household appliances by using phones through Internet when they are not at home. It also can be applied in some industry parts.

1.2 Background

FPGA is the abbreviation of Field Programmable Gate Arrays. It is semiconductor device which is consisted of configurable logic blocks (CLBs) matrix [1]. And can be reprogrammed for different applications and requirements manually [1].

Comparing with other microprocessor, like CPU, GPU, ASIC, FPGA has some advantages. It is flexible, reusable and quick to acquire [2]. And FPGA has short design period to market, because you can use a ready-made FPGA and configure it [2]. Although FPGA has higher

upfront cost than microprocessor like ASIC, it can avoid wasting by reprograming it for different tasks [2]. In performance, FPGA is very good at in highly parallelized tasks [2]. For example, some massive image or digital signal processing [2]. And FPGA has a concise design cycle, so it can meet different requirements with short period [2]. Thus, FPGA has wide market, like Aerospace & Defense, ASIC Prototyping, Audio, Automotive, Data center, Industrial, Medical, Video & Image Processing and Wired & Wireless Communications.

In this project, DE10-nano board has been used. The hardware has been shown below [3]:

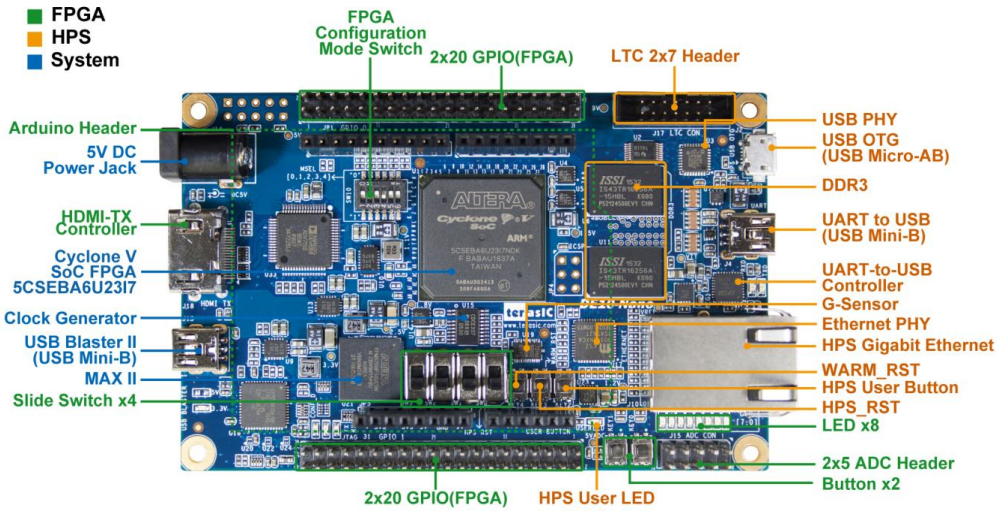


Fig. 1 The top view of DE10-nano board

In the Fig.1, the DE10-nano board have been divided into FPGA, HPS and System, three parts.

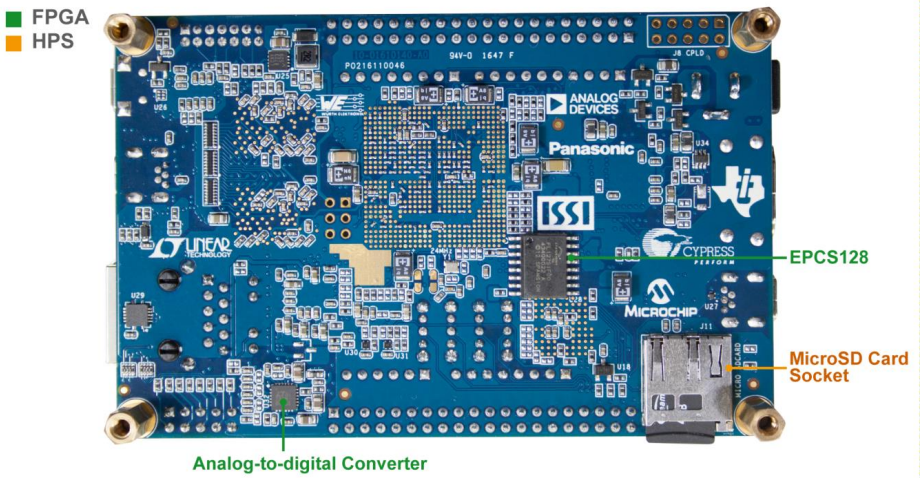


Fig. 2 The bottom view of DE10-nano board

In Fig.2 [3], it shows that the MicroSD card is belong to HPS part. The Linux operating system is in it. The project will upload the program to the MicroSD card.

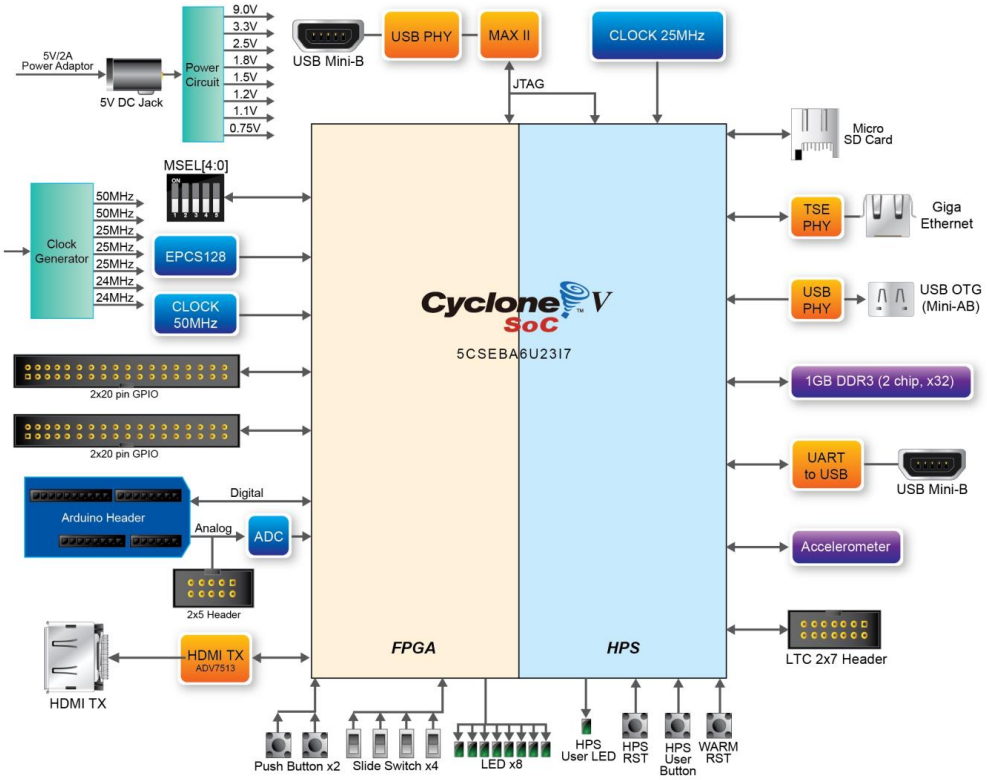


Fig. 3 The block diagram of DE10-nano board

Fig.3 [3] is the block diagram of the board. The FPGA board can be divided into FPGA part and HPS part. And both of them are through the Cyclone V SoC [3]. Giga Ethernet port is used to connect to the Ethernet. GPIO port is used to control the servo motor. And the figure shows clearly that the Giga Ethernet port is in HPS part and the GPIO port is in FPGA part. Thus, the project needs to connect the FPGA and HPS. The software for the FPGA design is Quartus II.

1.3 Scope

This report will introduce the theories of the technologies which are applied in this project. Then the Implementation section will describe the approach and methodology which is used to implement the project. After that, the Results section will show the result of the project and discus the performance and the advantages, disadvantages. After Conclusion, some parts which can be improved in future will also be covered in the Future Work section. Finally, the socket code and Qsys configuration will be attached in Appendix.

2.Methology

2.1 Ethernet technology

2.1.1 Introduction to ethernet.

Ethernet technique was purposed by Xerox Palo Alto research center in 1973. Currently ethernet standard has become the most commonly used telecommunication protocol standard in local area network (LAN), which is known as IEEE 802.3 standard [4]. The ethernet standard is consisted of standard ethernet (10 Mbps), fast ethernet (100 Mbps), Gigabit ethernet (1000 Mbps) and 10Gbps Ethernet, etc.

Gigabit ethernet is the most broadly applied high speed ethernet technique. It inherited the technique standards that the previous ethernet protocols identified, which includes CSMA/CD protocol, ethernet frames, full duplex communication and flow control, etc [4]. Besides, Gigabit ethernet also shows some new characters, for example, 8b/10b encoding is applied in case of meeting the requirement of optical fiber transmission, as well as frame bursting is applied in order to increase the efficiency. Thanks to the development of ethernet technique, its market occupation increases steadily, and as a result, the application of ethernet extended from LAN to Metropolitan area network and wide area network.

2.1.2 Ethernet protocols

The OSI seven layers model [5] is shown in the figure below:

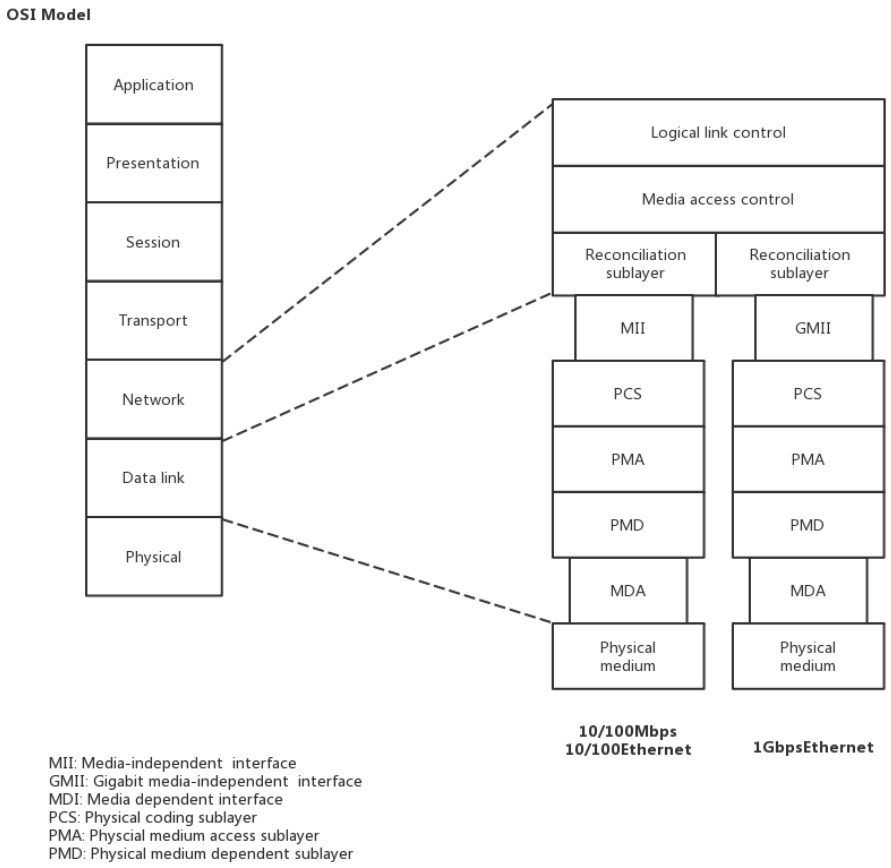


Fig. 4 OSI model

According to ISO/OSI seven layers model, the key of ethernet technology is about data link layer and physical layer. As illustrated in IEEE 802.3 standard, data link layer includes MAC (media access control) sublayer, and LLC (logical link control) sublayer. MAC sublayer performs the functions of packaging frames and media access, while the LLC sublayer binds the data package with destination address service access point and source access point to ensure the data package can be transported between different types of networks.

The Reconciliation sublayer is responsible for mapping GMII or MII data and control signal to the MAC interface [6]. GMII (Gigabit media-independent interface) and MII (Media-independent interface) achieve the interface protocol between upper protocol and physical chip. MDI (media dependent interface) is the interface between physical chip and physical medium. PCS, PMD and PMA are sublayers that achieve the physical layer protocols. In practice, the operation of these sublayers will be controlled by the physical chip, and users only need to design and config the GMII or MII correctly. Generally, the PCS sublayer is responsible for 8b/10b coding, which codes the 8 bits parallel data that the GMII or MII receives to 10 bits parallel data output [7]. After that, PMA sublayer transmits the coded data

to different types physical medium. Finally, the PMA sublayer builds the data link according to the physical medium that is used.

2.1.3 Ethernet data unit

In ethernet protocol system, the MAC sublayer and LLC sublayer perform the function of data link layer in OSI seven layers model together. The data link layer provides reliable and stable data transport by packaging the data into standard ethernet frames, where the MAC protocol is the key technique.

Basically, the MAC layer performs two functions. The first one is packaging and unpackaging ethernet frames. In MAC layer, information including the frame header, MAC address, frame check sequence and frame type is added to the data that the previous protocol stack packaged, which ensures the ethernet data frame can be conveyed from data link layer to physical layer correctly. Another function is access control, including flow control, MAC address filtering and control of physical layer, etc.

In ethernet protocol, there are two types of transmission modes, the half-duplex mode and full-duplex mode. The difference between these two modes is that in the half-duplex mode CSMA/CD is required, while in full-duplex mode we only need to insert a small interval between the data frames transmissions [8].

The ethernet frame is unpackaged when it is received. According to the standard, the MAC controller only receives data frame when the frame starting delimiter is received. The receiving procession is illustrated below:

Filter out data frames whose length does not meet the minimum length.

Identify if the destination address that the data frame provides is same as the local address. If not, the data frame is abandoned.

If the destination address is same as the local address, then unpackage the data frame and convey the data to high level protocol stack.

When using TCP or UDP as communication protocol, the data will be transported via protocol stacks in the TCP/IP model. The data will be packaged in order according to the

protocol requirements. When transported to the next layer, a message head and a tail will be added to the package. The message format for each layer is show in the figure below:

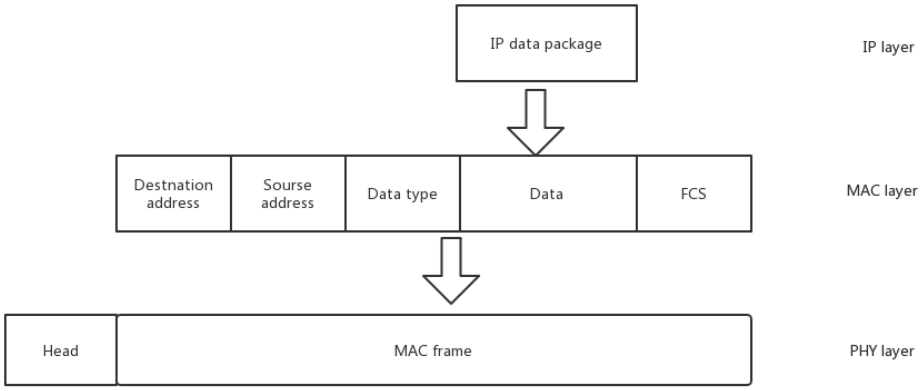


Fig. 5 Data message format for different layers

In the Mac layer, the destination MAC address, the source MAC address, the data type as well as the FCS are added on the data. The destination and source MAC address specify the MAC address of the transmitter and the receiver devices, whose length are 6 Bytes individually. The data type (2 Bytes) specifies the type of the implemented Ethernet protocol. The FCS (Frame Check Sequence) uses CRC to check the data between the destination MAC address to the end of valid data, and then check if errors occur in the data frame.

With respect to the PHY layer, the head of the data frame is composed of two parts. The first part is the Permeable, whose length is 7 Bytes. The content of each Byte is 0x55 (1010 1010, from right to left). The Permeable synchronizes the data frames. The second part is SFD (Start Frame Delimiter), consisting of 1 Byte 0x5D, whose purpose is to declare the start of a data frame.

In conclusion, to receiver the Ethernet package from the other endpoint and get the control command, firstly the SFD should be identified. After it is confirmed, the MAC frame is unpackaged by removing the MAC addresses and the Data type Byte and FCS, and then the IP message can be acquired.

2.1.4 TCP/IP Protocol and Socket

OSI seven layers model is designed by ISO as a concept network model. On the contrary, TCP/IP network model is designed to implement internet [9]. Different from OSI seven layers model, TCP/IP protocol [10] only contains 4 layers: application layer, transport layer, internet layer and host-to-network layer. The corresponding relationship between the two models is shown in the figure below:

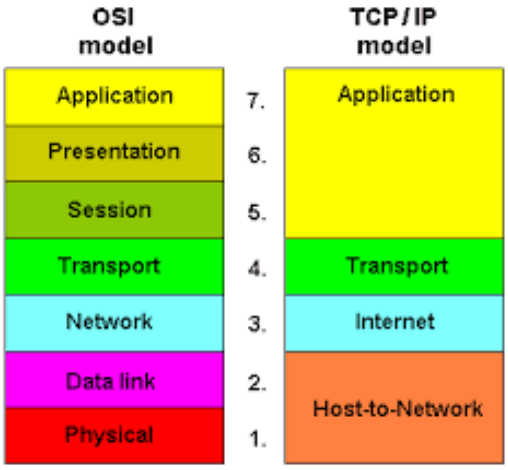


Fig. 6 The relationship between OSI 7 layers model and TCP/IP network model

In TCP/IP network model, the application layer provides interface for user applications, e.g. email service. The transport layer provides end-to-end communication service for the application layer. In this layer two protocols are defined: transmission control protocol (TCP) and user datagram protocol (UDP) [11]. The internet layer is the core of the TCP/IP model, which performs the same function as the MAC layer in the OSI model. This layer packages the TCP segment or UDP segment with destination IP address and source address. Finally, the host to network transmits data message given the exact medium, for example, ethernet, WIFI or optical fiber.

With regard to TCP and UDP. UDP provides connectionless, fast and unreliable communication service, which is broadly applied for web multimedia service since it requires high response speed but is not sensible to data lost. On the other hand, TCP is connection-orientated and provided reliable communication service. It ensures all data segments comes to the correct destination address and be reordered to the correct data sequence.

Fig.6 shows the process of building a TCP connection. Firstly, the client requires to establish a connection with the server, and transmits a Synchronize Sequence Number (SYN). Once the server receives this request, it responds the client with the acknowledgement (ACK) that it received the SYN as well as a new SYN. After that, when the client receives the ACK and SYN, it sends the ACK to the server that it is ready to establish a server. This process is called three ways handshake [12]. When breaking the connection, the TCP/IP protocol uses four ways handshake. There is another handshake between the server sending ACK and the client sending ACK [13] because that the protocol allows the server transmitting data once after it sends ACK.

Socket is an API provided by Linux operation system for network programming [14]. It can be described as an abstract layer between the application layer and the transport layer. By applying socket, users only have to choose the protocol that is used in the transport layer, while the lower layers protocols will be implemented by the operation system itself.

The procedure of communication using socket with TCP protocol is shown in Fig.. For both endpoints of the server and client, the interface socket() is used to create a new socket connection, where the return value is the descriptor of the connection. The function bind() is used to bind the IP address and the port number together. The process is blocked until requests are arrived from clients by using listen(). By using connect(), the client request a new connection to the server by identifying the destination IP address and the port number, and this step is corresponding to the first handshake that illustrated previously. After that, the function accept() is executed to respond the request, and the process will be blocked, which is corresponding the second handshake. After the client confirmed the accept() is executed, it will send a new SYN, and in this way the three-way handshake is completed.

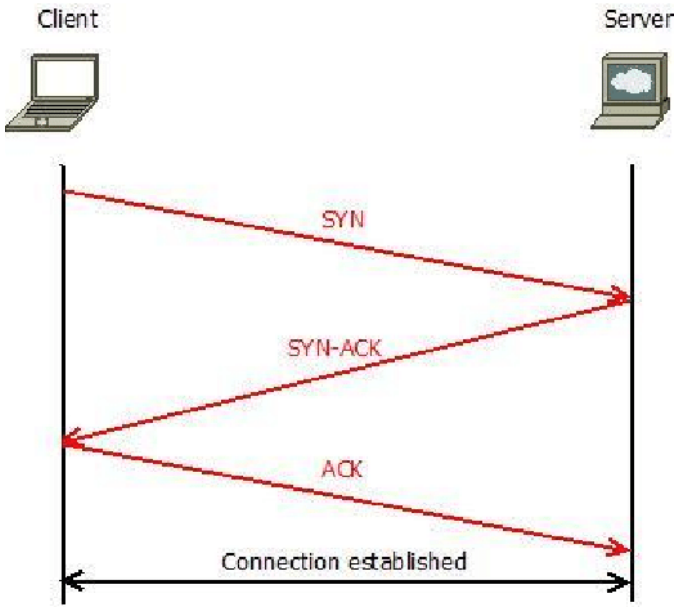


Fig. 7 Three ways handshake

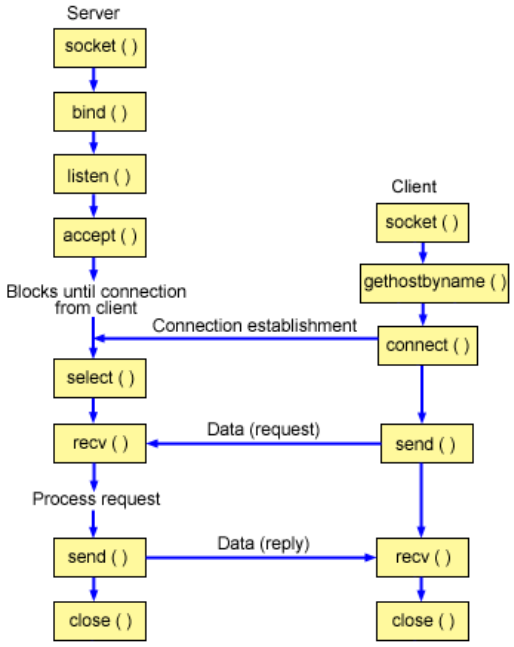


Fig. 8 Socket connection

2.2 Servo motor control

2.2.1 Servo motor

Servo motors have a lot of different types and are used widely in different areas. It can control the position and rotation speed precisely. It is a closed-loop system which control the

motion and final position by using the position feedback [15]. The figure below shows the closed-loop system of servo motor [15].

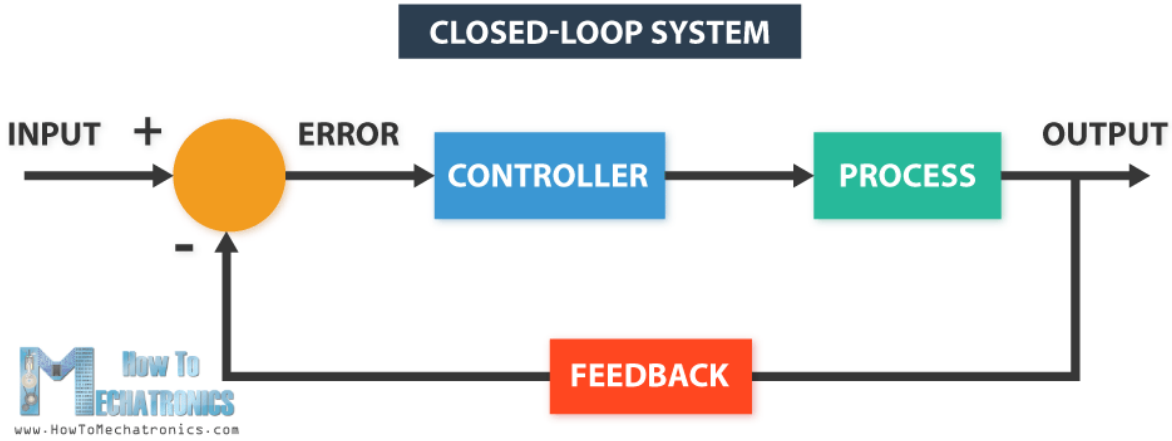


Fig. 9 Closed-loop system of servo motor

To compare the industrial servo motors and smaller RC or hobby servos, the feedback sensor of industrial servo motors is a high precision encoder, and the sensor for hobby servos is usually a simple potentiometer [15]. These sensors capture the actual position and feedback it to the error detector to compare with the target position [15]. Then the controller will correct the current position of motor to match the target position [15].

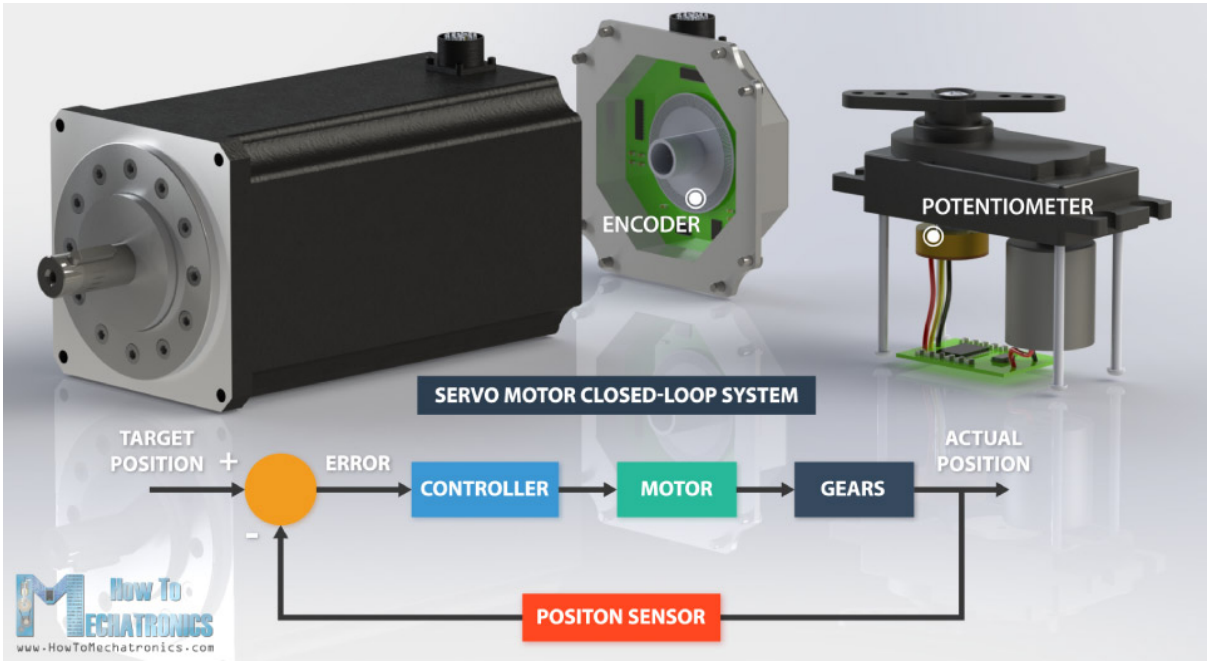


Fig. 10 Compare the industrial servo motors and smaller RC or hobby servos

In this project, the hobby servo has been used. The type is MG996R. The material object is showed below:

Fig. 11 MG996R Servo Motor

MG 996R is a high-torque digital servo motor which has metal gears [16]. And it has 10 kg stalling torque with small size device [16]. The servo motor includes 30 cm wire and 3 pins [16]. It is suitable for most receivers. The motor can rotate about 120 degrees by using any servo code [16].

The specifications of MG996R servo motor has been shown below [16]:

Table 1 The specifications of MG996R Servo Motor

Specifications	
Weight:	55 g
Dimension:	40.7 x 19.7 x 42.9 mm approx.
Stall torque:	9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
Operating speed:	0.17 s/60o (4.8 V), 0.14 s/60o (6 V)
Operating voltage:	4.8 V a 7.2 V
Running Current	500 mA – 900 mA (6V)
Stall Current	2.5 A (6V)
Dead band width:	5 µs
Temperature range:	0 °C – 55 °C
Stable and shock proof	double ball bearing design

The Fig.12 [16]below shows the three lines of the servo motor MG 996R. The three lines are orange, red and brown. They represent PWM, Vcc, Ground respectively.

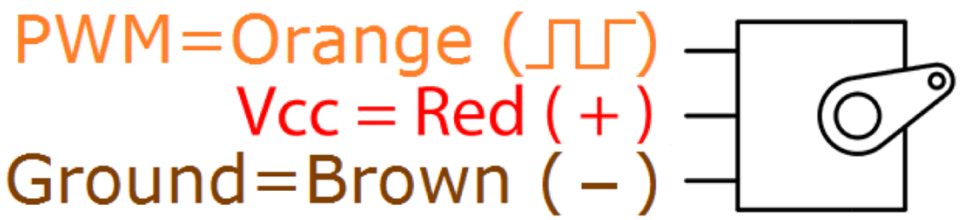


Fig. 12 Three lines of MG996R Servo Motor

Fig.13[15] shows using PWM to control the rotation of the servo motor.

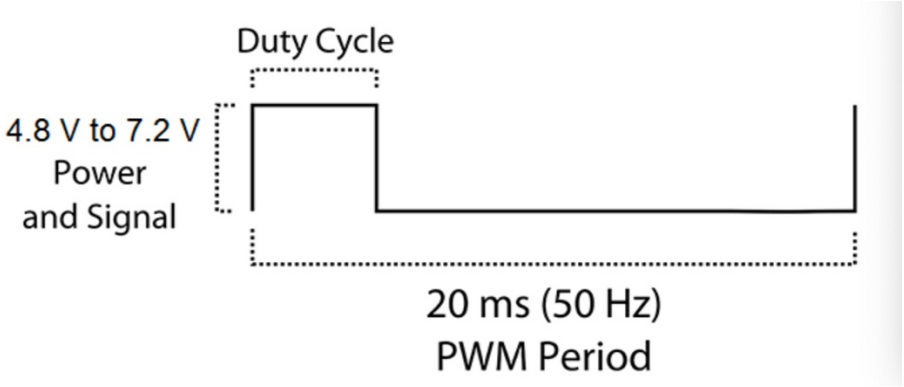


Fig. 13 PWM control of MG 996R

The frequency of PWM signal is 50Hz, and the PWM period is 20ms. It is digital signal with 4.8V to 7.2V power.

Then, to understand how servo motor works physically, the figure below shows the schematic graph of hobby servo motor [15].

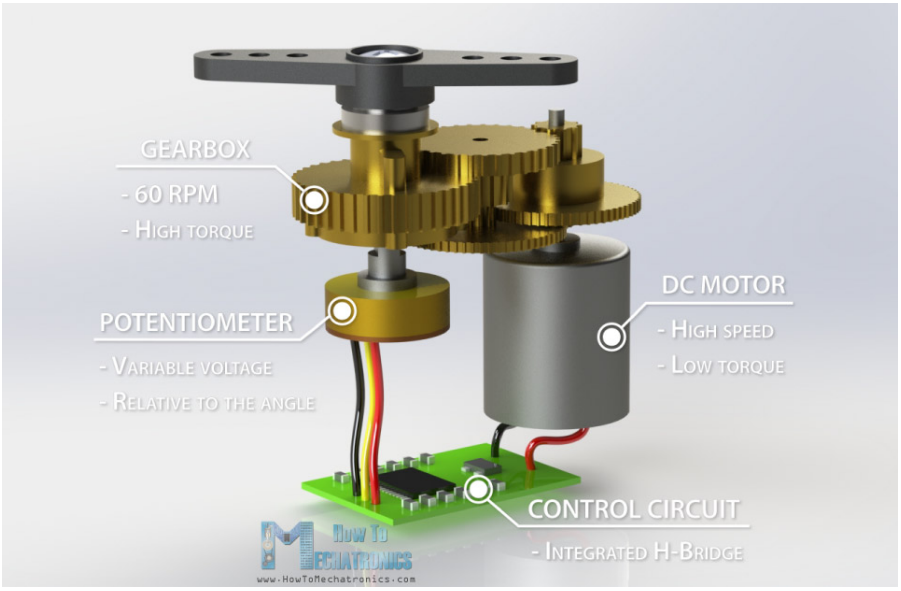


Fig. 14 The schematic graph of hobby servo motor

There are four main parts in a hobby servo, a DC motor, a gearbox, a potentionmeter and a control circuit [4]. The DC motor has high speed but has the low torque. Then the gearbox will reduce the speed from DC motor to around 60 RPM and increase the torque [15].

The potentiometer is putted on the final gear or output shaft and it has the same output rotation angle [15]. Thus, potentiometer can output a voltage which is related to the output absolute angle. Then, control circuit will compare the potentiometer voltage and input signal's voltage [15]. An integrated H-bridge may be activated to make the motor rotate to the target position [15]. When the actual position of servo motor is the target position, the difference between potentiometer voltage and input signal's voltage will be zero.

2.2.2 PWM Control

PWM control is Pulse width modulation control which is a method that chop an electrical signal up into discrete parts to control the average power [17].

The figure below shows using PWM to control the servo motor [15].

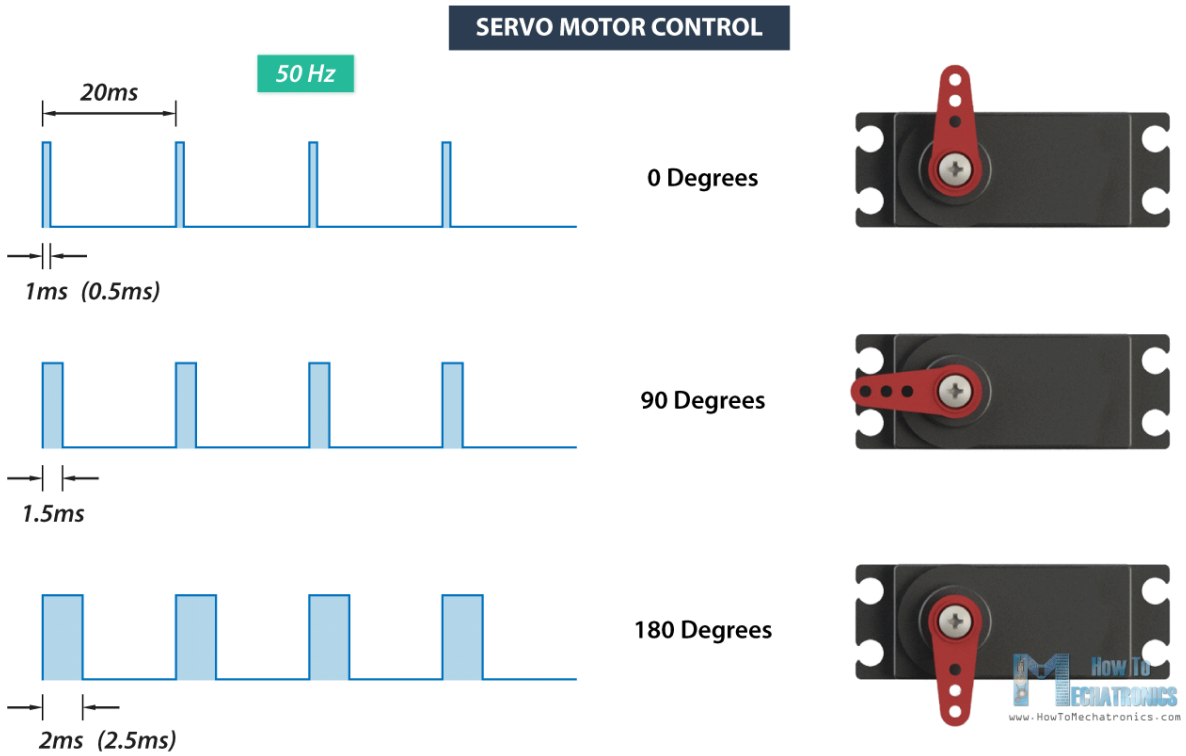


Fig. 15 PWM control of servo motor

To control the servo motor, the signal line with a series of pulses will be generated depending on the target position and speed [15]. The frequency of the control is 50Hz and the period time is 20ms [15]. Then the width of pulse is from 1ms (0.5ms) to 2ms (2.5ms) [4]. Different

width of pulse will correspond the specific angle of servo motor. 1ms – 2ms for the width of pulse is 0 – 180 degrees of servo motor.

In this project, the servo motor needs to be controlled to the specific position and with specific speed through Ethernet in real time. Thus, the command should include the target position and speed. The figure below shows the control system in FPGA and Servo Motor Kit part [18].

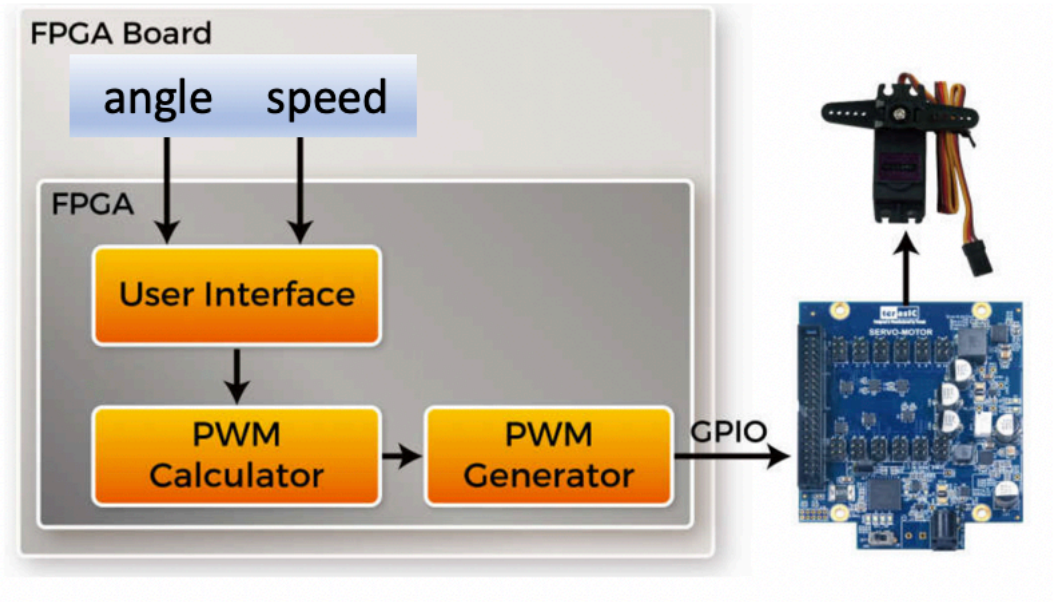


Fig. 16 PWM control in this project

In FPGA, the command which includes target angle and speed will be transported from HPS part. Then the FPGA board will calculate the parameters of PWM signal depending on the target angle and speed. After that, the PWM generator will generate the PWM signal and transport it to Servo Motor Kit through GPIO port. The servo motor in Servo Motor Kit will be controlled and move to the target angle in target speed.

2.3 Introduction to Qsys

SOPC (System on a Programmable Chip) refers to the use of programmable logic technology to put the entire system on a piece of silicon [19].

SOPC is a special embedded system. On the one hand, it is a system-on-a-chip (SOC), which is the main logic function of the whole system by a single chip; on the other hand, it is a programmable system with flexible design, can be cut, expanded, upgraded, and has

hardware and software. Programmable features in the system. This technology combines technologies such as EDA, computer design, embedded systems, DSP and digital communication systems.

2.3.1 Software and hardware requirements [20]

- Altera QuartusII V11.0 or higher software
- For system requirements and installation instructions, please refer to Altera Software Installation and Licensing.
- Nios II EDS V11.0 or higher
- Altera's Arria, Cyclone, or Stratix series FPGAs must be included on the board.
- The FPGA must include at least 12K logic elements (LEs) or adaptive lookup tables (ALUTs).
- The FPGA must include at least 150Kb of embedded memory.
- There must be a JTAG interface on the board for the FPGA internal memory program execution on the host monitor board.
- The board must include memory for design testing. For example, any Qsys controller-based memory with an Avalon-MM slave interface.

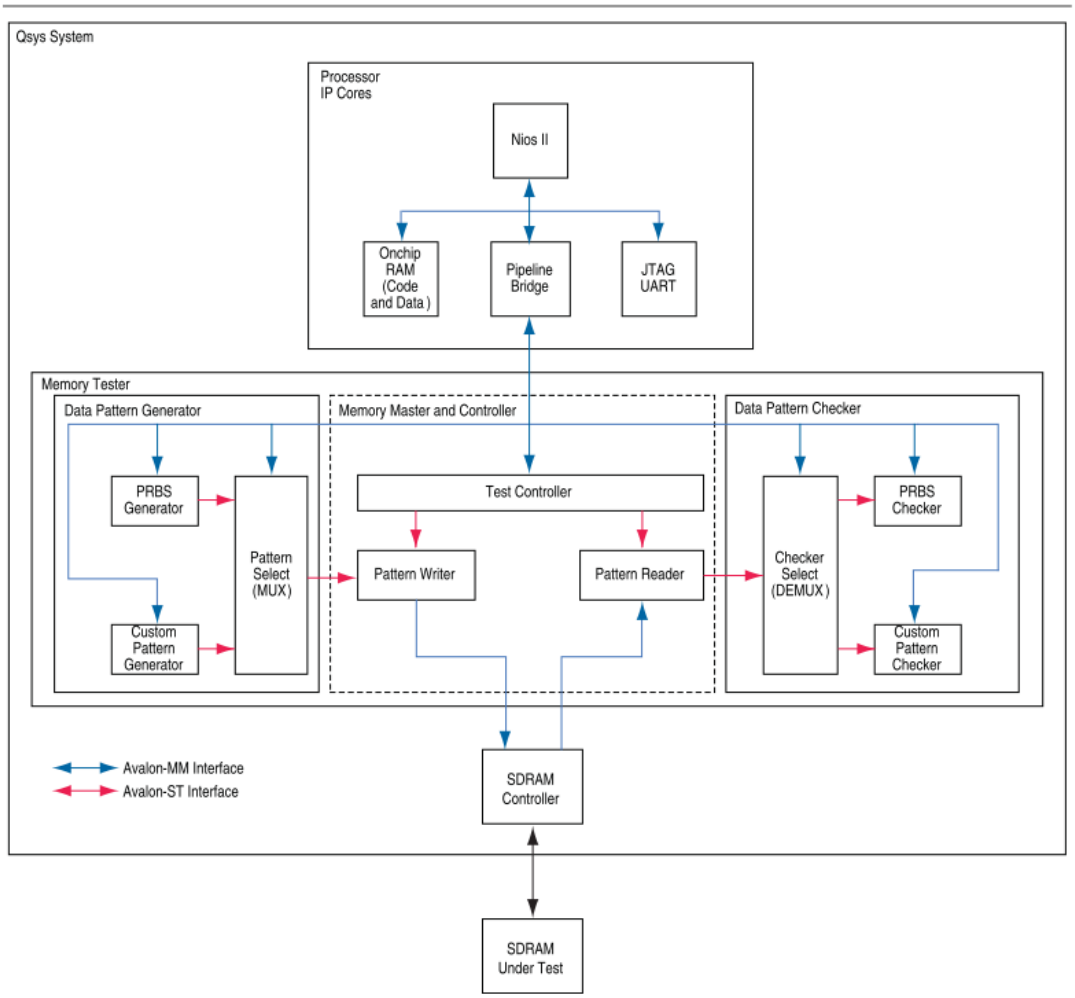


Fig. 17 Top level system structure

2.3.2 Qsys advantages

a. Accelerate development

- Easy to use GUI interface to support rapid integration of IP functions and subsystems.
- Automatic generation of interconnect logic (address/data bus connections, bus width matching logic, address decode logic, arbitration logic, etc.)
- Plug and Play Qsys compatible IP from Altera and its IP partners

b. System HDL is automatically generated

- Layered design process for flexible design, support for team-based design, and improved design reuse

c. Faster timing closure

- Compared to the SOPC Builder system interconnect architecture, the high-performance Qsys interconnect based on the NoC architecture and the automated pipeline double the performance [21].
- Control powerful automatic pipelines to meet fMAX and delay system requirements.

d. Complete verification faster

- Start simulation quickly with automated test bench generation and use the verification IP suite
- Accelerate board development by sending and reading system-level operations through the system console

2.3.3 Signal settings for HPS and FPGA interfaces [22]

a. General purpose interface

MPU standby and event signals ---FPGA obtains the status of the MPU through this signal

MPU general purpose signals ---- One-way universal signal from FPGA manager to FPGA logic inside HPS

FPGA Cross Trigger interface --- The interaction trigger signal between the FPGA and the HPS. When the FPGA or HPS is triggered, another device will save the current working state.

BOOT FROM FPGA signals--- Configure the interaction signal when the preloader is booted from the FPGA, determine whether the preloader is ready, or other signals that the FPGA participates in debugging.

b. AXI Bridge

The AXI Bridge is a set of signals that communicate between the HPS and the FPGA via L3 interactive logic. An interface that includes three different sets of functions is a portal that works together using SoC FPGA.

c. FPGA-to-HPS SDRAM Interface

This is a set of interfaces for FPGA logic to access HPS'DDR3 via HPS' SDRAM controller. The FPGA logic requests the HPS to enter the reset state signal. After the HPS enters the reset, it will notify the FPGA through the HPS-to-FPGA cold reset output.

FPGA logic requests FPGA internal DMA transfer channel interface

Interrupt The interface is divided into two categories, one is the interrupt manager that the FPGA initiates the interrupt to the HPS, and the other is the peripheral interrupt signal of the HPS to the FPGA.

2.3.4 Peripheral pin multiplexing

This interface is set for pin multiplexing, and related settings are made for each peripheral in the HPS according to the circuit diagram.

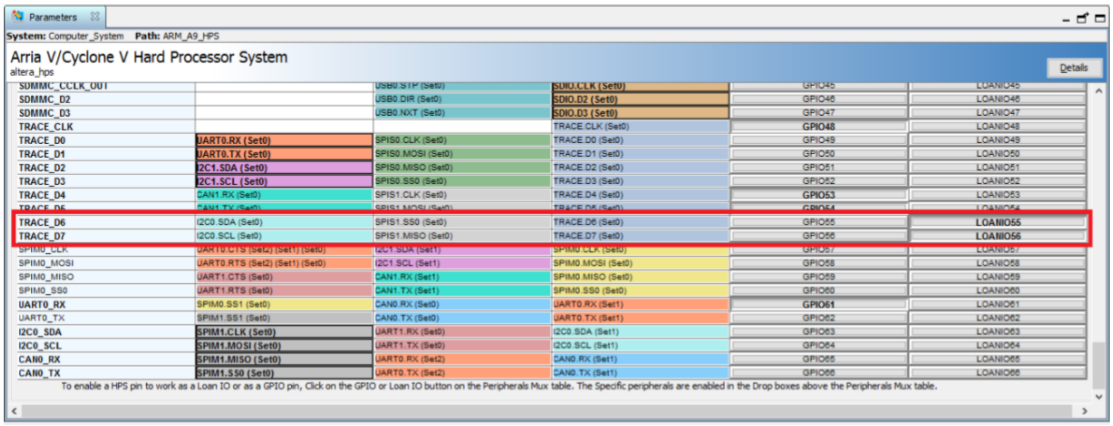


Fig. 18 Peripheral pin multiplexing

2.4 Introduction to AXI Bus

AXI is an advanced extension interface, proposed in AMBA 3.0, and AMBA 4.0 upgrades to AXI4.0. AMBA4.0 includes AXI4.0, AXI4.0-lite, ACE4.0, AXI4.0-stream [23].

AXI4.0-lite is a simplified version of AXI, ACE4.0 is an AXI cache coherency extension interface, AXI4.0-stream is proposed by ARM and Xilinx, mainly used in FPGA for data-driven transmission of large amounts of data. application.

2.4.1 AXI architecture

The AXI protocol is based on burst transmission and defines the following five independent transmission channels: read address channel, read data channel, write address channel, write data channel, write response channel.

The address channel carrying control message is used to describe the data attributes to be transmitted. The data transmission uses a write channel to implement the "master" to "slave" transmission, the "slave" uses the write response channel to perform a write transfer, and the read channel is used to implement the data. The transmission from "from" to "main".

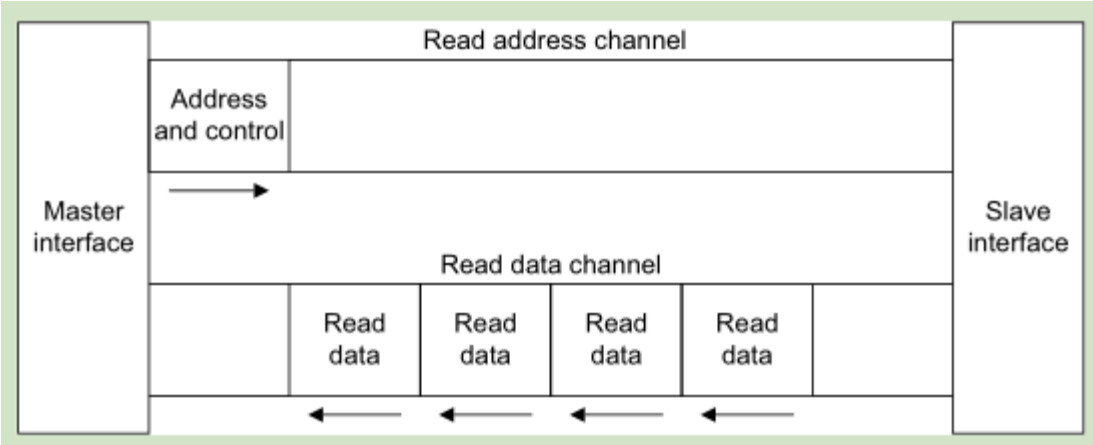


Fig. 19 Read architecture

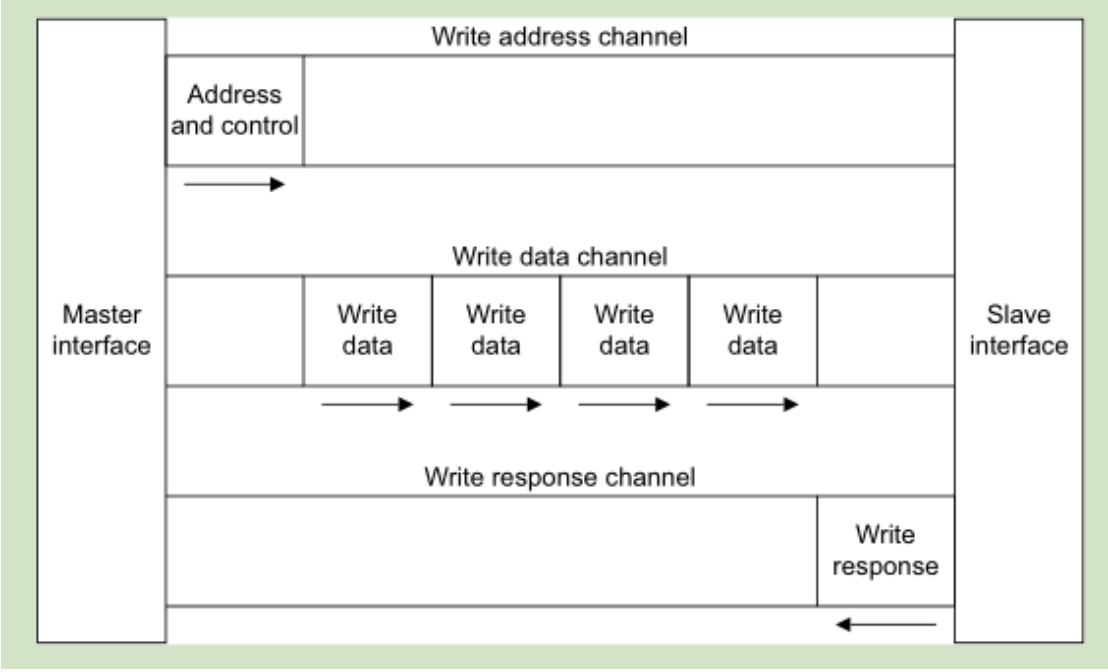


Fig. 20 Write architecture

AXI is a VALID/READY-based handshake mechanism data transmission protocol. The transmission source uses VALID to indicate that the address/control signal and data are valid. The destination uses READY to indicate that it can accept the information.

Read/write address channel: Each of the read and write transmissions has its own address channel, and the corresponding address channel carries the address control information corresponding to the transmission.

Read data channel: The read data channel carries read data and read response signals including a data bus (8/16/32/64/128/256/512/1024bit) and a read response signal indicating completion of the read transfer.

Write Data Channel: The data information of the write data channel is buffered. The "master" can initiate a new write transfer without waiting for the "slave" acknowledgement of the last write transfer. The write channel consists of a data bus (8/16...1024bit) and a byte line (used to indicate the validity of the 8-bit data signal).

Write Response Channel: "Client" responds to a write transfer using a write response channel. All write transfers require a completion signal to write the response channel.

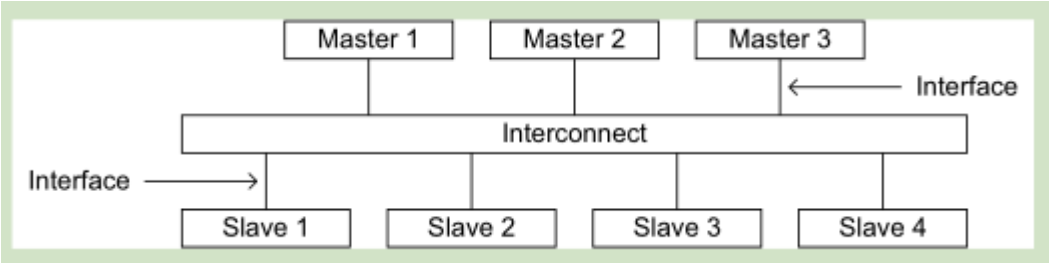


Fig. 21 Interface and interconnection

The AXI protocol provides a single interface definition that can be used between the following three interfaces: master/interconnect, slave/interconnect, master/slave.

The following typical system topology architectures are available:

• Shared address and data bus

• Shared address bus, multiple data bus

• multilayer multi-layer, multi-address bus, multi-data bus

In most systems, the bandwidth requirements of the address channel are not as high as the data channel, so a shared address bus, multi-data bus structure can be used to balance system performance and interconnect complexity.

Register Slices:

Each AXI channel transmits information in a single direction, and each channel does not have any fixed relationship directly. It is therefore possible to insert a register slice at any point in any channel, which of course leads to additional cycle delays.

The use of register slices allows for a compromise between cycles of latency and maximum operating frequency; the use of register slices allows the long path of low-speed peripherals to be split.

2.4.2 Signal description

Table 2 Global signal

Signal name	source	description
ACLK	clock source	global clock signal
ARESETn	reset source	global reset signal, low effective

Table 3 Write address channel signal

Signal name	source	description
AWID	Host	Write address ID to mark a set of write signals
AWADDR	Host	Write address, giving the write address of a write burst transfer
AWLEN	Host	Burst length, gives the number of burst transfers
AWSIZE	Host	Burst size, giving the number of bytes per burst transfer
AWBURST	Host	Burst type
AWLOCK	Host	Bus lock signal that provides atomicity of operation
AWCACHE	Host	Memory type, indicating how a transfer is through the system

AWPROT	Host	Type of protection, indicating the privilege level and security level of a transmission
AWQOS	Host	Quality service QoS
AWREGION	Host	Area flag, which can implement multiple logical interfaces corresponding to a single physical interface
AWUSER	Host	User-defined signal
AWVALID	Host	A valid signal indicating that the address control signal for this channel is valid
AWREADY	Slave	Indicates that "slave" can receive the address and corresponding control signal

Table 4 Read address channel signal

Signal name	source	description
ARID	Host	Read address ID, used to mark a set of write signals
ARADDR	Host	Read address, giving a read address for a write burst transfer
ARLEN	Host	Burst length, gives the number of burst transfers
ARSIZE	Host	Burst size, giving the number of bytes per burst transfer
ARBURST	Host	Burst type
ARLOCK	Host	Bus lock signal that provides atomicity of operation
ARCACHE	Host	Memory type, indicating how a transfer is through the system
ARPROT	Host	Type of protection, indicating the privilege level and security level of a transmission
ARQOS	Host	Quality service QoS
ARREGION	Host	Area flag, which can implement multiple logical interfaces corresponding to a single physical interface
ARUSER	Host	User-defined signal
ARVALID	Host	A valid signal indicating that the address control signal for this channel is valid

ARREADY	Slave	Indicates that "slave" can receive the address and corresponding control signal

2.4.3 Signal interface requirements

Each AXI component uses a clock signal, ACLK, and all input signals are sampled on the rising edge of ACLK. All output signals must occur after the rising edge of ACLK.

AXI uses an active-low reset signal, ARESETn, which can be asserted asynchronously, but must be asserted in synchronization with the rising edge of the clock.

The interface has the following requirements during reset: host interface must drive ARVALID, AWVALID, WVALID is low; slave interfaces must drive RVALID, BVALID is low and that all other signals can be driven to any value. After reset, the host can drive ARVALID, AWVALID, and WVALID high on the rising edge of the clock.

2.4.4 basic read and write transmission

The five transmission channels use the VALID/READY signal to handshake the address, data, and control signals of the transmission process. Using the two-way handshake mechanism, the transfer only occurs when both VALID and READY are active. The following figure is a few handshake mechanisms: [24]

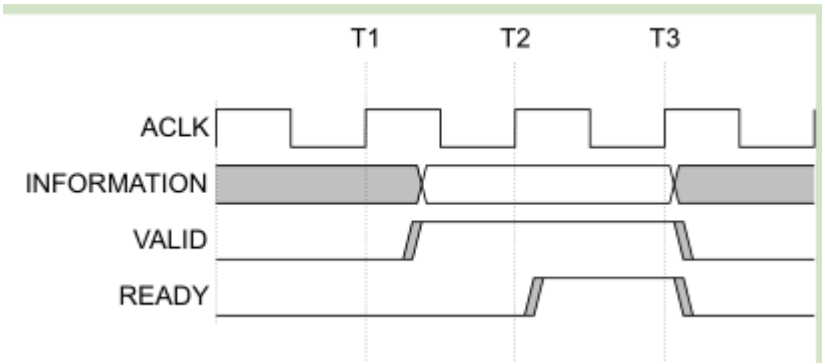


Fig. 22 VALID before READY handshake

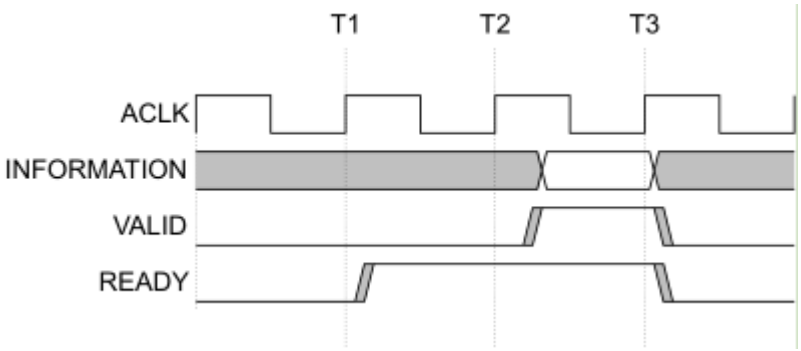


Fig. 23 READY before VALID handshake

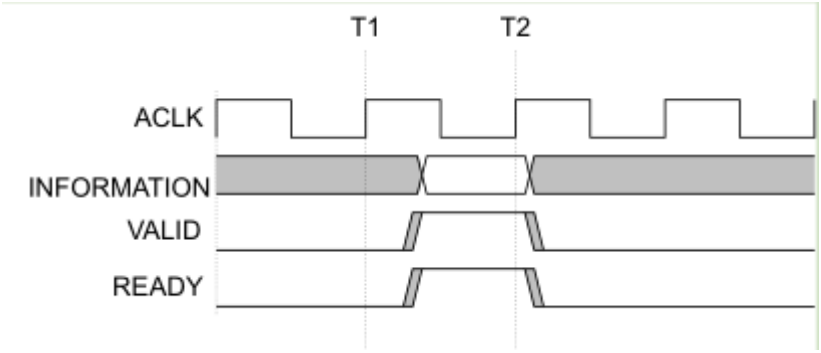


Fig. 24 VALID with READY handshake

2.4.5 Read and write response structure

The response information of the read transfer is appended to the read data channel, and the response of the write transfer is in the write response channel.

RRESP[1:0], read transfer

BRESP[1:0], write transfer

OKAY('b00): normal access succeeded

EXOKAY('b01): Exclusive access succeeded

SLVERR('b10): Slave error. Indicates that the access has been successful to the slave, but the slave wants to return an error condition to the host.

DECERR ('b11): Decoding error. Usually given by the interconnect component, indicating that there is no corresponding slave address.

2.4.6 Accessing HPS Devices from the FPGA

Table xxxx lists the devices built into HPS. These devices provide memory-mapped interfaces that map to addresses within the 32-bit (4GB) address space of the HPS interconnect. Any host device connected to the interconnect (such as a master device instantiated in an FPGA) can read and write to these interfaces at their respective addresses [18].

Table 5 Base address of built-in devices

Device	Interface	Base Address
SD/MMC Controller	sdmmc	0xFF704000
Quad SPI Flash Controller	qspiregs	0xFF705000
	qspidata	0xFFA00000
Ethernet Media Access Controller (EMAC)	emac0	0xFF700000
	emac1	0xFF702000
General Purpose I/O (GPIO) Controller	gpio0	0xFF708000
	gpio1	0xFF709000
	gpio2	0xFF70A000
NAND Flash Controller	nanddata	0xFF900000
	nandregs	0xFFB80000
USB OTG Controller	usb0	0xFFB00000
	usb1	0xFFB40000
CAN Controller	can0	0xFFC00000
	can1	0xFFC001FF
UART Controller	uart0	0xFFC02000
	uart1	0xFFC03000
I2C Controller	i2c0	0xFFC04000
	i2c1	0xFFC05000
	i2c2	0xFFC06000
	i2c3	0xFFC07000
Timer	sptimer0	0xFFC08000
	sptimer1	0xFFC09000
	osctimer0	0xFFD00000
	osctimer1	0xFFD01000
SDRAM Controller	sdr	0xFFC20000
DMA Controller	dmanonsecure	0xFFE00000
	dmasecure	0xFFE01000
SPI Controller	spis0	0xFFE02000
	spis1	0xFFE03000
	spim0	0xFFF00000
	spim1	0xFFF01000
On-Chip Memory	ocram	0xFFFF0000

2.4.7 Connecting an FPGA Master to the HPS Interconnect

Through the FPGA to HPS bridge, AXI or Avalon bus masters within the FPGA can be connected to the HPS interconnect. This connection can be established in the Qsys System Integration Tool by connecting the memory mapped master port of the master device to the AXI_Slave port of the hard processor system component named f2h_axi_slave.

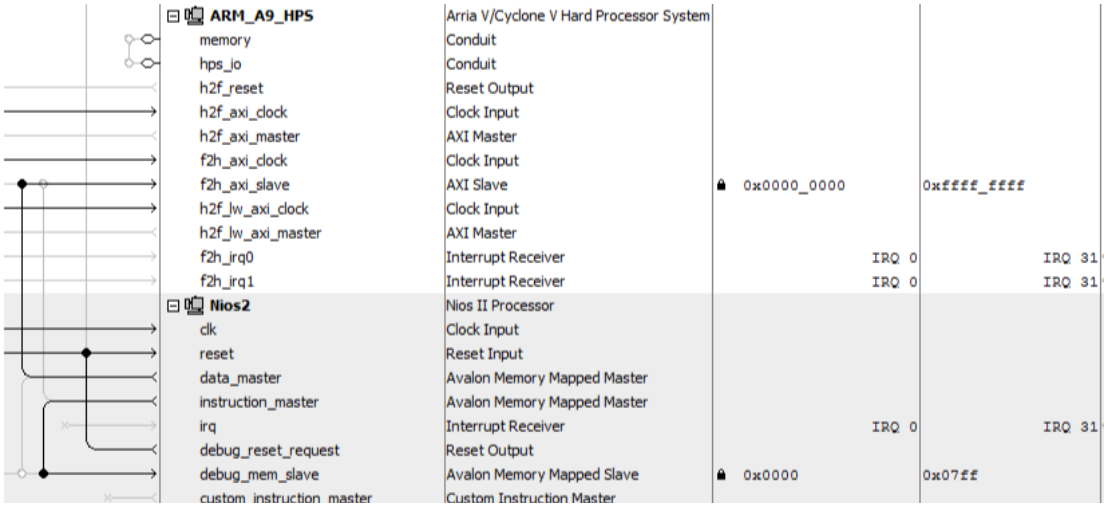


Fig. 25 Qsys GUI

2.4.8 Enabling the FPGA-to-HPS Bridge

Before the FPGA accesses the device, the FPGA is first enabled to the HPS bridge by invalidating the reset bit in brgmodrst. The brgmodrst register is located at address 0xFFD0501C in the HPS address space. The FPGA-side master server cannot access the HPS address space until the AXI Bridge is started, so the reset must be deasserted by the primary server inside HPS. Typically, this is done by running a bare metal program on the ARM Cortex A9 processor and writing a 0 to bit 2 of the brgmodrst register. After de-asserting the reset of the bridge, the FPGA-side master can access the complete 4GB address space through the FPGA to the HPS bridge for system-to-system communication.

3 Implementation

3.1 Design Overview

The aim of this design is to control the position and speed through FPGA via Ethernet. In this project, we use HPS on the de-Nano 10 board as the master controller while the FPGA as slave controller. C language is used as the programming language which is run on the Linux Operation System. To fulfill this task, two functions are essential to be implemented: the communication function and the control function. The flow chart of the running procedure is shown in Fig26.

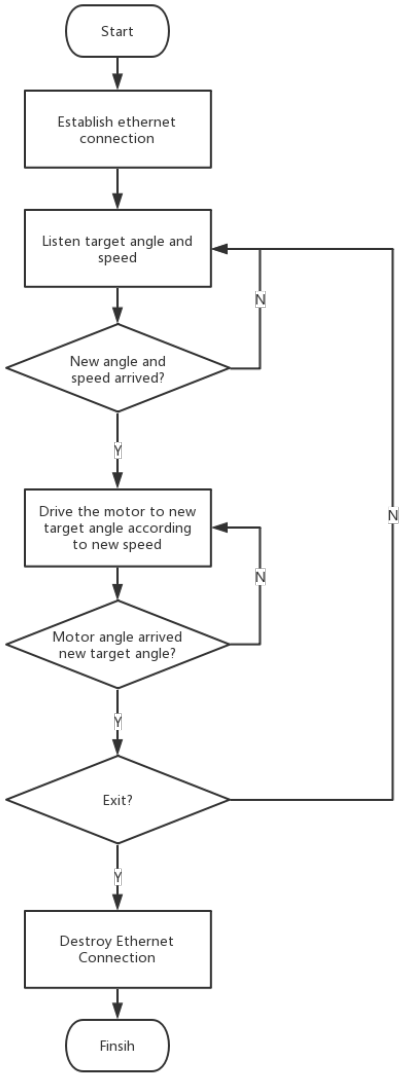


Fig. 26 Flow chart of the process

From the figure it can be seen that the first is to establish the ethernet connection between the HPS and the host computer. After that, the process will be blocked until the host computer sends a new message that includes target angle and speed. The next step is to drive the motor given the target angle and speed that mentioned before, and the process will still be blocked until the motor arrived the target angle. This step is followed by checking if the user wants to kill the process. If yes, then the connection will be destroyed to save the resource, while if not, it will go back to listen to the new command.

3.2 Communication function

As illustrated in Chapter 2, Socket with TCP protocol is implemented in this project to fulfill ethernet communication function. For Socket programming, both of the server and the client shall be programmed. In this design, we used the HPS as the server and the host computer as Socket client, as the server program can run without assigning the other endpoint's IP address and in this way we can make the program on the board as boot program. On the other hand, if we make the board as the client, then the program cannot run appropriately if the IP address of the host computer changes.

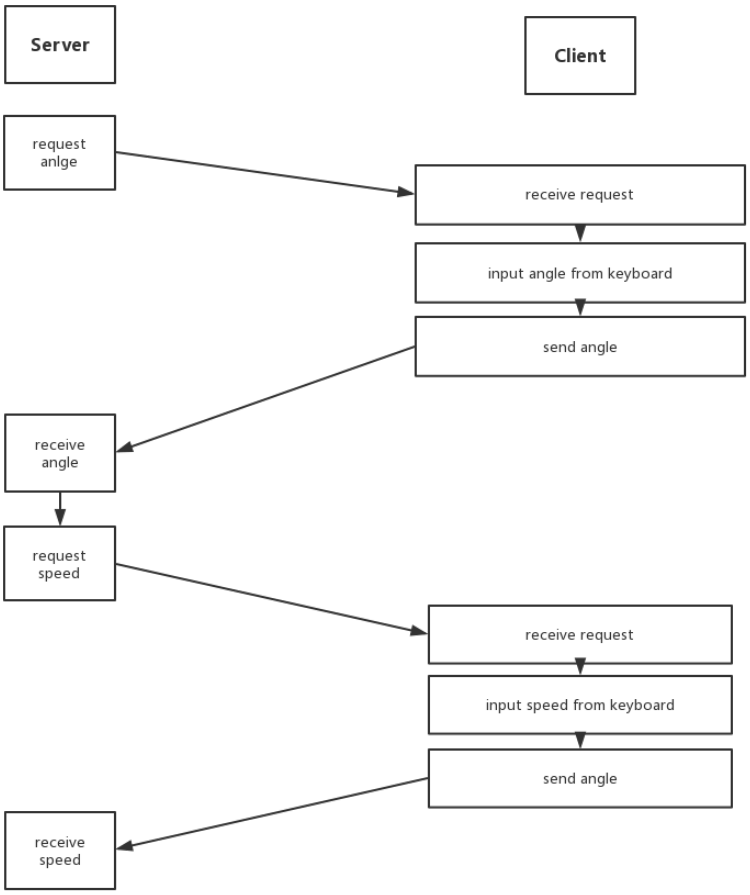


Fig. 27 Flow chart of Socket communication function

Fig .. shows the flow chart of the Socket communication function. The procedure of Socket connection establishing is demonstrated in Chapter 2, so here we only introduce the procedure of sending and receiving data between the endpoints. The process is illustrated as following: firstly, the server will request the target angle by sending a message with the content “Please input the target angle:”, and when the client receives this message, it will be printed on the screen so that the user will be clear what to do in the next step. After that, the user can use the keyboard on the computer to input the target angle. Once the inputting is finished, the client will send the value of target angle as a string, and then the server will receive this string and convert it back to a float number using atof() function. Up to now the communication procedure of requesting and receiving target angle is illustrated, and he procedure of this to target speed is similar expect that the request message is “Please input target speed:”. In this way, the target angle and speed is acquired. It is worth to mention that for both of the client and the server, all of the sending and receiving command use the same socket describer as handle, and they can be executed many times without establishing new sockets, since the socket describer is valid until the connection is closed.

3.3 Motor control function

As illustrated in Chapter 2, PWM is implemented to control the angle of the motor in this design. However, for servo motor there is only relationship between the output angle of the motor and the high duty cycle in PWM, and the motor will goes to the target angle with highest speed, while one requirement of this design is to control the speed. To fulfill this goal, the set angle for each total cycle of PWM is not the target angle itself, but an incrementing angle. As a result, by controlling the incrementing speed can the speed of the motor be controlled. The block diagram of PWM control is shown in Fig ..

In Fig .., if the current angle of motor has arrived the target angle, the procedure will finish directly, and if not, the following procedure will be: firstly increment or decrement the current angle which depends on if the current speed is higher than the target angle or lower. After that, the high duty cycle will be calculated according to the current angle. Next, the output signal to control the motor will be set to high level until the high duty cycle is finished, and for the rest time of the total cycle the output will be set to low level. When this finished, the state goes back to checking if the current angle is equal to the target angle, which will decide if the process will exit or continue running.

With respect to assigning the output signal to the motor, the output signal can be directly wrote to the corresponding register. To access this register, firstly a virtual RAM address space is created with mmap function in Linux, where user can decide the capacity of this space and the start address. After that, the address of the GPIO which controls the servo motor is this virtual address added by the offset of GPIO port to the AXI bridge. Finally, the output signal can be assigned to the pointer that points to this address, and in this way the servo motor is controlled.

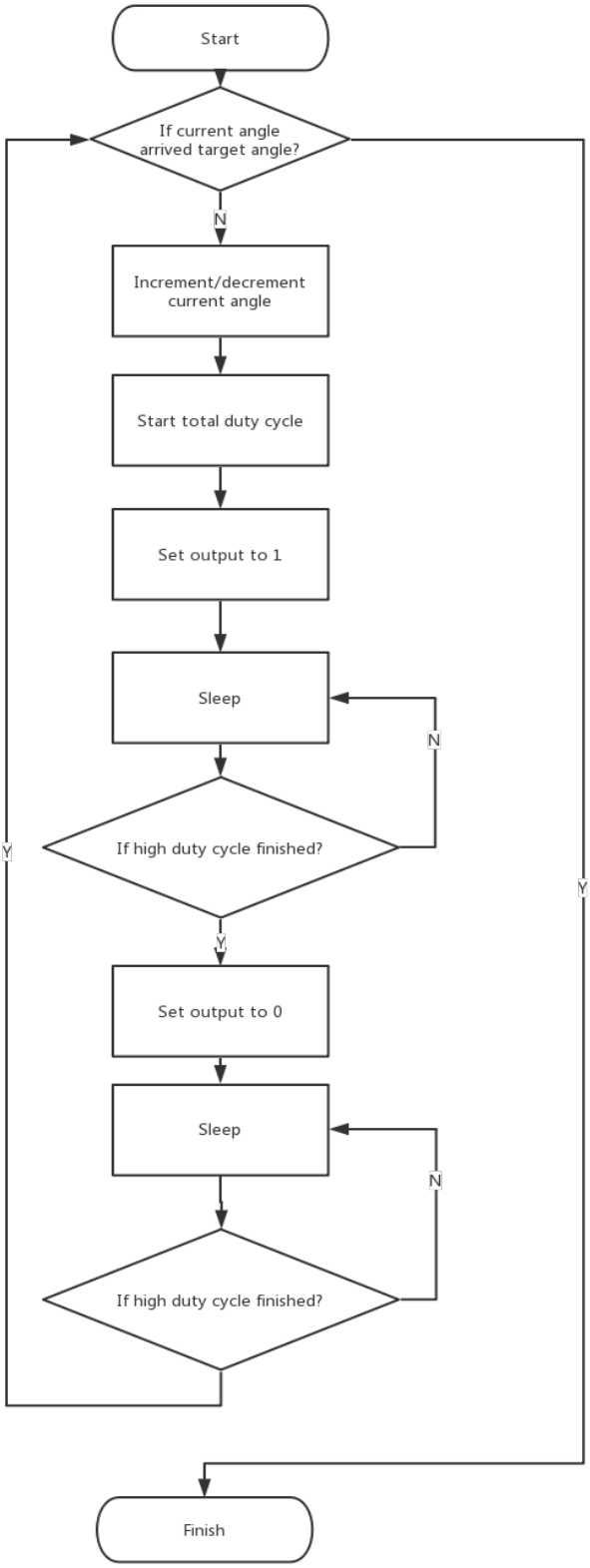


Fig. 28 Flow chart of PWM control using timer

3.4 Building connection between HPS and FPGA

Reference Project: DE10_NANO_SoC_GHRD [3]

The reference project consists of the following components: ARM Cortex™-A9 MP Core HPS, Two user push-button inputs, Four user DIP switch inputs, Seven user I/O for LED outputs, JTAG to Avalon master bridges, Interrupt capturer for use with System Console, System ID

a. Click the platform designer icon on the interface to enter the platform designer interface.

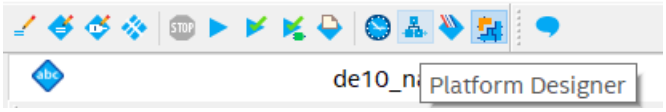


Fig. 29 Platform Designer icon

b. Read the GHRD Qsys file: soc_system.qsys, you can see the Qsys already designed as shown below.

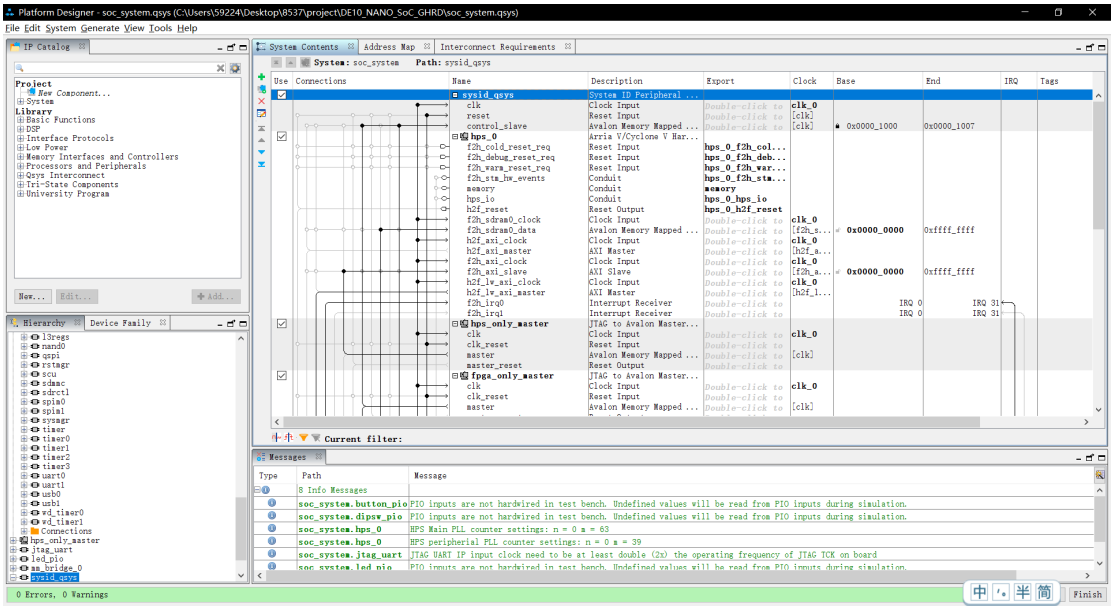


Fig. 30 Qsys interface

c. Add peripherals

Double-click clk_0 to set the parameters. In general, the clock in this is the external input clock, and the frequency is known, so check the “Clock frequency is known” to declare the frequency known.

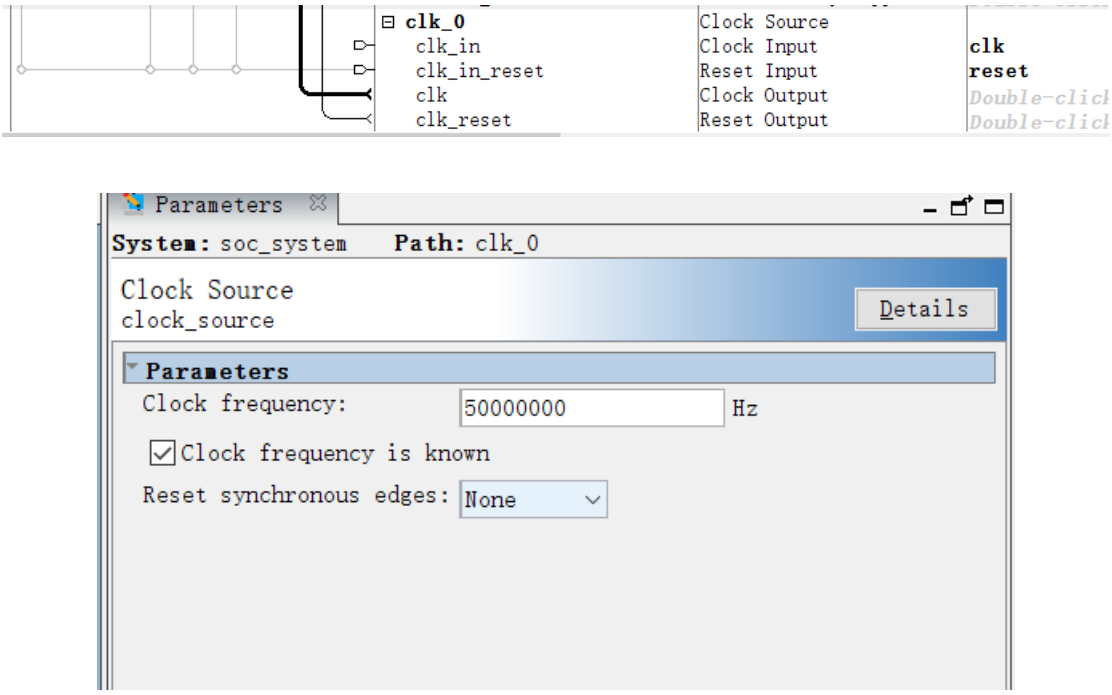


Fig. 31 Set the clock source

d. Modify the relevant parameters of the HPS core.

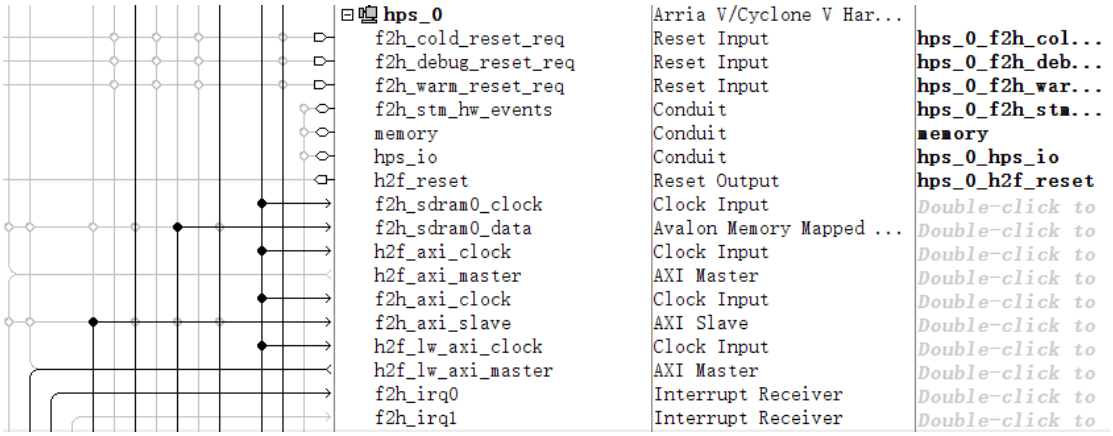


Fig. 32 Set the parameters of HPS

e. Set the bandwidth of AXI Bridges.

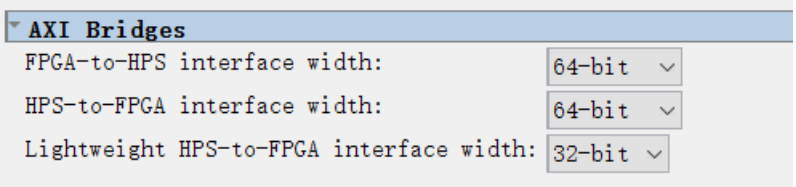


Fig. 33 Set the parameters of AXI Bridge

f. Set the parameters of the Peripheral pins that need to be used in the project. Here we use the Ethernet Media Access Controller and UART controllers.

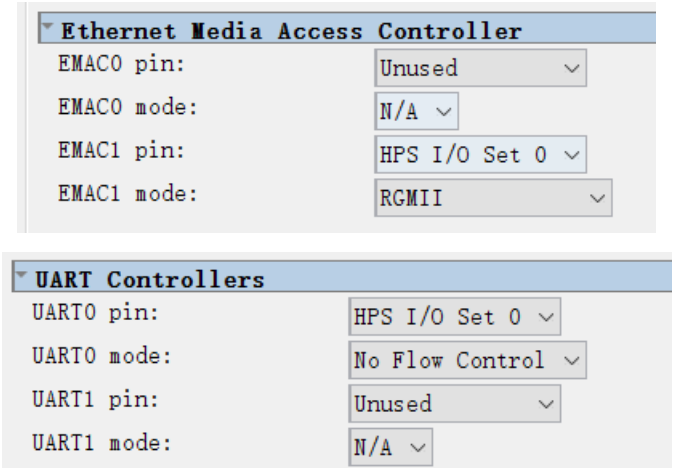


Fig. 34 Set the peripheral pins

g. Set the external clock sources to 25MHz

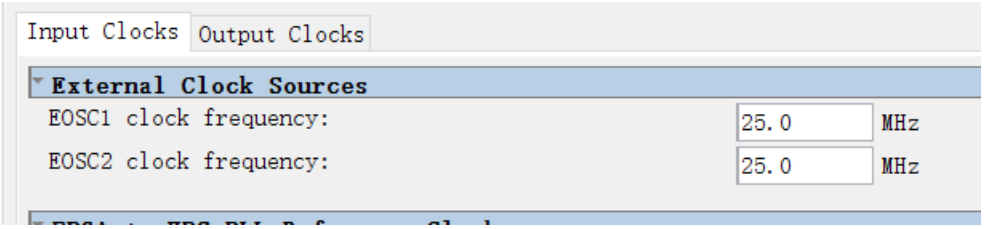


Fig. 35 Set the external clock sources

h. Set the Memory clock frequency to 400MHz and select it according to the type of memory.

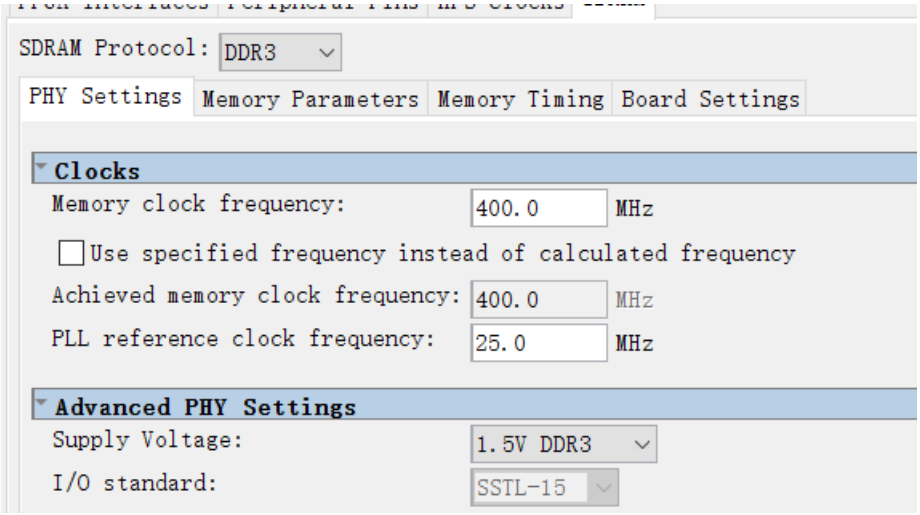


Fig. 36 Set the memory clock frequency

i. Set memory timing according to the parameters of the memory

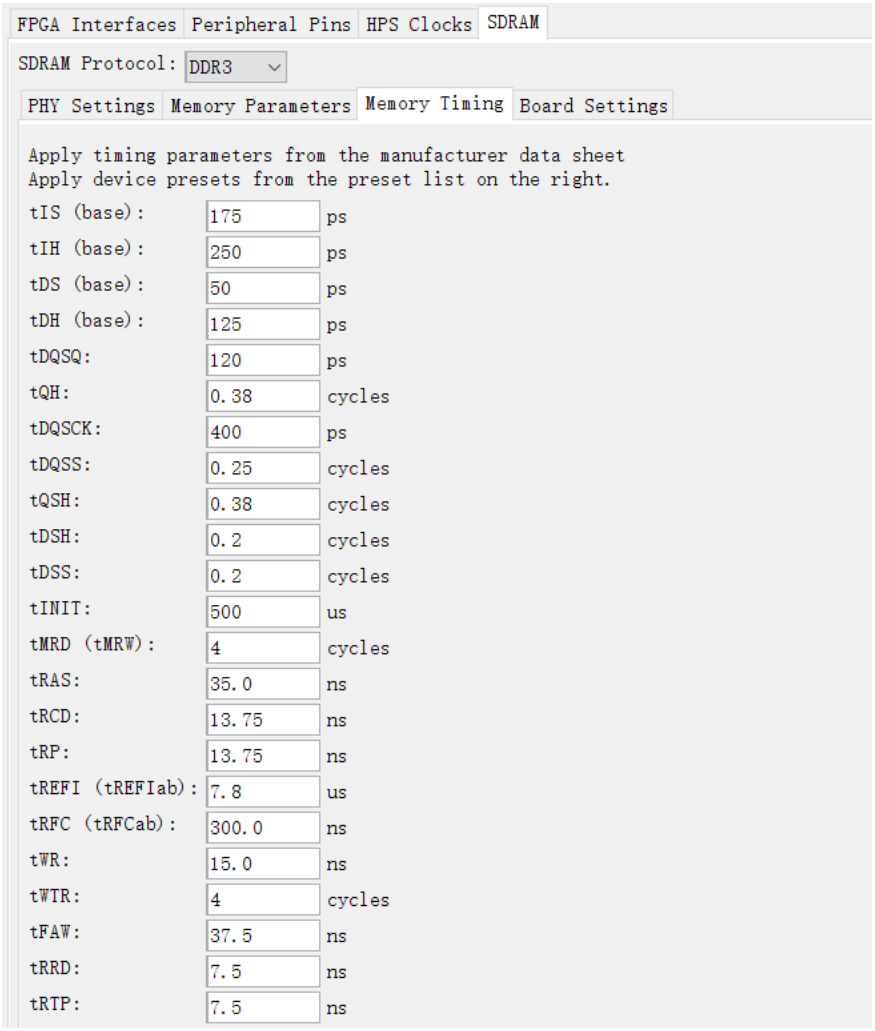


Fig. 37 Set memory timing

j. Add PIO. Search IP: PIO (Parallel I/O), and double-click to enter, first we set an output pin LED, the bit width is set to 8 bits, as shown below.

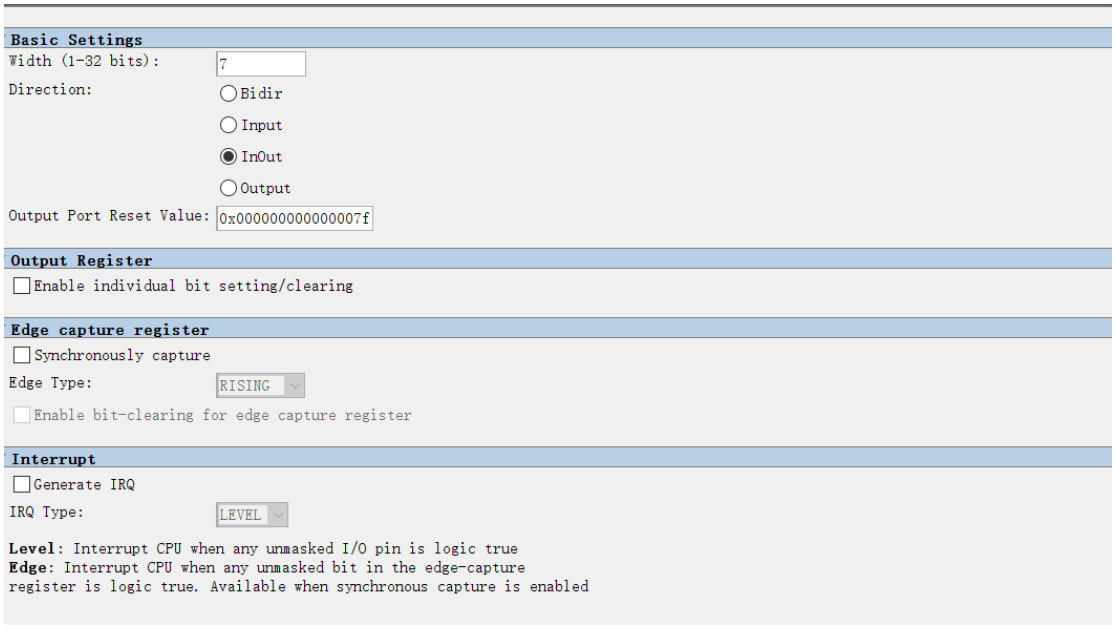


Fig. 38 Set the PIO component

k. Add memory SDRAM

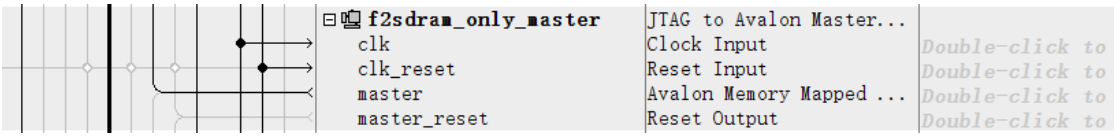


Fig. 39 Set the SDRAM

l. After assigning one of the addresses after assigning base addresses

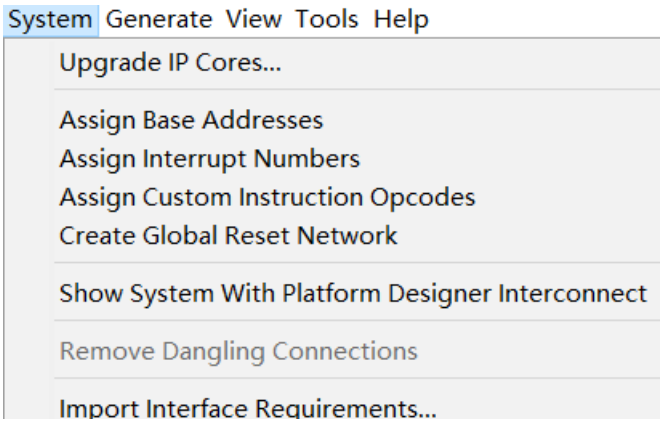


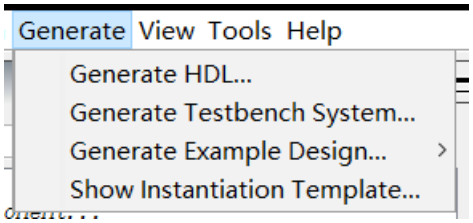
Fig. 40 Assigning the addresses of components

m. The memory map of system peripherals in the FPGA portion of the SoC as viewed by the MPU starts at the lightweight HPS-to-FPGA base address 0xFF20_0000. The MPU can access these peripherals through the Address offset setting in the Qsys.

	r	hps_only_master.master	mm_bridge_0.m0
sysid_qsys.control_slave			0x0000_1000 - 0x0000_1007
jtag_uart.avalon_jtag...			0x0000_2000 - 0x0000_2007
led_pio.s1			0x0000_3000 - 0x0000_300f
dipsw_pio.s1			0x0000_4000 - 0x0000_400f
button_pio.s1			0x0000_5000 - 0x0000_500f
ILC.avalon_slave			0x0003_0000 - 0x0003_00ff
mm_bridge_0.s0	ff		
hps_0.f2h_sdram0_data			
hps_0.f2h_axi_slave		0x0000_0000 - 0xffff_ffff	
sysid_qsys.control_s1...	07		
jtag_uart.avalon_jtag...	07		
led_pio.s1 via mm_bri...	0f		
dipsw_pio.s1 via mm_b...	0f		
button_pio.s1 via mm_...	0f		
ILC.avalon_slave via ...	ff		

Fig. 41 Addresses of system peripherals

n. Show Instantiation Template and assign the relative pins.



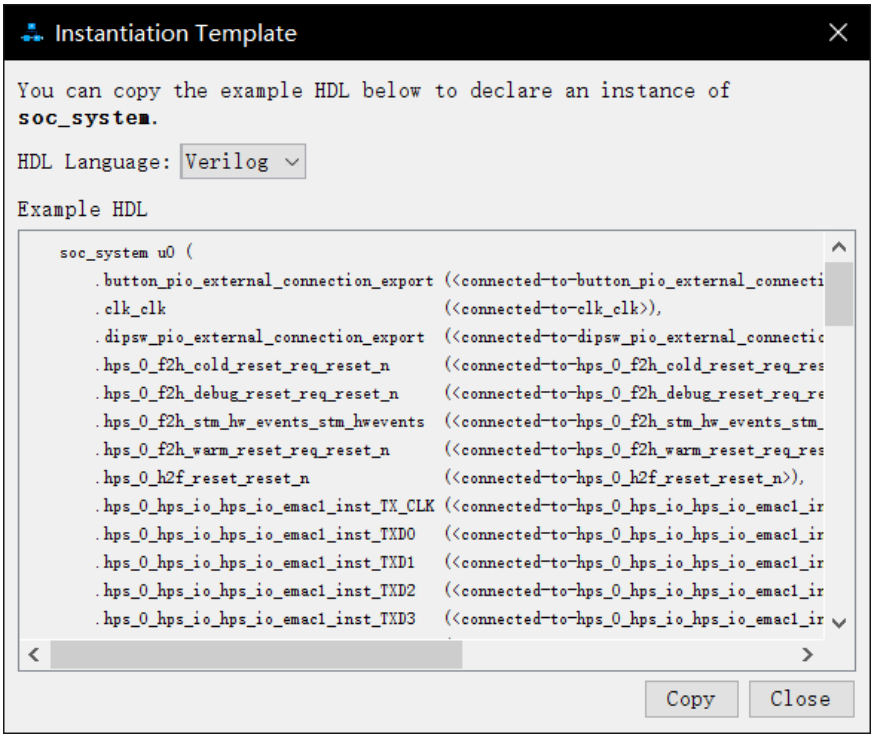
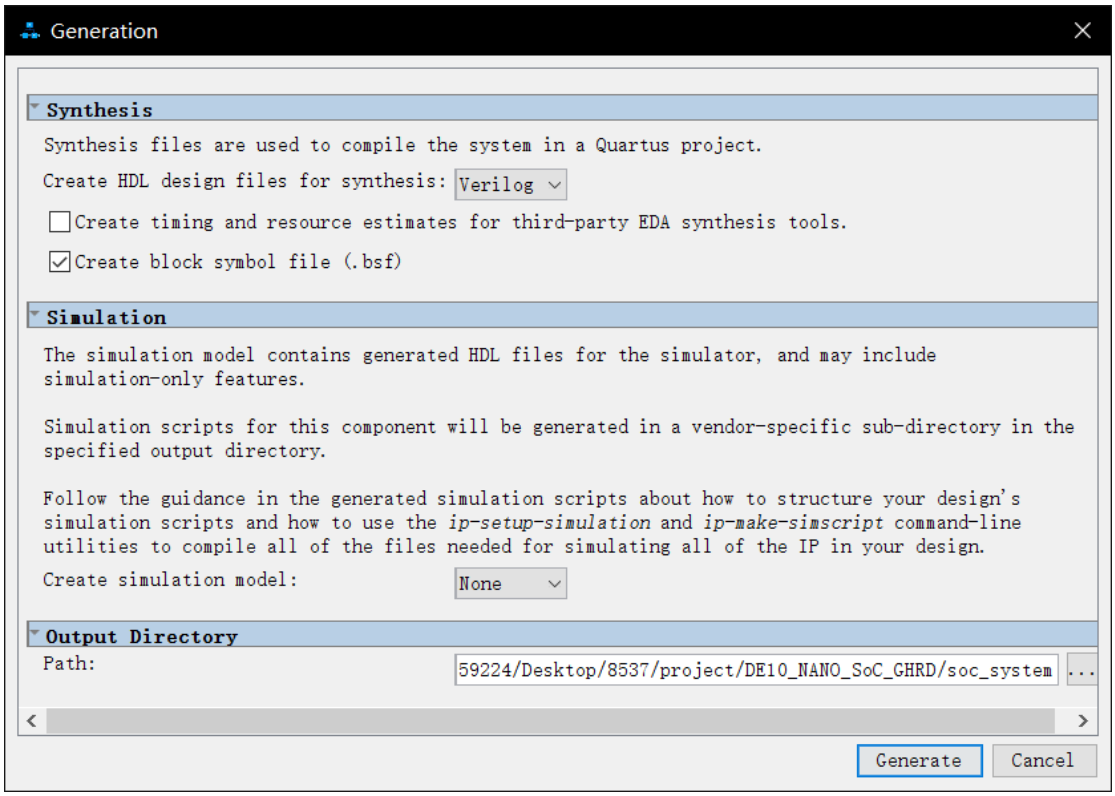


Fig. 42 Building instantiation template

o. Then generate HDL for the top-level design.




```
//FPGA Partion
.pio_external_connection_in_port   (GPIO[7: 1]),   //
.pio_external_connection_out_port  (GPIO[7: 1]),  //
.dipsw_pio_external_connection_export(SW),                          //  dipsw_pio_extern
.button_pio_external_connection_export(fpga_debounced_buttons),
                                                                    // button_pio_extern
.hps_0_h2f_reset_reset_n(hps_fpga_reset_n),                         //              hps
.hps_0_f2h_cold_reset_req_reset_n(~hps_cold_reset),                 //       hps_0_f2h_c
.hps_0_f2h_debug_reset_req_reset_n(~hps_debug_reset),               //       hps_0_f2h_del
.hps_0_f2h_stm_hw_events_stm_hwevents(stm_hw_events),               //         hps_0_f2h_
.hps_0_f2h_warm_reset_req_reset_n(~hps_warm_reset),                 //       hps_0_f2h_w
```

Fig. 43 Generating HDL

p. After assigning relative pins, start compilation.

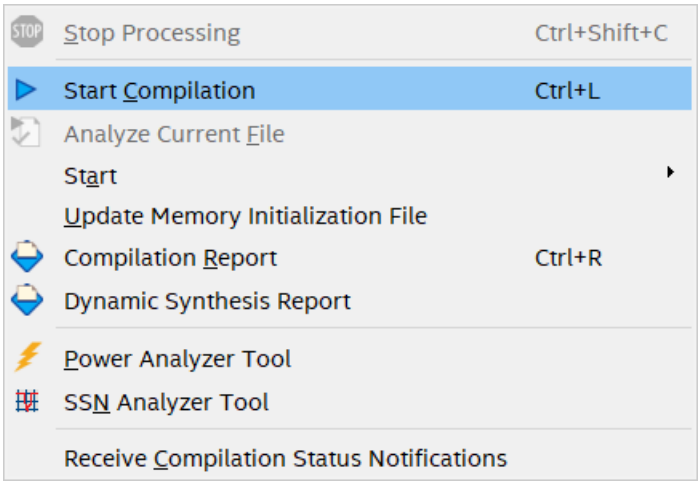


Fig. 44 Start compilation

q. HPS Header File

pio component information is required for ARM C program as the program will attempt to control the component. This section describes how to use a given Linux shell batch file to extract the Qsys HPS information to a header file which will be included in the C program later.

The batch file mentioned above is called as generate_hps_qsys_header.sh. It is in the same folder as DE10_NANO_SoC_GHRD Quartus project. To generate the header file, launch SoC EDS command shell, go to the Quartus project folder, and execute generate_hps_qsys_header.sh by typing ‘./generate_hps_qys_header.sh”. Then, press ENTER key, a header file hps_0.h will be generated. In the header file, the pio base address is represented by a constant PIO_BASE as show in Figure 7-5. The pio width is represented by a constant PIO_DATA_WIDTH. These two constants will be used in the C program demonstration code

```
59224@DESKTOP-FQ3NIV7 ~/Desktop/8537/project/DE10_NANO_SoC_GHRD
$ ./generate_hps_qsys_header.sh
swinfo2header: Creating macro file 'hps_0.h' for module 'hps_0'
```

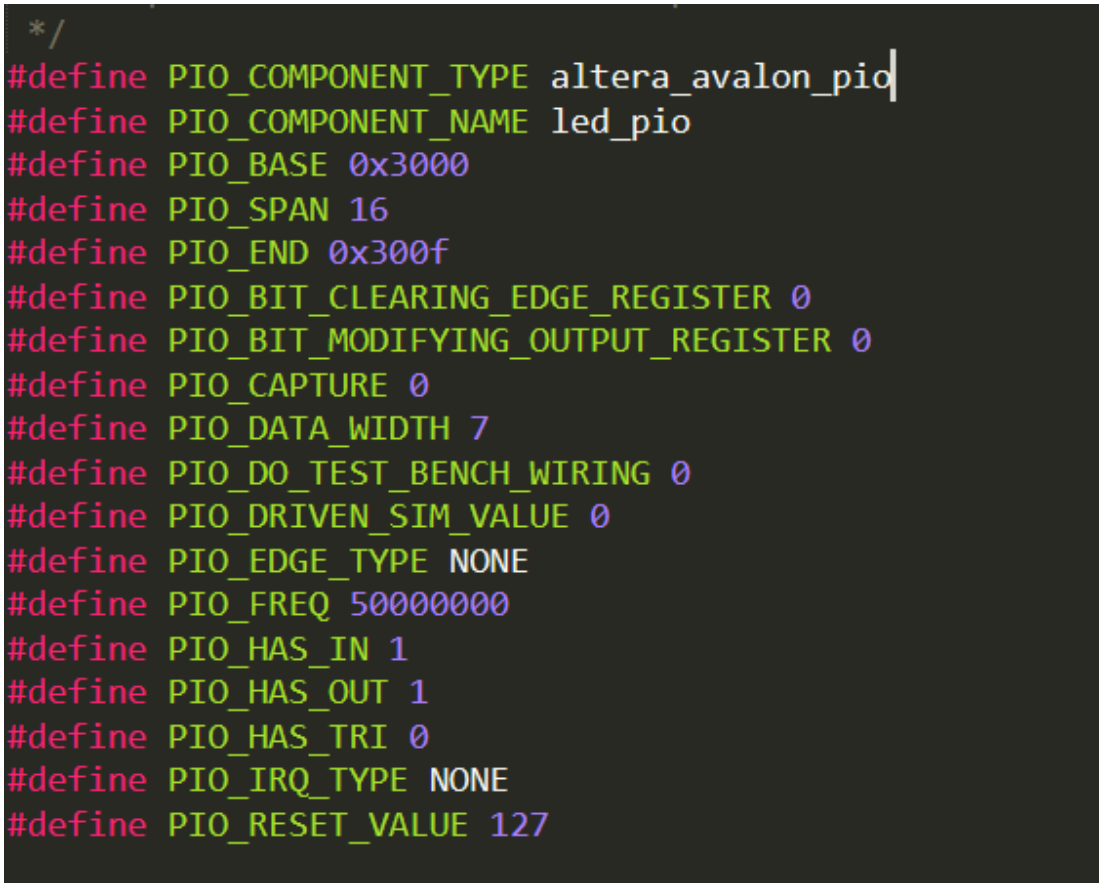


Fig. 45 Information in header file

4.Results

After the setting up and configuration, the client computer and server FPGA board can communicate now. The figure below shows that the input command in client and the corresponding position of servo motor in server FPGA board.

Fig. 46 The input command in client and the corresponding position of servo motor in FPGA board

From the Fig.46, the input command includes the target angle and speed. When the target angle is 0 degree, the sever motor points to the left directly. Then, the target angle in the next command is 180 degree, the servo motor points to the right directly. The angle control is very precise.

When the angle of servo motor rotates from 0 degree to 180-degree, PuTTY is used to monitor the current angle of servo motor. Because the rotation speed is 1, the increasement of current angle is slow. The output of current angle of servo motor in PuTTY has been shown below

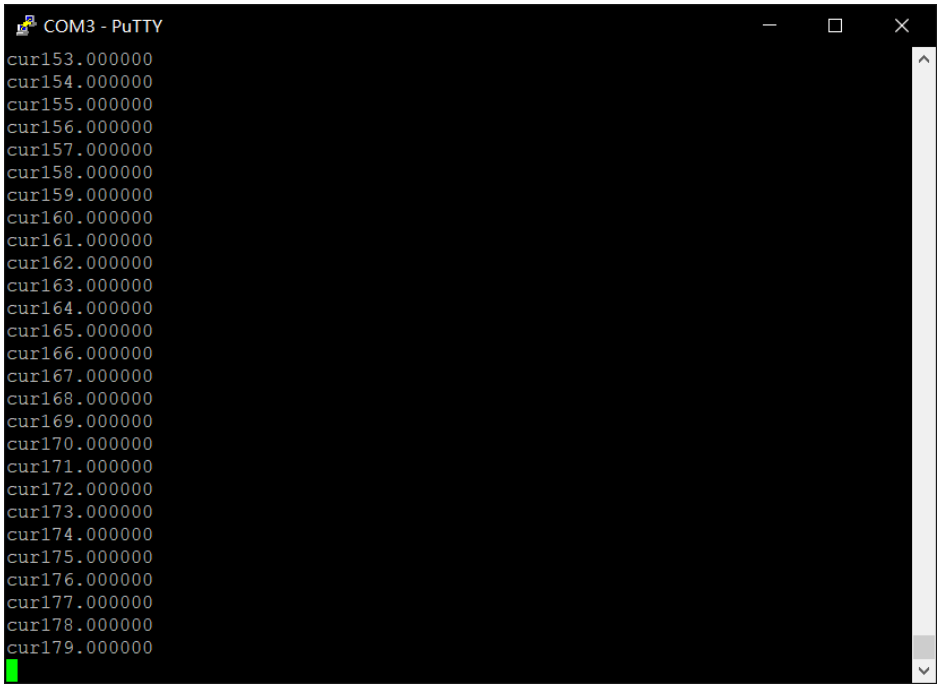


Fig. 47 The output of current angle of servo motor in PuTTY

The input angle can be any integer from 0 to 210, it corresponds the real angle of the servo motor. If the input angle is smaller than 0 or larger than 210, the client shell will output error and hint to rewrite the valid value from 0 to 210. In theory, the input speed can be from 0 to any number. The value is larger, the speed is faster. However, the motor in servo motor has the maximum speed, when the input speed is too large, the servo motor cannot response with the same speed. Thus, the maximum input speed is set to 5. The figure below shows some continuous input commands in client, they all can be executed by servo motor in real time.

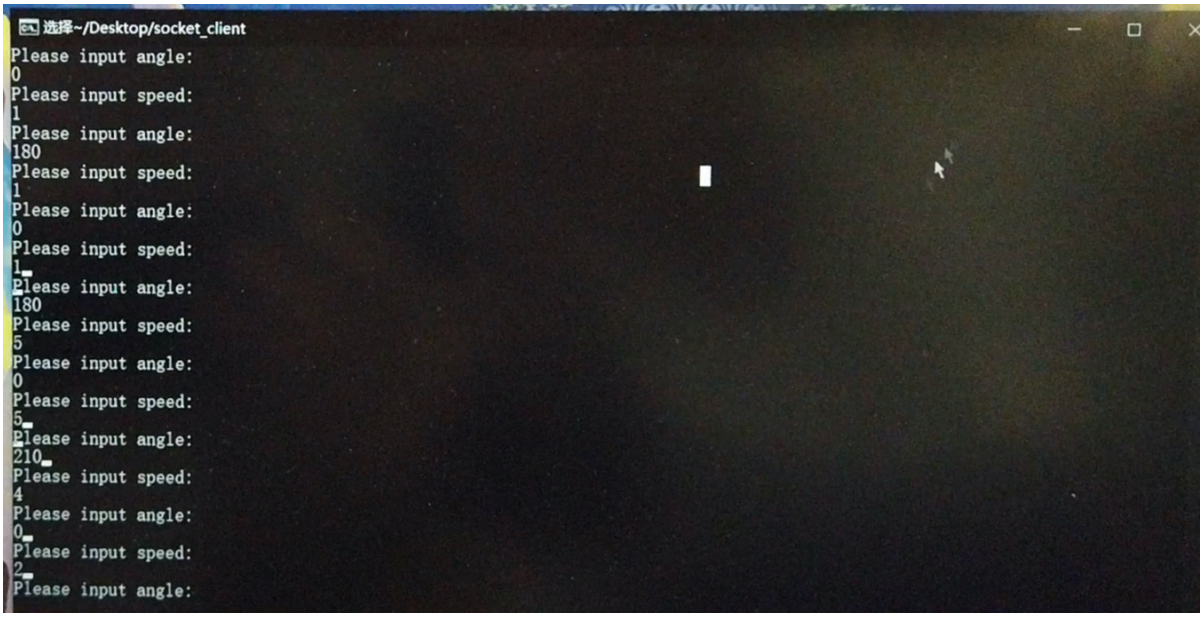


Fig. 48 Some continuous input commands in client

5.Discussion

The Results part shows clearly that the servo motor can be controlled precisely for position and speed in real time. The delay cannot be recognized by human eyes. When comparing TCP and UDP, although UDP is considered faster in theory since that the header of UDP is 8 Bytes while it of TCP is 20 Bytes, the delay of them are both acceptable. Besides, the most of time that is spent on TCP communication is the three way handshake. Once the connection is established, then the transmitting and receiving time of TCP is similar to UDP. For test, our team has tested these two protocols, and both of them are in real time and have almost none delay. Due to the reason that TCP provides more robust connection, the TCP has been applied. For the further work, if there are serval FPGA board connected to the Ethernet, and the client needs to control them at the same time. the UDP will be applied. It is not as stable as TCP, but it can do broadcast for one-to-many communications.

Considering the robustness of the system, the most fragile part of the system is the servo motor. It can be broken when the duty cycle drives it out of the clip position. To avoid this, the checking procedure of the input speed and angle has been implemented on the host computer side. If and only if the input angle is between 0 degree and 180 degree as well as the speed is above 0, the input will be accepted, while in other cases the hint message will be printed on the screen to let the user to retry.

Besides, a potential bug of the program is that the angle can be override if the speed is very huge. This is because that the angle will continuously increment until the current angle is larger than the target. Nevertheless, if the speed is very huge, then the current angle will increment with a huge step which causes it be greater than the target angle. To solve this problem, we set the ceiling and floor as the target angle of the current angle when it increments and decrements, respectively, which means that if the current angle plus the speed is greater than the target angle, the next current angle will be set to the target angle directly.

At the beginning, our team try to use Verilog in FPGA to implement the socket and communicate with the client computer. However, we found that the Ethernet port is in HPS part in DE10-nano board. To connect this port to the Ethernet, we should run Linux in the HPS part. Then the socket in Linux has many advantages, like Linux supplies socket API directly, and the connection in Linux will be more stable. And it is convenience for our team to write the code in Linux for socket and control system.

6.Conclusion

Finally, the project has been completed totally. The servo motor in FPGA board can be controlled to target position and speed via Ethernet Local Area Network (LAN) in real time. The protocol which is used in the Network level in Ethernet is TCP. The client socket is in a Window computer and the servo socket is in a Linux operating system in HPS. After test, the control system is almost real time. The delay cannot be recognized by human eyes. And the target position angle is very precise which is controlled.

The most difficult part in this project is that the continuous data transport. When the socket in HPS has received the data from window computer via Ethernet, the HPS need transport the data to FPGA via AXI bridge. And the Qsys needs to be used to construct the bridge and do the address mapping.

After solving those problems, this project can have wider applications. The controlled device can be more than one servo motor or any other electric devices. They can connect to the FPGA board and use other commands to control them.

7. Future Work

For future work, there are still several parts can be developed.

7.1 More servo motors

In this project, there is one servo motor has been controlled via Ethernet. In future, we can control several servo motors at the same time. It will be more useful for some real situation. Then, we have two approaches.

For the first one. The command will be a queue, the first one value in the queue is used to control the No.1 servo motor, the second value in the queue is used to control the No.2 servo motor, the third value in the queue is used to control the No.3 servo motor… Until each servo motor has a corresponding command value. At each period, the client socket sends the queue command to every servo motor, the different servo motor will just act depending on the corresponding command value. When input the command in client, we just need to write the target value for the servo motor which needs to be controlled. The other variables in the queue will keep the previous value and the corresponding servo motor will not act.

The second approach is that, there is a specific label for each servo motor. when the client sends the command to the server, the command will include the label to definite which servo motor it wants to control.

7.2 Control other devices

On the other hand, the devices will not just be servo motors. There are many other devices need to be controlled. For example, if we control a robot via Ethernet, we need to control the servo motors, DC motors, some lights, and some sensors. Sometime the robot will also feedback some data from sensors. It is a complex communication and should be considered.

8.Reference

[1] FPGA XILINX. (2019) Link: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an- fpga.html

[2] Cloer, L. (2017) FPGA vs Microcontroller – Advantages of Using An FPGA. Doutech. Links: https://duotechservices.com/fpga-vs-microcontroller-advantages-of-using-fpga

[3] DE10-nano2 User Manual. (2018). Links: https://www.manualslib.com/manual/1266461/Terasic-De10-Nano.html

[4] Christensen, K., Reviriego, P., Nordman, B., Bennett, M., Mostowfi, M., & Maestro, J. A. (2010). IEEE 802.3 az: the road to energy efficient ethernet. IEEE Communications Magazine, 48(11), 50-56.

[5] Briscoe, N. (2000). Understanding the OSI 7-layer model. PC Network Advisor, 120(2).

[6] Sublayer, R. (1998). and Media Independent Interface (MII),(Local and Metropolican Area Networks). IEEE Std, 802, 488-524.

[7] Widmer, A. X., & Franaszek, P. A. (1983). A DC-balanced, partitioned-block, 8B/10B transmission code. IBM Journal of research and development, 27(5), 440-451.

[8] Desai, B., Frigo, N. J., Smiljanic, A., Reichmann, K. C., Iannone, P. P., & Roman, R. S. (2001, March). An optical implementation of a packet-based (Ethernet) MAC in a WDM passive optical network overlay. In Optical Fiber Communication Conference (p. WN5). Optical Society of America.

[9] Piscitello, D. M., & Chapin, A. L. (1993). Open systems networking: TCP/IP and OSI (pp. 349-351). Reading, MA: Addison-Wesley.

[10] Forouzan, B. A., & Fegan, S. C. (2006). TCP/IP protocol suite (Vol. 2). McGraw-Hill.

[11] Bansal, S., Shorey, R., & Kherani, A. A. (2004, March). Performance of tcp and udp protocols in multi-hop multi-rate wireless networks. In 2004 IEEE Wireless Communications and Networking Conference (IEEE Cat. No. 04TH8733) (Vol. 1, pp. 231-236). IEEE.

[12] Katz, D., & Saluja, R. (2008). Three-Way Handshake for IS-IS Point-to-Point Adjacencies.

[13] Agrawal, A., Julian, D. J., & Prakash, R. (2012). U.S. Patent No. 8,125,961. Washington, DC: U.S. Patent and Trademark Office.

[14] Gay, W. W. (2000). Linux socket programming: by example. Que Corp.

[15] Dejan, "How Servo Motors Work & How To Control Servos using Arduino," How to Mechatronics, 18 07 2018. Link: https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/

[16] User manual of MG 996R servo motor. Link: https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf

[17] Pulse-width modulation (2019). Wikipedia. Link: https://en.wikipedia.org/wiki/Pulse-width_modulation#cite_note-1

[18] Accessing HPS Devices from the FPGA. (2017). Intel FPGA.

[19] Chu, P. P. (2011). Embedded SOPC design with NIOS II processor and VHDL examples. John Wiley & Sons.

[20] Hawkins, D. W. (2012). Altera JTAG-to-Avalon MM Tutorial.

[21] Altera, C. (2014). Qsys-Altera's System Integration Tool.

[22] Specifications, A. I., & Specifications, A. P. (2013). Qsys Components.

[23] Chen, C. H., Ju, J. C., & Huang, J. (2010, November). A synthesizable AXI protocol checker for SoC integration. In 2010 International SoC Design Conference (pp. 103-106). IEEE.

[24] Gandhani, P., & Patel, C. (2011). Moving from AMBA AHB to AXI Bus in SoC Designs: A Comparative Study. International Journal of Computer Science & Emerging Technologies (IJCSET), 2(4), 476-479.

9.Appedix

Server Program

```
/* A simple server in the internet domain using TCP
   The port number is passed as an argument */

// http://www.linuxhowtos.org/C_C++/socket.htm
// http://www.linuxhowtos.org/data/6/server.c

// function: Create a socket server. Dump the received message from the client and loopack the message to client.
// uage: ./socket_server port_no (porn_no is a interger larger than 2000.)

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_gpio.h"
#include "hps_0.h"

#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

void error(const char *msg)
{
   perror(msg);
   exit(1);
}


int main(int argc, char *argv[])
{
    int sockfd, newsockfd, portno;
    socklen_t clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;
    int angle;
    int speed;
```

```
    void *virtual_base;
          int fd;
          int loop_count;
          int led_direction;
          int led_mask;
          int gpio_mask;
          int *h2p_lw_led_addr;

          if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
          printf( "ERROR: could not open \"/dev/mem\"...\n" );
          return( 1 );
          }

          virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ |
PROT_WRITE ), MAP_SHARED, fd, HW_REGS_BASE );

          if( virtual_base == MAP_FAILED ) {
                    printf( "ERROR: mmap() failed...\n" );
                    close( fd );
                    return( 1 );
          }

          h2p_lw_led_addr=virtual_base + ( ( unsigned long  )( ALT_LWFPGASLVS_OFST +
LED_PIO_BASE ) & ( unsigned long)( HW_REGS_MASK ) );

         if (argc < 2) {
        fprintf(stderr,"ERROR, no port provided\n");
        exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
      error("ERROR opening socket");
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
            sizeof(serv_addr)) < 0)
            error("ERROR on binding");
    listen(sockfd,5);
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd,
              (struct sockaddr *) &cli_addr,
              &clilen);
    if (newsockfd < 0)
        error("ERROR on accept");
    float cur = 0; //initialize current angle
    while(1)
```

```
    {

    //input angle
    n = write(newsockfd,"Please input angle:",19);
    if (n < 0) error("ERROR writing to socket angle");
    bzero(buffer,256);
    n = read(newsockfd,buffer,255);
    if (n < 0) error("ERROR reading from socket");
    angle = atoi(buffer);


    //input speed
    n = write(newsockfd,"Please input speed:",19);
    if (n < 0) error("ERROR writing to socket speed");
    bzero(buffer,256);
    n = read(newsockfd,buffer,255);
    if (n < 0) error("ERROR reading from socket");
    speed = atoi(buffer);

    printf("Target angle: %d Target speed: %d \n", angle, speed);


          float degree_max = 2.5;
          float degree_min = 0.5;
          int min_angle = 0;
          int max_angle = 180;
          float high_dur;

/*
    printf("input speed\n");
    scanf("%d\n",&speed);
    printf("input angle");
    scanf("%d\n",&angle);
    printf("speed: %d  angle:%d\n",speed,angle);
    //speed = 0.2;
    //angle = 175;
*/



          float res;
          if (cur < angle){
              while (cur < angle){
         if ((cur + speed) > angle ){
            cur = angle;
         }
         else{
            cur += speed;
         }
```

```
                        *h2p_lw_led_addr = 0x1f;
                        high_dur = (degree_max-degree_min) / 180 * cur + degree_min;
                        usleep(high_dur * 1000);
                        *h2p_lw_led_addr = 0;
                        res = 20-high_dur;
                        usleep(res * 1000);

                }
                }
         else{
             while (cur > angle){
        if ((cur - speed) < angle ){
           cur = angle;
        }
        else{
           cur -= speed;
        }

                        *h2p_lw_led_addr = 0x1f;
                        high_dur = (degree_max-degree_min) / 180 * cur + degree_min;
                        usleep(high_dur * 1000);
                        *h2p_lw_led_addr = 0;
                        res = 20-high_dur;
                        usleep(res * 1000);

                }
                 }

         }
    close(newsockfd);
    close(sockfd);
    return 0;
}
```

Client Program

```
/* A simple server in the internet domain using TCP
   The port number is passed as an argument */

// http://www.linuxhowtos.org/C_C++/socket.htm
// http://www.linuxhowtos.org/data/6/client.c

// function: Create a socket client. Connect a socket server. Send a message to the server, and dump message received from the server.
// uage: ./socket_client hostname port_no
// you can check the server host name by typing "hostname" in the machine where socket_server is running.

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg)
{
    perror(msg);
    exit(0);
}
//TCP
/*
int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
```

```
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    while(1)
    {
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    if (n < 0)
         error("ERROR writing to socket");
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);
    }
    close(sockfd);
    return 0;
}
*/

//UDP
int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    socklen_t serlen;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0)
         error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
         fprintf(stderr,"ERROR, no such host\n");
         exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
```

```
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    serlen = sizeof(serv_addr);
    while(1)
    {
    //input angle
    bzero(buffer,256);
    n = recvfrom(sockfd,buffer,255,0,(struct sockaddr *) &serv_addr,
                &serlen);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);

    //printf("Please enter the message: ");
    bzero(buffer,256);
    while (1){
        fgets(buffer,255,stdin);
        if (atoi(buffer) > 0 && atoi(buffer) < 180){
            break;
        }
        else{
            printf("The angle must be between 0 and 180, please retry");
        }
    }

    n = sendto(sockfd,buffer,strlen(buffer),0,(struct sockaddr *) &serv_addr,
sizeof(serv_addr));
    if (n < 0)
         error("ERROR writing to socket");

    //input speed
    bzero(buffer,256);
    n = recvfrom(sockfd,buffer,255,0,(struct sockaddr *) &serv_addr,
                &serlen);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);
    bzero(buffer,256);
    while (1){
        fgets(buffer,255,stdin);
        if (atoi(buffer) > 0){
            break;
        }
        else{
            printf("The speed must be larger than 0, please retry");
```

```
        }
    }
    n = sendto(sockfd,buffer,strlen(buffer),0,(struct sockaddr *) &serv_addr,
sizeof(serv_addr));
    if (n < 0)
         error("ERROR writing to socket");

    }
    close(sockfd);
    return 0;
}
```

Hps_0

```
#ifndef _ALTERA_HPS_0_H_
#define _ALTERA_HPS_0_H_

/*
 * This file was automatically generated by the swinfo2header utility.
 *
 * Created from SOPC Builder system 'soc_system' in
 * file './soc_system.sopcinfo'.
 */

/*
 * This file contains macros for module 'hps_0' and devices
 * connected to the following masters:
 *   h2f_axi_master
 *   h2f_lw_axi_master
 *
 * Do not include this header file and another header file created for a
 * different module or master group at the same time.
 * Doing so may result in duplicate macro names.
 * Instead, use the system header file which has macros with unique names.
 */

/*
 * Macros for device 'parallel_port_0', class 'altera_up_avalon_parallel_port'
 * The macros are prefixed with 'PARALLEL_PORT_0_'.
 * The prefix is the slave descriptor.
 */
#define PARALLEL_PORT_0_COMPONENT_TYPE altera_up_avalon_parallel_port
#define PARALLEL_PORT_0_COMPONENT_NAME parallel_port_0
#define PARALLEL_PORT_0_BASE 0x0
#define PARALLEL_PORT_0_SPAN 16
#define PARALLEL_PORT_0_END 0xf

/*
 * Macros for device 'sysid_qsys', class 'altera_avalon_sysid_qsys'
 * The macros are prefixed with 'SYSID_QSYS_'.
 * The prefix is the slave descriptor.
 */
#define SYSID_QSYS_COMPONENT_TYPE altera_avalon_sysid_qsys
#define SYSID_QSYS_COMPONENT_NAME sysid_qsys
#define SYSID_QSYS_BASE 0x1000
#define SYSID_QSYS_SPAN 8
#define SYSID_QSYS_END 0x1007
#define SYSID_QSYS_ID 2899645186
#define SYSID_QSYS_TIMESTAMP 1569211904

/*
 * Macros for device 'jtag_uart', class 'altera_avalon_jtag_uart'
```

```
 * The macros are prefixed with 'JTAG_UART_'.
 * The prefix is the slave descriptor.
 */
#define JTAG_UART_COMPONENT_TYPE altera_avalon_jtag_uart
#define JTAG_UART_COMPONENT_NAME jtag_uart
#define JTAG_UART_BASE 0x2000
#define JTAG_UART_SPAN 8
#define JTAG_UART_END 0x2007
#define JTAG_UART_IRQ 2
#define JTAG_UART_READ_DEPTH 64
#define JTAG_UART_READ_THRESHOLD 8
#define JTAG_UART_WRITE_DEPTH 64
#define JTAG_UART_WRITE_THRESHOLD 8

/*
 * Macros for device 'led_pio', class 'altera_avalon_pio'
 * The macros are prefixed with 'LED_PIO_'.
 * The prefix is the slave descriptor.
 */
#define LED_PIO_COMPONENT_TYPE altera_avalon_pio
#define LED_PIO_COMPONENT_NAME led_pio
#define LED_PIO_BASE 0x3000
#define LED_PIO_SPAN 16
#define LED_PIO_END 0x300f
#define LED_PIO_BIT_CLEARING_EDGE_REGISTER 0
#define LED_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define LED_PIO_CAPTURE 0
#define LED_PIO_DATA_WIDTH 7
#define LED_PIO_DO_TEST_BENCH_WIRING 0
#define LED_PIO_DRIVEN_SIM_VALUE 0
#define LED_PIO_EDGE_TYPE NONE
#define LED_PIO_FREQ 50000000
#define LED_PIO_HAS_IN 0
#define LED_PIO_HAS_OUT 1
#define LED_PIO_HAS_TRI 0
#define LED_PIO_IRQ_TYPE NONE
#define LED_PIO_RESET_VALUE 127

/*
 * Macros for device 'dipsw_pio', class 'altera_avalon_pio'
 * The macros are prefixed with 'DIPSW_PIO_'.
 * The prefix is the slave descriptor.
 */
#define DIPSW_PIO_COMPONENT_TYPE altera_avalon_pio
#define DIPSW_PIO_COMPONENT_NAME dipsw_pio
#define DIPSW_PIO_BASE 0x4000
#define DIPSW_PIO_SPAN 16
#define DIPSW_PIO_END 0x400f
#define DIPSW_PIO_IRQ 0
#define DIPSW_PIO_BIT_CLEARING_EDGE_REGISTER 1
```

```
#define DIPSW_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define DIPSW_PIO_CAPTURE 1
#define DIPSW_PIO_DATA_WIDTH 4
#define DIPSW_PIO_DO_TEST_BENCH_WIRING 0
#define DIPSW_PIO_DRIVEN_SIM_VALUE 0
#define DIPSW_PIO_EDGE_TYPE ANY
#define DIPSW_PIO_FREQ 50000000
#define DIPSW_PIO_HAS_IN 1
#define DIPSW_PIO_HAS_OUT 0
#define DIPSW_PIO_HAS_TRI 0
#define DIPSW_PIO_IRQ_TYPE EDGE
#define DIPSW_PIO_RESET_VALUE 0

/*
 * Macros for device 'button_pio', class 'altera_avalon_pio'
 * The macros are prefixed with 'BUTTON_PIO_'.
 * The prefix is the slave descriptor.
 */
#define BUTTON_PIO_COMPONENT_TYPE altera_avalon_pio
#define BUTTON_PIO_COMPONENT_NAME button_pio
#define BUTTON_PIO_BASE 0x5000
#define BUTTON_PIO_SPAN 16
#define BUTTON_PIO_END 0x500f
#define BUTTON_PIO_IRQ 1
#define BUTTON_PIO_BIT_CLEARING_EDGE_REGISTER 1
#define BUTTON_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define BUTTON_PIO_CAPTURE 1
#define BUTTON_PIO_DATA_WIDTH 2
#define BUTTON_PIO_DO_TEST_BENCH_WIRING 0
#define BUTTON_PIO_DRIVEN_SIM_VALUE 0
#define BUTTON_PIO_EDGE_TYPE FALLING
#define BUTTON_PIO_FREQ 50000000
#define BUTTON_PIO_HAS_IN 1
#define BUTTON_PIO_HAS_OUT 0
#define BUTTON_PIO_HAS_TRI 0
#define BUTTON_PIO_IRQ_TYPE EDGE
#define BUTTON_PIO_RESET_VALUE 0
/*
 * Macros for device 'ILC', class 'interrupt_latency_counter'
 * The macros are prefixed with 'ILC_'.
 * The prefix is the slave descriptor.
 */
#define ILC_COMPONENT_TYPE interrupt_latency_counter
#define ILC_COMPONENT_NAME ILC
#define ILC_BASE 0x30000
#define ILC_SPAN 256
#define ILC_END 0x300ff


#endif /* _ALTERA_HPS_0_H_ */
```

Modified GHRD

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module DE10_NANO_SoC_GHRD(

    ///////////// CLOCK /////////
    input              FPGA_CLK1_50,
    input              FPGA_CLK2_50,
    input              FPGA_CLK3_50,

    ///////////// HDMI /////////
    inout              HDMI_I2C_SCL,
    inout              HDMI_I2C_SDA,
    inout              HDMI_I2S,
    inout              HDMI_LRCLK,
    inout              HDMI_MCLK,
    inout              HDMI_SCLK,
    output             HDMI_TX_CLK,
    output   [23: 0]   HDMI_TX_D,
    output             HDMI_TX_DE,
    output             HDMI_TX_HS,
    input              HDMI_TX_INT,
    output             HDMI_TX_VS,

    ///////////// HPS /////////
    inout              HPS_CONV_USB_N,
    output   [14: 0]   HPS_DDR3_ADDR,
    output   [ 2: 0]   HPS_DDR3_BA,
    output             HPS_DDR3_CAS_N,
    output             HPS_DDR3_CK_N,
    output             HPS_DDR3_CK_P,
    output             HPS_DDR3_CKE,
    output             HPS_DDR3_CS_N,
    output   [ 3: 0]   HPS_DDR3_DM,
    inout    [31: 0]   HPS_DDR3_DQ,
    inout    [ 3: 0]   HPS_DDR3_DQS_N,
    inout    [ 3: 0]   HPS_DDR3_DQS_P,
    output             HPS_DDR3_ODT,
    output             HPS_DDR3_RAS_N,
    output             HPS_DDR3_RESET_N,
    input              HPS_DDR3_RZQ,
    output             HPS_DDR3_WE_N,
    output             HPS_ENET_GTX_CLK,
    inout              HPS_ENET_INT_N,
    output             HPS_ENET_MDC,
    inout              HPS_ENET_MDIO,
    input              HPS_ENET_RX_CLK,
```

```
    input    [ 3: 0]    HPS_ENET_RX_DATA,
    input               HPS_ENET_RX_DV,
    output   [ 3: 0]    HPS_ENET_TX_DATA,
    output              HPS_ENET_TX_EN,
    inout               HPS_GSENSOR_INT,
    inout               HPS_I2C0_SCLK,
    inout               HPS_I2C0_SDAT,
    inout               HPS_I2C1_SCLK,
    inout               HPS_I2C1_SDAT,
    inout               HPS_KEY,
    inout               HPS_LED,
    inout               HPS_LTC_GPIO,
    output              HPS_SD_CLK,
    inout               HPS_SD_CMD,
    inout    [ 3: 0]    HPS_SD_DATA,
    output              HPS_SPIM_CLK,
    input               HPS_SPIM_MISO,
    output              HPS_SPIM_MOSI,
    inout               HPS_SPIM_SS,
    input               HPS_UART_RX,
    output              HPS_UART_TX,
    input               HPS_USB_CLKOUT,
    inout    [ 7: 0]    HPS_USB_DATA,
    input               HPS_USB_DIR,
    input               HPS_USB_NXT,
    output              HPS_USB_STP,

    ////////// KEY //////////
    input    [ 1: 0]    KEY,

    ////////// LED //////////
    inout    [ 7: 0]    LED,

    ////////// SW //////////
    input    [ 3: 0]    SW
);



//=======================================================
//  REG/WIRE declarations
//=======================================================
wire hps_fpga_reset_n;
wire    [1: 0]    fpga_debounced_buttons;
wire    [6: 0]    fpga_led_internal;
wire    [2: 0]    hps_reset_req;
wire              hps_cold_reset;
wire              hps_warm_reset;
wire              hps_debug_reset;
wire    [27: 0]   stm_hw_events;
```

```
wire            fpga_clk_50;
// connection of internal logics
// assign LED[7: 1] = fpga_led_internal;
assign fpga_clk_50 = FPGA_CLK1_50;
assign stm_hw_events = {{15{1'b0}}, SW, fpga_led_internal, fpga_debounced_buttons};



//=======================================================
//  Structural coding
//=======================================================
soc_system u0(
        //Clock&Reset
        .clk_clk(FPGA_CLK1_50),                         //                clk.clk
        .reset_reset_n(hps_fpga_reset_n),               //              reset.reset_n
        //HPS ddr3
        .memory_mem_a(HPS_DDR3_ADDR),                   //
memory.mem_a
        .memory_mem_ba(HPS_DDR3_BA),
//                      .mem_ba
        .memory_mem_ck(HPS_DDR3_CK_P),
//                      .mem_ck
        .memory_mem_ck_n(HPS_DDR3_CK_N),
//                      .mem_ck_n
        .memory_mem_cke(HPS_DDR3_CKE),
//                      .mem_cke
        .memory_mem_cs_n(HPS_DDR3_CS_N),
//                      .mem_cs_n
        .memory_mem_ras_n(HPS_DDR3_RAS_N),
//                      .mem_ras_n
        .memory_mem_cas_n(HPS_DDR3_CAS_N),
//                      .mem_cas_n
        .memory_mem_we_n(HPS_DDR3_WE_N),
//                      .mem_we_n
        .memory_mem_reset_n(HPS_DDR3_RESET_N),
//                      .mem_reset_n
        .memory_mem_dq(HPS_DDR3_DQ),
//                      .mem_dq
        .memory_mem_dqs(HPS_DDR3_DQS_P),
//                      .mem_dqs
        .memory_mem_dqs_n(HPS_DDR3_DQS_N),
//                      .mem_dqs_n
        .memory_mem_odt(HPS_DDR3_ODT),
//                      .mem_odt
        .memory_mem_dm(HPS_DDR3_DM),
//                      .mem_dm
        .memory_oct_rzqin(HPS_DDR3_RZQ),
//                      .oct_rzqin
        //HPS ethernet
```

```
        .hps_0_hps_io_hps_io_emac1_inst_TX_CLK(HPS_ENET_GTX_CLK),    //
hps_0_hps_io.hps_io_emac1_inst_TX_CLK
        .hps_0_hps_io_hps_io_emac1_inst_TXD0(HPS_ENET_TX_DATA[0]),
//                      .hps_io_emac1_inst_TXD0
        .hps_0_hps_io_hps_io_emac1_inst_TXD1(HPS_ENET_TX_DATA[1]),
//                      .hps_io_emac1_inst_TXD1
        .hps_0_hps_io_hps_io_emac1_inst_TXD2(HPS_ENET_TX_DATA[2]),
//                      .hps_io_emac1_inst_TXD2
        .hps_0_hps_io_hps_io_emac1_inst_TXD3(HPS_ENET_TX_DATA[3]),
//                      .hps_io_emac1_inst_TXD3
        .hps_0_hps_io_hps_io_emac1_inst_RXD0(HPS_ENET_RX_DATA[0]),
//                      .hps_io_emac1_inst_RXD0
        .hps_0_hps_io_hps_io_emac1_inst_MDIO(HPS_ENET_MDIO),
//                      .hps_io_emac1_inst_MDIO
        .hps_0_hps_io_hps_io_emac1_inst_MDC(HPS_ENET_MDC),
//                      .hps_io_emac1_inst_MDC
        .hps_0_hps_io_hps_io_emac1_inst_RX_CTL(HPS_ENET_RX_DV),
//                      .hps_io_emac1_inst_RX_CTL
        .hps_0_hps_io_hps_io_emac1_inst_TX_CTL(HPS_ENET_TX_EN),
//                      .hps_io_emac1_inst_TX_CTL
        .hps_0_hps_io_hps_io_emac1_inst_RX_CLK(HPS_ENET_RX_CLK),
//                      .hps_io_emac1_inst_RX_CLK
        .hps_0_hps_io_hps_io_emac1_inst_RXD1(HPS_ENET_RX_DATA[1]),
//                      .hps_io_emac1_inst_RXD1
        .hps_0_hps_io_hps_io_emac1_inst_RXD2(HPS_ENET_RX_DATA[2]),
//                      .hps_io_emac1_inst_RXD2
        .hps_0_hps_io_hps_io_emac1_inst_RXD3(HPS_ENET_RX_DATA[3]),
//                      .hps_io_emac1_inst_RXD3
        //HPS SD card
        .hps_0_hps_io_hps_io_sdio_inst_CMD(HPS_SD_CMD),
//                      .hps_io_sdio_inst_CMD
        .hps_0_hps_io_hps_io_sdio_inst_D0(HPS_SD_DATA[0]),
//                      .hps_io_sdio_inst_D0
        .hps_0_hps_io_hps_io_sdio_inst_D1(HPS_SD_DATA[1]),
//                      .hps_io_sdio_inst_D1
        .hps_0_hps_io_hps_io_sdio_inst_CLK(HPS_SD_CLK),
//                      .hps_io_sdio_inst_CLK
        .hps_0_hps_io_hps_io_sdio_inst_D2(HPS_SD_DATA[2]),
//                      .hps_io_sdio_inst_D2
        .hps_0_hps_io_hps_io_sdio_inst_D3(HPS_SD_DATA[3]),
//                      .hps_io_sdio_inst_D3
        //HPS USB
        .hps_0_hps_io_hps_io_usb1_inst_D0(HPS_USB_DATA[0]),
//                      .hps_io_usb1_inst_D0
        .hps_0_hps_io_hps_io_usb1_inst_D1(HPS_USB_DATA[1]),
//                      .hps_io_usb1_inst_D1
        .hps_0_hps_io_hps_io_usb1_inst_D2(HPS_USB_DATA[2]),
//                      .hps_io_usb1_inst_D2
        .hps_0_hps_io_hps_io_usb1_inst_D3(HPS_USB_DATA[3]),
//                      .hps_io_usb1_inst_D3
```

```
        .hps_0_hps_io_hps_io_usb1_inst_D4(HPS_USB_DATA[4]),
//                  .hps_io_usb1_inst_D4
        .hps_0_hps_io_hps_io_usb1_inst_D5(HPS_USB_DATA[5]),
//                  .hps_io_usb1_inst_D5
        .hps_0_hps_io_hps_io_usb1_inst_D6(HPS_USB_DATA[6]),
//                  .hps_io_usb1_inst_D6
        .hps_0_hps_io_hps_io_usb1_inst_D7(HPS_USB_DATA[7]),
//                  .hps_io_usb1_inst_D7
        .hps_0_hps_io_hps_io_usb1_inst_CLK(HPS_USB_CLKOUT),
//                  .hps_io_usb1_inst_CLK
        .hps_0_hps_io_hps_io_usb1_inst_STP(HPS_USB_STP),
//                  .hps_io_usb1_inst_STP
        .hps_0_hps_io_hps_io_usb1_inst_DIR(HPS_USB_DIR),
//                  .hps_io_usb1_inst_DIR
        .hps_0_hps_io_hps_io_usb1_inst_NXT(HPS_USB_NXT),
//                  .hps_io_usb1_inst_NXT
        //HPS SPI
        .hps_0_hps_io_hps_io_spim1_inst_CLK(HPS_SPIM_CLK),
//                  .hps_io_spim1_inst_CLK
        .hps_0_hps_io_hps_io_spim1_inst_MOSI(HPS_SPIM_MOSI),
//                  .hps_io_spim1_inst_MOSI
        .hps_0_hps_io_hps_io_spim1_inst_MISO(HPS_SPIM_MISO),
//                  .hps_io_spim1_inst_MISO
        .hps_0_hps_io_hps_io_spim1_inst_SS0(HPS_SPIM_SS),
//                  .hps_io_spim1_inst_SS0
        //HPS UART
        .hps_0_hps_io_hps_io_uart0_inst_RX(HPS_UART_RX),
//                  .hps_io_uart0_inst_RX
        .hps_0_hps_io_hps_io_uart0_inst_TX(HPS_UART_TX),
//                  .hps_io_uart0_inst_TX
        //HPS I2C1
        .hps_0_hps_io_hps_io_i2c0_inst_SDA(HPS_I2C0_SDAT),
//                  .hps_io_i2c0_inst_SDA
        .hps_0_hps_io_hps_io_i2c0_inst_SCL(HPS_I2C0_SCLK),
//                  .hps_io_i2c0_inst_SCL
        //HPS I2C2
        .hps_0_hps_io_hps_io_i2c1_inst_SDA(HPS_I2C1_SDAT),
//                  .hps_io_i2c1_inst_SDA
        .hps_0_hps_io_hps_io_i2c1_inst_SCL(HPS_I2C1_SCLK),
//                  .hps_io_i2c1_inst_SCL
        //GPIO
        .hps_0_hps_io_hps_io_gpio_inst_GPIO09(HPS_CONV_USB_N),
//                  .hps_io_gpio_inst_GPIO09
        .hps_0_hps_io_hps_io_gpio_inst_GPIO35(HPS_ENET_INT_N),
//                  .hps_io_gpio_inst_GPIO35
        .hps_0_hps_io_hps_io_gpio_inst_GPIO40(HPS_LTC_GPIO),
//                  .hps_io_gpio_inst_GPIO40
        .hps_0_hps_io_hps_io_gpio_inst_GPIO53(HPS_LED),
//                  .hps_io_gpio_inst_GPIO53
```

```
            .hps_0_hps_io_hps_io_gpio_inst_GPIO54(HPS_KEY),
//                      .hps_io_gpio_inst_GPIO54
            .hps_0_hps_io_hps_io_gpio_inst_GPIO61(HPS_GSENSOR_INT),
//                      .hps_io_gpio_inst_GPIO61
            //FPGA Partion
            .led_pio_external_connection_in_port   (LED[7: 1]),   //
led_pio_external_connection.in_port
                                        .led_pio_external_connection_out_port  (LED[7: 1]),  //
            .dipsw_pio_external_connection_export(SW),                //
dipsw_pio_external_connection.export
            .button_pio_external_connection_export(fpga_debounced_buttons),
                                                    // button_pio_external_connection.export
            .hps_0_h2f_reset_reset_n(hps_fpga_reset_n),               //
hps_0_h2f_reset.reset_n
            .hps_0_f2h_cold_reset_req_reset_n(~hps_cold_reset),         //
hps_0_f2h_cold_reset_req.reset_n
            .hps_0_f2h_debug_reset_req_reset_n(~hps_debug_reset),       //
hps_0_f2h_debug_reset_req.reset_n
            .hps_0_f2h_stm_hw_events_stm_hwevents(stm_hw_events),       //
hps_0_f2h_stm_hw_events.stm_hwevents
            .hps_0_f2h_warm_reset_req_reset_n(~hps_warm_reset),         //
hps_0_f2h_warm_reset_req.reset_n

        );

// Debounce logic to clean out glitches within 1ms
debounce debounce_inst(
            .clk(fpga_clk_50),
            .reset_n(hps_fpga_reset_n),
            .data_in(KEY),
            .data_out(fpga_debounced_buttons)
        );
defparam debounce_inst.WIDTH = 2;
defparam debounce_inst.POLARITY = "LOW";
defparam debounce_inst.TIMEOUT = 50000;              // at 50Mhz this is a debounce time of
1ms
defparam debounce_inst.TIMEOUT_WIDTH = 16;           // ceil(log2(TIMEOUT))

// Source/Probe megawizard instance
hps_reset hps_reset_inst(
            .source_clk(fpga_clk_50),
            .source(hps_reset_req)
        );

altera_edge_detector pulse_cold_reset(
                    .clk(fpga_clk_50),
                    .rst_n(hps_fpga_reset_n),
                    .signal_in(hps_reset_req[0]),
                    .pulse_out(hps_cold_reset)
                );
```

```
defparam pulse_cold_reset.PULSE_EXT = 6;
defparam pulse_cold_reset.EDGE_TYPE = 1;
defparam pulse_cold_reset.IGNORE_RST_WHILE_BUSY = 1;

altera_edge_detector pulse_warm_reset(
                    .clk(fpga_clk_50),
                    .rst_n(hps_fpga_reset_n),
                    .signal_in(hps_reset_req[1]),
                    .pulse_out(hps_warm_reset)
                );
defparam pulse_warm_reset.PULSE_EXT = 2;
defparam pulse_warm_reset.EDGE_TYPE = 1;
defparam pulse_warm_reset.IGNORE_RST_WHILE_BUSY = 1;

altera_edge_detector pulse_debug_reset(
                    .clk(fpga_clk_50),
                    .rst_n(hps_fpga_reset_n),
                    .signal_in(hps_reset_req[2]),
                    .pulse_out(hps_debug_reset)
                );
defparam pulse_debug_reset.PULSE_EXT = 32;
defparam pulse_debug_reset.EDGE_TYPE = 1;
defparam pulse_debug_reset.IGNORE_RST_WHILE_BUSY = 1;

reg [25: 0] counter;
reg led_level;
always @(posedge fpga_clk_50 or negedge hps_fpga_reset_n) begin
   if (~hps_fpga_reset_n) begin
      counter <= 0;
      led_level <= 0;
   end

   else if (counter == 24999999) begin
      counter <= 0;
      led_level <= ~led_level;
   end
   else
      counter <= counter + 1'b1;
end

assign LED[0] = led_level;


endmodule
```