Xilinx Version

XC3S1000 Development Board

Experiment Guider

Ver1.0

Eleckits

www.eleckits.com

Catalogue

Experiment 1 LED Control Experiment

1.1.1 Experiment purpose

1. Control 8 LED'S displaying status by 4 buttons on development board.

1.1.2 Experiment theory

Write one button on the development board to control the LED displaying.
Detailed displaying program is as following:

State	SW2	SW3	SW4	SW5	LED0	LED1	LED2	LED3	LED4	LED5	LED6	LED7
S1	1	1	1	0	0	0	0	0	0	0	0	1
S2	1	1	0	1	0	0	0	0	0	0	1	0
S3	1	0	1	1	0	0	0	0	0	1	0	0
S4	0	1	1	1	0	0	0	0	1	0	0	0
S5	1	1	0	0	0	0	0	1	0	0	0	0
S6	1	0	0	1	0	0	1	0	0	0	0	0
S7	0	0	1	1	0	1	0	0	0	0	0	0
S8	0	1	1	0	1	0	0	0	0	0	0	0
Default				0	0	0	0	0	0	0	0	0

1.1.3 Experiment content

Write button controlling LED program and achieve them one the development board.

1.1.4 Experiment steps

Programmings download on Xilinx ISE 11. Debug on the development board. Detailed ISE software operation is referenced to《ISE Software Using Description》.

1.1.5 Experiment result

You can see the expected LED turns to shining on the development board.

Experiment 2 Divider Experiment

1.2.1 Experiment purpose

1. Design one divider which is specified frequency coefficient.

1.2.2 Experiment theory

Frequency divided means the processed clock frequency is lower than input clock frequency. On the contrary, the output clock frequency is higher input one, we call it frequency multiplication. Frequency divided is achieved by user programming. The frequency multiplication is achieved by PLL or DLL which FPGA owns itself. Theoretically speaking, there is no limitation of frequency divided if the clock cycle is less then endless. However, frequency multiplication is upon the FPGA's features and some constraints in actual design. They will decide the frequency after multiplied.

Divider is the base of digital circuit design. Not only in image processing but also in audio signal processing, you need to use it a lot.

Experiment development board provides one 50MHz clock frequency. In actual using, we seldom to use the precise clock frequency 50MHz. We use the frequency lower than 50MHz. For example, in the video processing, most of the chips' working frequency like SAA7121 is in 20-30MHz, the working frequency of the clock line SCL of IIC controller is in 20-30MHz and etc. We have to provide clock frequency dividing to the development boards to make them suit the different application programs as the main clock frequency.

There are many ways of frequency dividing. Or you can generate the clock that has different duty cycles and frequencies. The frequency factor normally used is integer power of 2 and frequency deviding's duty cycle is 50%.

1.2.3 Experiment content

Do the 2 integer power frequency dividing of the input clock 50MHz. Powers are：18，19，20，21，22，23，24，25. Then, use the divided clocks to control 8 LEDs on the board shinning. Watch the dividing effect.

1.2.4 Experiment result

See LED on the board shinning by different frequencies. At the high frequency, we think it keep shining as the visual sensitivity is not enough.

Experiment 3 State device application experiment

1.3.1 Experiment purpose

1. Learn how to use ISE.
2. Know the structure of state machine.
3. Learn how to use state machine to write relatively complex programs.

1.3.2 Experiment theory

State machine design is the core part of HDL designing. Almost all designs use its thought. State machine is the cycle mechanism composited by serial of states. This structure can make programmer to use HDL language better. Meanwhile, the state machine with certain style can improve the readability and the debugging of the programs.

There are many elements of state machine design. Following are some important ones:

- State machine's coding. Binary、gray-code coding uses least triggers and more combination logics. But the one-hot coding is opposite. Because CPLD providing more combination logics and FPGA providing more triggers, CPLD uses gray-code，and normally FPGA uses one-hot coding。On the other hand, gray-code and binary is more effective to small design and one-hot suits large state machines more.
- About FSM coding. FSM has two modes: Miller and Moore. Elements are input (including reset), status (including current state operation), state transfer condition and state output condition. There are many ways and skills of FSM designing. Generally speaking, there are two types. One is write state transfer, operation and judging to one module(process、block）. The other is write state transfer in one module, state operation and judging in another one (in Verilog codes, equals to use two "always" blocks).

The second way is better. Following are reasons:

First, FSM is the same as others. You'd better uses timing synchronization way to design. No repetition of advantages here. After state machine achieved, state transfer is achieved by register which is the part of the time synchronization. The judging of state transfer condition is achieved by the judging of combination logic. Why the second way is more reasonable than the first one, is the second coding put the synchronization timing and combination logic to different program blocks(process，block）to achieve. The

advantages of this are not only for better the reading, understanding, maintaining, but also better for the codes optimizing, suitable timing constraint condition adding, designing achieved by placement and routing device.

● Initial status and default status

One complete state device (nice robustness) should have initialization state and default state. When the chip is powered or reset, the state device should reset all judgments conditions automatically and enter initialization state. One thing need you attention. Most of FPGA has GSR (Global Set/Reset）signal. When FPGA is powered, GSR signal higher, it will reset/locate all registers, RAM units and etc. It is the logic configuration in FPGA and not yet be effective. So it cannot guarantee entering the initialization state correctly. Therefore, use GSR to enter FPGA initialization state usually will cause some problems. The normal way is use the asynchronous reset signal, and sometimes synchronous reset. However, please attention to the synchronous reset's logic design. Another way to solve this problem is set default initialized state codes zero. In this way, when GST reset, the state device will enter initialization state automatically. On the other hand, the state device should have one default (default) state. When can meet transfer conditions or state changed suddenly, it can protect the logic from "bad cycle". It is the important requirement to the state device's robustness. The state device has to have the feature "self-recover". To coding is to case, and please pay high attention to sentence if-else. You have to complete condition judgments sentences. In VDL, when use CASE sentences, you have to use "When Others "to set up default state. When use sentence "IF...THEN...ELSE", you have to specify default state in "ELSE". In Verilog, when use "case" sentences, you have to use "default" to set up default state. Notes of using "if ...else" are similar.

Here introduce another skill: most of the synthesizers support Verilog coding state device's complete state feature—"full case". This feature is used to specify the state that integrates the state device into complete state. For example: following are the command formats which Synplicity's synthesis tools（Synplify/Synplify Pro,Amplify，etc）support:

```
case (current_state) // synthesis full_case
2'b00 : next_state <= 2'b01;
2'b01 : next_state <= 2'b11;
2'b11 : next_state <= 2'b00;
//these two sections of codes are equal.
case (current_state)
2'b00 : next_state <= 2'b01;
2'b01 : next_state <= 2'b11;
2'b11 : next_state <= 2'b00;
default : next_state <= 2bx;
```

● You can use parameter to definite the state device. We do not suggest

use define macro to definite. When define Marco is programming, it will replace the Marco definite in whole design automatically. However, parameter only defines the module internal specifications. They will not be confused with other state devices out of the module.

● When program the state device, you'd better write state transfer and works in each state separately in two or more always sub-blocks. It makes reading easier and is better for the adjusting. Following are details （suggest use three-step FAS description way）：

```
always @ ( posedge clk or negedge rst_n )
if( !rst_n )
state <= 2'b00 ;
else
state <= next_state ;
always @ ( posedge clk or negedge rst_n )
if( !rst_n )
next_state <= 2'b00 ;
else
case ( state )
2'b00：begin if(en) next_state<=2'b01; else next_state<=state; end
2'b01：begin if(en) next_state<=2'b10; else next_state<=state; end
2'b10：begin if(en) next_state<=2'b11; else next_state<=state; end
2'b11：begin if(en) next_state<=2'b00; else next_state<=state; end
default：state<=2'b00;
endcase
always @ ( posedge clk or negedge rst_n )
if( !rst_n )
dout<=4'b0000;
else
case ( state )
2'b00：dout<=4'b0001;
2'b01：dout<=4'b0011;
2'b10：dout<=4'b0111;
2'b11：dout<=4'b1111;
default：dout<=4'b0000;
endcase
```

The state device upper uses three-step-description way. One always block is in charge of sending next_state value to state. One always block is in charge of judging trigger and generating next_state. The third one includes the descriptions of works to be finished by state device in each state step. There are many advantages of this way: simple structure, better for timing constraints, no combinational logic output, and better for controlling synthesis, high reliabilities and maintains of the codes.

ISE provides users another special input way: state device input. This way is more complex in inputting and the using range is very limited.

1.3.3 Experiment content

Design one state device which makes 8 LED on the development board shinning in cycle.

1.3.4 Experiment result

See 8 LED on the development board turned on in cycle.

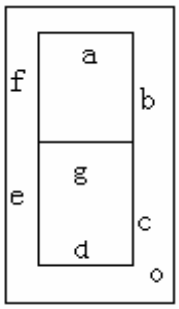
Experiment 4 Digital tube control experiment

1.4.1 Experiment purpose

1. Learn digital tube working principles.
2. Achieve control of digital tube's display by programming.

1.4.2 Experiment theory

Following is the digital tube appearance drawing:



Digital tube displayer is the display device which is often used in digital system experiment. Usually it displays decimal or hex numbers. Therefore we have to decoding all binary numbers used in the experiment. Change them to decimal or hex numbers. There are two kinds of digital tube displayer: common cathode (CC) and common anode (CA). Development board uses CA connection and high level is valid. Input signals are D0,D1,D2,D3, corresponding 8 segments outputting are a,b,c,d,e,f,g,Dp. Their relations are as following:

D0	D1	D2	D3	a	b	c	d	e	f	g	Dp
0	0	0	0	1	1	1	1	1	1	0	0

0	0	0	1	0	1	1	0	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1	0
0	0	1	1	1	1	1	1	0	0	1	0
0	1	0	0	0	1	1	0	0	1	1	0
0	1	0	1	1	0	1	1	0	1	1	0
0	1	1	0	1	0	1	1	1	1	1	0
0	1	1	1	1	1	1	0	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	0	1	1	0
1	0	1	0	1	1	1	0	1	1	1	0
1	0	1	1	0	0	1	1	1	1	1	0
1	1	0	0	1	0	0	1	1	1	0	0
1	1	0	1	0	1	1	1	1	0	1	0
1	1	1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	0	0	0	1	1	1	0

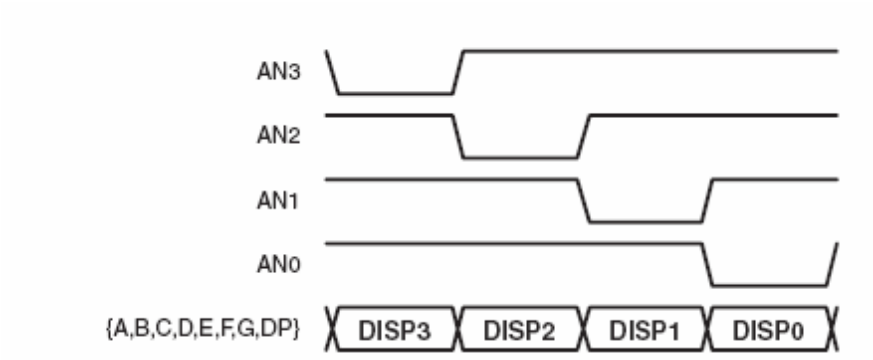
In the experiment, you have to attention to that each segment of LED has to be corresponding to the program on the monitor.

There are 4 digital tubes on the development board which reuse 8 data lines. It is easy to achieve if 4 tubes display the same value. If you want to display 1234, you may need to change the data line's content. You have to find a way to make 4 tubes display 4 contents separately. In many situations, to save I/O pins and internal logic source, we often use dynamic scanning to display. Dynamic scanning uses hours theory and man's persistence of version effect. For example, one 4 bytes dynamic scanning monitor's displaying cycle can be divided into 4 periods:

Period 1------→period 2-------→period 3-------→period4

Each cycle only strobe one byte data. In cycle 1, displays the first data. The second cycle displays the second one… After scanned 4 periods, recycle by order. If the scan speed is fast enough, it will make people feel 4 digital tubes are displaying at the same time.

4 bytes scan digital monitor has 4 groups of BCD code (4 bytes) input lines, 8 pieces of 8 period decoding output lines and 4 strobing lines. In scanning, choose one group data from 4 groups of BCD data. Decode them by BCD shortness of breath decoder and then output. At the same time, 3/8 decoder generates strobe signal. In this moment, the monitor is changed to the digital codes to be output. Then, choose the next group of data, decoding and output. Bit strobe is down by one bit correspondingly. Strobe the next digital code and output it.



1.4.3 Experiment content

1. Make four digital tubes display the same value, from 1 to f.
2. Make four digital tubes display different values, output 1234.

1.4.4 Experiment result

See the requested output result on the development board.

Experiment 5 Counter experiment

1.5.1 Experiment purpose

1. Handle the counters basic concept and implementation.
2. Write one counter program.
3. Display the process of counting by digital tube.
4. Change counting frequency. Watch for counting result.

1.5.2 Experiment theory

Like the divider, the counter is also one of the basic design ways in electrical designing. We can generate many feature modules based on the counter：

Divider：actually, the divider is the clock level whose output is controlled by a counter. When the counter is full, turn the output clock, or make the output clock level equal to the certain bit level of the counter. In this way, we can generate one clock after divided.

Frequency counter： the frequency counter is the normal measuring instrument. It measures the signal frequency by count the signal pulse in unit time. When frequency counter start to work, it will generate count permit

signal that is gate signal. The width of the gate signal is unit time, e.g. 1s or 10ms. Count the signal be tested in the valid time of gate signal, and then, convert it to signal frequency. When the measurement is finished, you have to lock save the counting value or leave some time to show the measurement value. Before next measurement, clear the counter.

Macro generator: Macro generator can generate one or more specified width Macros according to the requirements. There are many way to achieve. Most of them are that generate one high level and start counting. When the counter is full, lower the level. In this way, you can generate different pulse width Macros by change the counting value.

There are many kinds of counters. Either plus counting or minus counting is ok. When the counter is full, both clear up and keep the full state are ok. The actual design has to according to the demands to do the programming.

1.5.3 Experiment content

This experiment needs to design one counter and display the counting process on the digital tubes.-

As the experiment required, the process can be divided into three main parts: frequency dividing, counting and displaying. As the clock frequency provide by development board is 50MHz which cannot be recognized by human eyes, four digital tubes cannot display the huge number generated by such high frequency. Therefore, we have to divide the 50MHZ first which can guarantee the circle of counter's each number is around 1 second.

The counter part is set up by several registers. The raising of each clock will cause the number addition 1 in the register. The counter's reset value is 0000. When the counting reaches 9999, the counter returns to 0000 and restart the counting.

Digital tube controlling sees experiment 4.

1.5.4 Experiment result

See the counting process of the counter on the development board.

Experiment6 Button debounce experiment

1.6.1 Experiment purpose

1. Be familiar with the development environment of ISE11.1;
2. Be familiar with the usage of development board;
3. Learn the operation principle of the button and the way of anti-shake;
4. Program and set an anti-shake circuit.

1.6.2 Experiment theory

If we intend to use the four *SW* buttons to do the counting input, we should firstly know how many times the buttons have been pressed. In this case, we can't just detect if the button is pressed or not by using the rising edge of the input clock as before. Supposed that the clock frequency is 10Hz after frequency division, and we keep the button being pressed for one second, if we simply detect it by the clock rising edge, the program will tell us that the button has been pressed for ten times.

This circumstance also exists in our commonly used keyboard. We need part of the circuit to prevent the above circumstance occur.

Therefore, in order to prevent shaking, we should detect the falling edge and rising edge of the button, rather than detect if the button is pushed. For example, when pressing the button, we should check the falling edge of FPGA pin connected to the button, and the rising edge when releasing it. In this case, we can count input according to the times of pressing and releasing, regardless of how long the button is pressed.

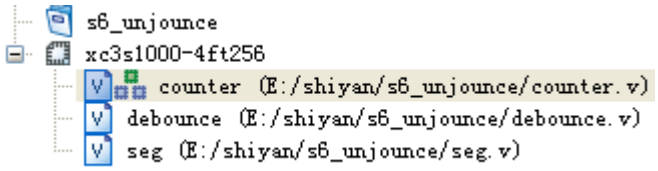
1.6.3 Experiment content

This experiment is to design a debounce circuit which is used to check the button's input. Set a counter. The initial value is zero. Use debounce checking circuit to check the button SW2's input. Each time we get bottom's press or release, the counter's value is added 1. Show the counter's value at the digital tubes.

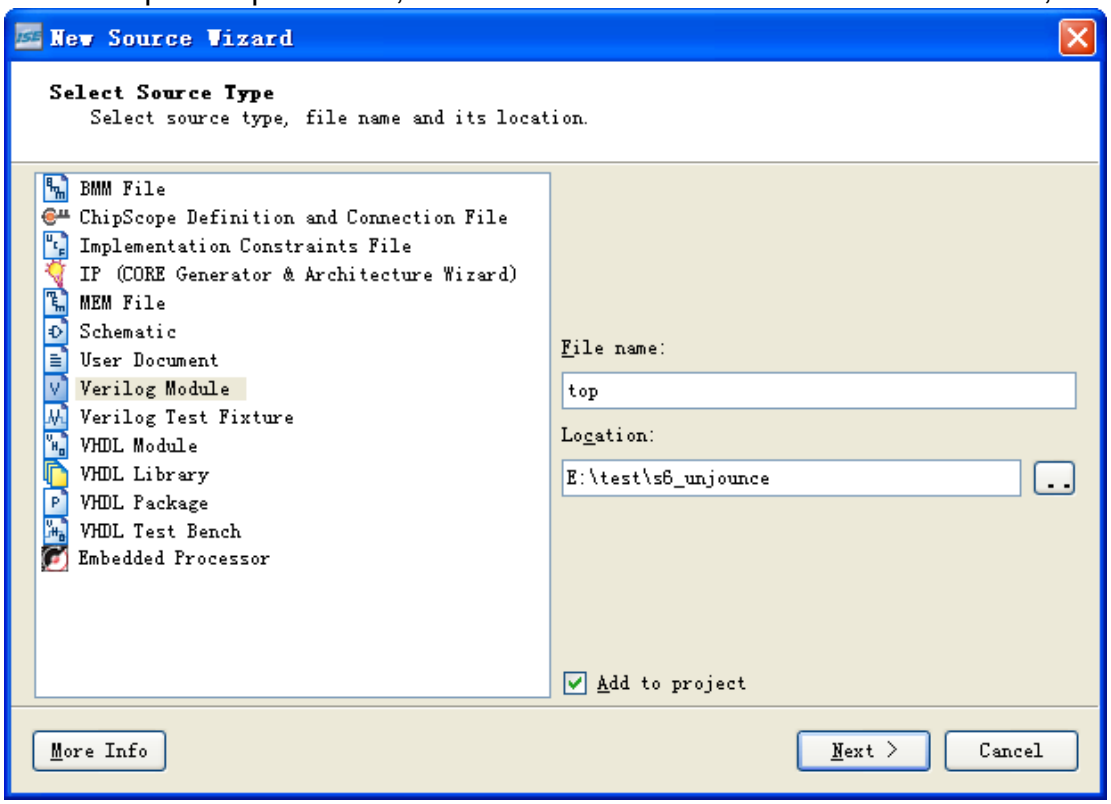
This experiment mainly is set up by three parts: button anti-shake, counting and digital tube controlling.

1.6.4 Experiment steps

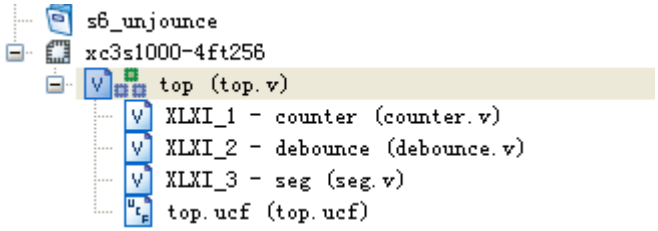
1. Set a new project;
2. Generate the source programs of anti-shaking button, counting, and digital tube controlling separately;



3. Input a top-level file, and call the three modules mentioned above;



See the changes in the folder after saving:



4. Integrate, layout, and route;
5. Download and debug.

1.6.5 Experiment result

Control the digital tube's displaying by SW2. The displaying value is added 1

with every press.

Experiment 7 Buzzer control experiment

1.7.1 Experiment purpose

1. Learn buzzer's structure and working theory.
2. Learn to control buzzer to send different frequencies' voices.

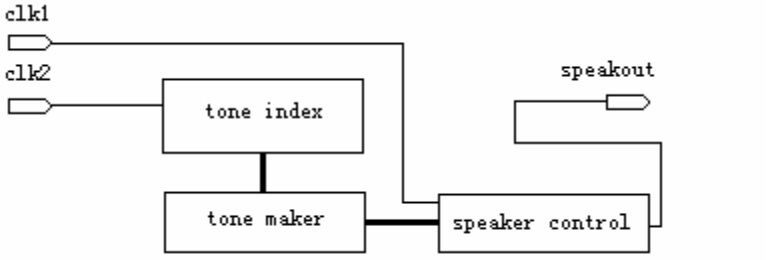
1.7.2 Experiment theory

Compared to control playing music by micro-processer (CPU or MCU), the logic of playing music by pure hardware is more complex. If you do not use powerful EDA tool or hardware description language, only use traditional number logic, you will find it is really very hard to achieve even the simplest circuit.

First, this experiment is used to find the different sounds of buzzers' at different frequencies on development board. See whether it is the same as the following table. Then, do the programming. The buzzer sends the sounds do, re, mi, fa,so, la in turn when clicks the development board.

	do	re	mi	fa	so	la
frequency /Hz	262	294	330	349	392	440
circle /us	3816	3401	3030	2865	2551	2273

Second, this experiment needs the program to control the buzzer on the development board which used the VerilogHDL language. As we know, the sounds frequency value of each note composing the music and the lasting time are two basic elements to guarantee the music's lasting displaying. The details are as following:



You can get the note's frequency by the part speaker control as upper

picture shows. Speaker control is kind of frequency divider controller. The clk1 input the higher clock frequency (as 12MHz, 25MHz and etc). After dividing, there will be speaking out output. It will be connected to the buzzer directly.

The lasting time of note is upon the different music spread and the lasting beats. In upper picture, tone index is checkout table of notes. Input clk is the lower clock (8Hz or 10Hz and etc). The checkout table searches the note to be played by the “plus 1” order and send them to the module tone maker. The tone maker here is the 8-bit binary counter (the highest value is 138). The frequency is at 4Hz. In this way, the stay time of counting one number is 0.25S which is equal to the four-four beat quarter note lasting time when the lasting time of whole note is set 1 second.

Through the upper description, we can use the hardware to achieve different notes’ frequency and lasting time. In this way, “Chinese Romeo and Juliet” can be played consistently.

1.7.3 Experiment content

1. Use buzzer to make different notes sounds on the development board.
2. Make a program to play the “Chinese Romeo and Juliet”.

1.7.4 Experiment steps

Program and download on Xilinx ISE. Debug on the development board.

1.7.5 Experiment result

Hear the “Chinese Romeo and Juliet”.

Experiment8 LCD display control experiment

2.1.1 Experiment purpose

1、Learn the control theory of char-LCD.
2、Handle the basic ideas and methods of driver designing by FPGA.

2.1.2 Experiment theory

1、LCD 1602 brief introduction：

There are two types of LCD screen: dot matrix and LCD type. Experiment here uses LCD screen. It is character type can show 2 lines 16 characters. LCD module uses 14-pin standard interface:

Pin 1：VSS is ground power.

Pin 2：VDD connects to 5V + power;

Pin 3：V0 is LCD contrast adjusting terminal. The contrast is weakest when connect to “+” and is highest when connected to ground power. Over high contrast will cause “ghost shadow”. During the use, you can adjust it by one 10K potential regulator.

Pin 4：RS is register selection. Data register select when it is high level, and command register select when it is low level.

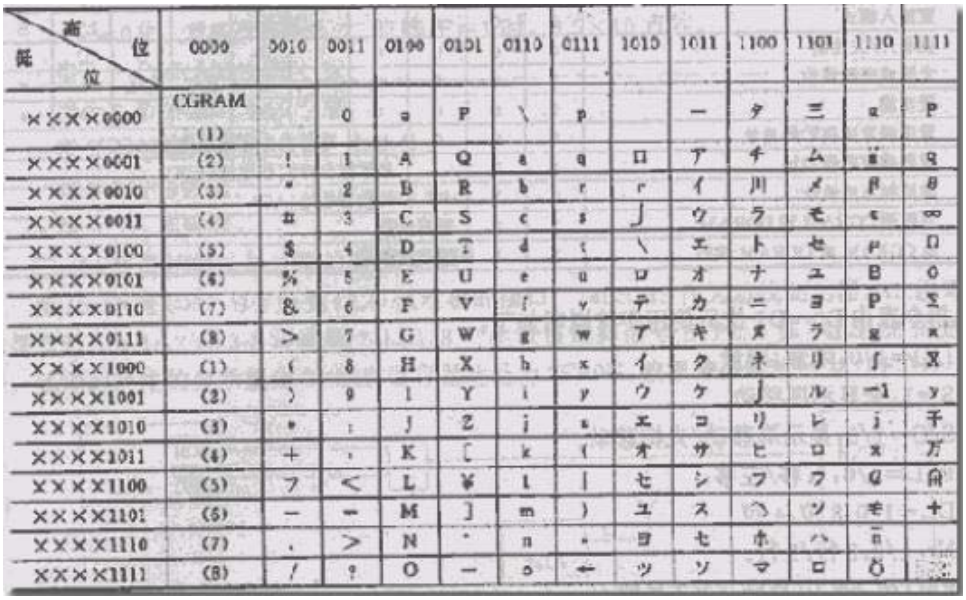
Pin 5：RW is read/write signal. Read during the high level while write during the low level. When both RS and RW are low level, you can write command or show the address. When RS is low level and RW is high lever, you can read busy signal. When RS is high level and RW is low level, you can write the data.

Pin 6：E is enable terminal. When E jumps to low level from high level, LCD module can do the commands.

Pin 7~14：D0～D7 are 8-bit two-way data lines.

The character generating memory (CGROM) internal of the 1602 LCD module has already stored 160 different dot-matrix character graphics, like table 1 shows. These characters are: Arabic numbers, capital letters and low case letters, commonly used symbols, Japanese Kana and etc. Each symbol has one fixed code. For example: the capital letter “A” has the code 0100_0001B（41H）. During the displaying, the module shows the dot-matrix graphic in the address in which way we can see the letter. In programming, you only need to input the related character’s address, and the LCD will output the corresponding character.

Following is the table of relationships between each character and CGROM.



高位 / 低位	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
××××0000	CGRAM (1)		0	@	P	`	p		ー	タ	三	α	P
××××0001	(2)	!	1	A	Q	a	q	口	ア	チ	ム	ä	q
××××0010	(3)	"	2	B	R	b	r	r	イ	川	メ	β	θ
××××0011	(4)	#	3	C	S	c	s	」	ウ	ラ	モ	ε	∞
××××0100	(5)	$	4	D	T	d	t	ヽ	エ	ト	セ	μ	Ω
××××0101	(6)	%	5	E	U	e	u	ロ	オ	ナ	ユ	B	ü
××××0110	(7)	&	6	F	V	f	v	テ	カ	ニ	ヨ	P	Σ
××××0111	(8)	>	7	G	W	g	w	ア	キ	メ	ラ	g	π
××××1000	(1)	(	8	H	X	h	x	イ	ク	ネ	リ	∫	又
××××1001	(2)	)	9	I	Y	i	y	ウ	ケ	」	ル	-1	y
××××1010	(3)	*	:	J	Z	j	z	エ	コ	リ	レ	j	千
××××1011	(4)	+	;	K	[	k	{	オ	サ	ヒ	ロ	x	万
××××1100	(5)	7	<	L	¥	l	|	セ	シ	フ	フ	¢	円
××××1101	(6)	-	=	M	]	m	}	ユ	ス	ハ	ソ	÷	+
××××1110	(7)	.	>	N	^	n	→	ヨ	セ	ホ	ハ	ñ	
××××1111	(8)	/	?	O	_	o	←	ツ	ソ	マ	ロ	ö	

Table 1 CGRAM character and address chart

2、LCD driver designing requirements:

LCD driver's design has to clear the LCD operation command, as following:

表 指令表

	指令	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清显示	0	0	0	0	0	0	0	0	0	1
2	光标返回	0	0	0	0	0	0	0	0	1	*
3	置输入模式	0	0	0	0	0	0	0	1	I/D	S
4	显示开/关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位	0	0	0	0	0	1	S/C	R/L	*	*
6	置功能	0	0	0	0	1	DL	N	F	*	*
7	置字符发生存贮器地址	0	0	0	1	字符发生存贮器地址(AGG)					
8	置数据存贮器地址	0	0	1	显示数据存贮器地址(ADD)						
9	读忙标志或地址	0	1	BF	计数器地址(AC)						
10	写数到 CGRAM 或 DDRAM	1	0	要写的数							
11	从 CGRAM 或 DDRAM 读数	1	1	读出的数据							

Table 2 LCD command table

All of its read/write operation, screens and cursor operation are finished by command programming. (Note: 1 is high level and 0 is low level).

Command 1：clear displaying. Command code is 01H and the cursor is reset to position 00H.

Command 2：cursor reset. Cursor returns to address 00H.

Command 3：cursor and displaying mode set I/D.Cursor moving direction is high level right moving and low level left moving. S: all characters on the screen are moved left or right. High level means valid and low level means invalid.

Command 4：displaying switch controlling. D：control overall display's on and off. High level means display of on and low level means display of off. C：control cursor's on and off. High level means cursor exists and low level means no cursor. B：control

whether the cursor flashing or not. High level means flashing and low level means no flashing.

Command 5：cursor or displacement shows S/C. Moving displays character during high level while cursor in low level.

R/L：character or cursor moving direction. High level is right moving and low level is left moving.

Command 6：feature setting command. DL：in high level is 4-bit general line，and in low level is 8-bit general line. N：single line displaying in low level and double lines displaying in high level. F: in low level shows the Dort-matrix 5x7，while in high level shows 5x10.

Command 7：character generator RAM address set

Command 8：DDRAM address set

Command 9：read busy signal and cursor address. BF：is busy signal. High level means busy. At this time, the module cannot receive command or data. If it is low level, means free.

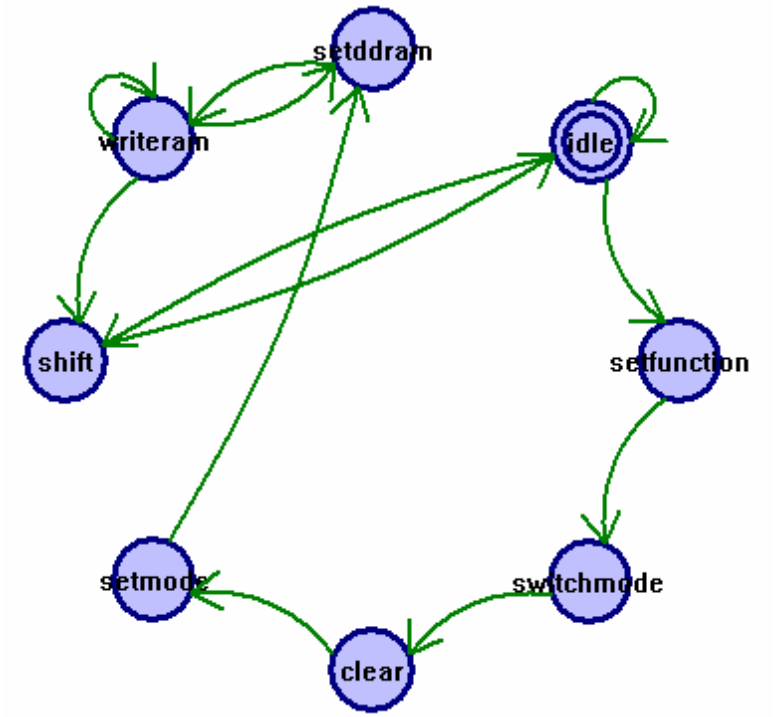
Command 10：write data

Command 11：read data

3、FPGA driver circuit design:

The displaying features to be achieved here are as following: use 5*10 Dort-matrix,; double lines displaying; the first line shows “Welcome to SOLID!”; the second line shows “SOLID!”. As one line can only show 16 characters, the screen has to be left-moving displayed.

There are mainly two modules in this designed driver program: oneischar_ram whose main feature is to output the addresses in CGRM (character generator register memory) of the related characters’ according to the input addresses. In LCD controlling displaying, user only needs to provide related character’s address to display it. In char_ram, firstly, you need to set all characters’ related addresses (according to upper table), then define new address of characters to be used to select output. Another module is LCD’s driver module lcd. This module is used in driver lcd normal working. LDC is a slow displaying device. Therefore, the clock must meet the requirements. Here the clock circle got by frequency division of 50MHz is around 100us (about 10HZ) to meet slow displaying requirements. LCD driver is achieved by one status device. State picture is as following:



LCD driver circuit status picture

Detailed processing is as following:

Powered and reset, the system enter IDLE statue.

First enter SETFUNCTION status. Do command 6 (do command here means write relate control character to data terminal, and set RS and R/W). Set bit numbers of general line and the kind of Dort-matrix to display. After one clock circle (around 100us), enter SWITCHMODE state. Do command 4. set the switch of overall display, switch of cursor and whether the cursor is shinning or not. When the setting is finished, enter CLEAR status and do command 1. Clear the screen. Then, enter SETMODE status and do command 3. Set whether the characters and cursor is moving and moving direction. After it, enter SETDDRAM status and do command 8. Set the initial address of DDRAM. Here set the first line displaying internal address: 1000_0000. The highest bit 1 is reserved bit and following 7 bits are initial address. After it, enter WRITERAM status. Write the address of the character to be displayed into DDRAM. Here the first line displayed is “Welcome to SOLID”（one line is only can show 16 characters）When the displaying is finished, re-enter the SETDDRAM status. Set the initial address of the displaying in second line: 1100_0000. In second line, shows: “SOLID!” And then, enter SHIFT status. Do command 5. Set the character left moving. In moving process, the first line shows “Welcome to SOLID!”completely. Then, keep cycling in IDLE and SHIFT. Keep the characters in left-moving displaying status.

2.1.3 Experiment content

This experiment will read data form ROM and display them through LCD.

2.1.4 Experiment steps

1．Design general feature module. Input verilog design file which can achieve LED displaying character.
2．Complication debugging passed.
3．Download program to experiment board. Debug successfully.

2.1.5 Experiment result

See the first line in LED shows：“Welcome to SOLID”，第二行显示“SOLID!” Then, left-moving display “Welcome to SOLID!”in cycle.

Experiment9 VGA display control experiment

2.2.1 Experiment purpose

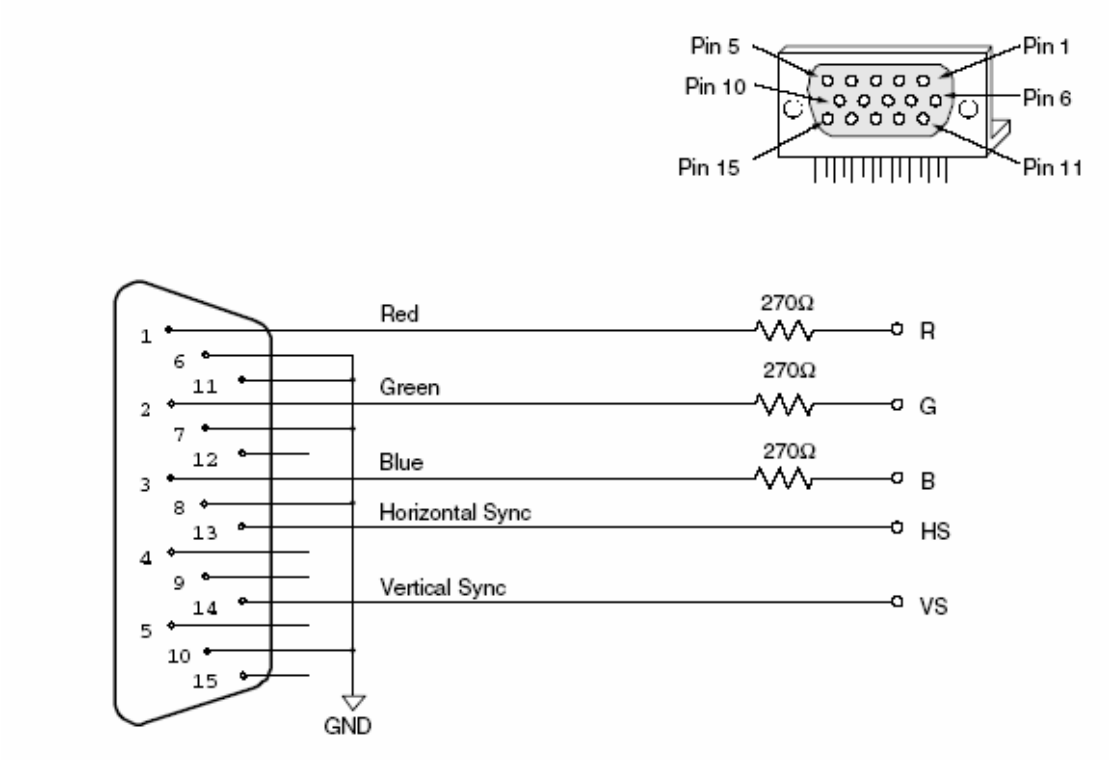
1．Learn CRT monitor working theory.
2．Learn VGA interface timing control.
3．Write a program to control the monitor.

2.2.2 Experiment theory

This experiment requires using verilogHDL to write an outputting program which can control the VGA port on the experiment board. Control the monitor through experiment board to display the color in the finished programs. To guarantee the monitor working normally, you need to know the structure of VGA port firstly, then, the CRT monitor working principles and the timing relationship of VGA port outputting signals. .

1．VGA port structure:

VGA port is the video output port. It includes 15 pins as following:



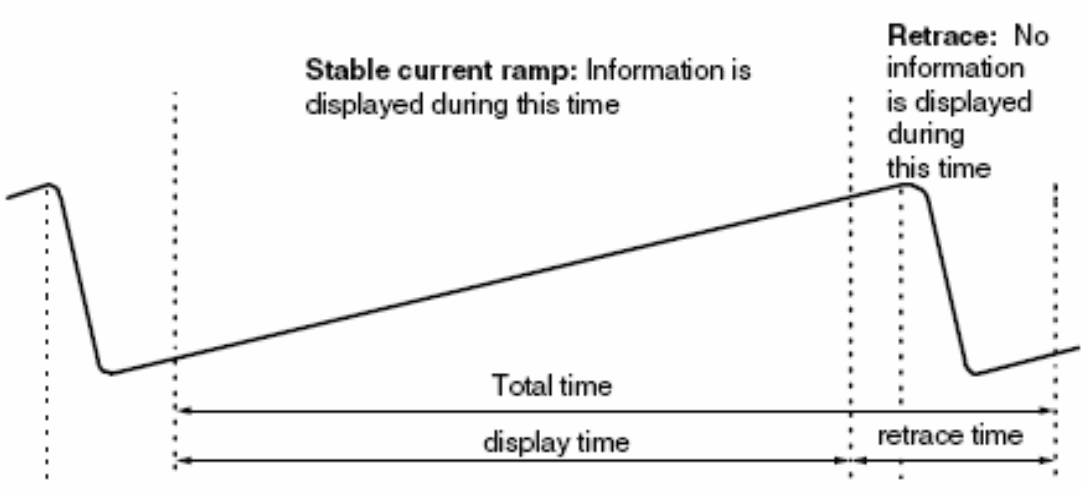
In common connection ways, there are 5 most important pins in 15. They include 3 basic color lines red, green, blue and two control lines (level and vertical). We can display 8 different colors in the screen as following:

red	green	blue	Display color
0	0	0	Black
0	0	1	Green
0	1	0	Blue
0	1	1	Blue green
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

On the development board, each basic color line (red, green and blue) is controlled by three input lines. These three control lines have different resistances. Under these three input lines controlling, the three basic colors (red, green and blue) are divided into 8 levels separately. Theoretically, the VGA interface on the development board has 9 color control lines in total which can display 512 different colors.

2. CRT monitors working theory:

Inside of the monitor, the current flows through the coil and generates the magnetic field. In this way, it control electron beam flow the monitor surface, from left to right in level and from up to low in vertical. Following picture is an example of level direction. Only in the positive direction flow (left to right, up to low), the monitor works. When the electron returns to monitor's left or up, the monitor doesn't work.



3、VGA driver program's FPGA achievement:

Driver program mainly finishes the following tasks: generate the synchronize signal (line, row) according to the VGA timing requirement, and output the data of color to be displayed in specified time (pixels valid period) to RGB.

In different displaying modes and refresh frequencies, detailed synchronize signals (front, behind, synchronization signal) has different valid pixels. You have to set according to pixels clock frequency. Like upper table:以800*600，60HZ, pixel clock is 40M, pixel clock =（800+40+128+88）*（600+1+4+23）*60=40MHZ. In designing, you can choose suitable display mode according to system clock frequency.

2.2.3 Experiment content

This experiment mainly has two parts: one is displaying color lines required by experiment in the monitor VGA which is simple. Another is to simulate a Ping-Pong game in VGA monitor. Use keys on development board SW1, SW2, SW3, SW4 to control rackets and jumper SW6 to control balling status.

The most important thing in programming to control VGA is to learn the timing and working mode of VGA interface. The first experiment is to show some color belts on the monitor. Therefore, giving the different RGB value (different currents) according to different areas of the data in scanned registers' when the monitor is scan-displaying will be ok. For example, we have to show a vertical red belt in the left most of them monitor. As the monitor is progressive scan, we can judge whether the data value in the register is less than a fix value. Set R be 1 and GB be 0 when meet requirement, or all is set 0.

Ping-Pong program is more complex. You can program a VGA controller first. And display the table, rackets and ball by controlling this VGA controller. At last, write the programs of balling and score recording

2.2.4 Experiment steps

1. Set up a new project.
2. Set up source file and programming
3. Integrate, layout, and route;
4. Download and debug.

2.2.5 Experiment result

1. You can see the color belts in the monitor according to the experiment requirements.
2. You can play Ping-Pong game on the monitor.

Experiment10 Serial communication experiment

2.3.1 Experiment purpose

1. Learn RS232 interface agreement;
2. Program to achieve the communication between serial and PC.

2.3.2 Experiment theory

1、Serial description：

RS-232-C standard is initially the remote communication connection data terminal equipment (DTE) and data communication equipment (DCE) to customize. Therefore, this standard's set up doesn't consider the application requirements of the computer system. However, now days, it is widely used as the connection standard between the computer (computer interface) and terminal or peripheral proximal. Obviously, some rules of this standard are different from computer systems, and some are conflicts. With the knowledge of this background, it is easy for us to understand the not compatible place between RS-232C and the computer.

Second, the "send" and "receive" mentioned in RS-232C standard, are defined at the position of DTE instand of the DCE. As in the computer system, messages sent between both of the CUP and I/O device is based on the DTE, both parties can send and receive.

The full name of RS-232C standard（agreement）is standard EIA-RS-232C . Here, EIA is stand for Electronic Industry Association, RS is stand for recommended standard, 232 is ID number, C stands for the latest adjustment of RS232 （1969）.

2、Serial electrical standard：

RS232 uses negative logic instand of TTL level interface standard. That is when the logic is "1" the range is -3 V～-15 V；the logic is "0" the range is +3 V～+15 V；EIA-RS-232C defines to the electrical features, logic level and all kinds of signal line features:

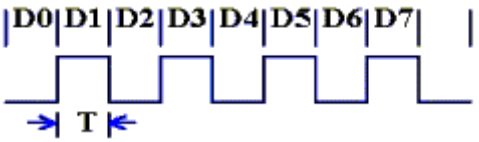
On TxD and Red：logic 1(MARK)=-3V～-15V; logic 0(SPACE)=+3～＋15V；

On the control lines as RTS、CTS、DSR、DTR and DCD: signal valid (connected, ON, positive voltage) ＝+3V～+15V; signal invalid (disconnected, OFF, negative voltage))=-3V～-15V

EIA-RS-232C which is different from TTL (uses high / low level to show the different logic status) uses positive / negative level to show the logic status. Therefore, to connect to the computer interface or TTL devices, you have to change the level and logic between EIA-RS-232C and TTL circuit. To achieve this, you can use discrete components as well as IC. Here, what we use is MAX3232 changes the signal sent/received by the interface to TTL level.

3、 Serial communication agreement:

“Serial communication” means using one signal line between peripherals and the computer (more control line needs if grand line asked). Data are transformed in one signal line bit by bit. Each bit gets one fixed time period. As following picture shows:



This communication way uses fewer data lines which can save the cost in long distance communications. Of course, the speed is lower than parallel way.

The transform between CPU（FPGA is equal to one CPU）and interface is parallel way, and be serial way between peripheral and interface. Therefore, in serial interface, there must be “receiving displacement register” (serial-parallel) and “sending displacement register” (parallel – serial)

During the data inputting, data enter interface’s “receiving displacement register” bit by bit from peripheral. When “receiving displacement register” has finished the receiving bits of 1 character, the data enter “data inputting register” from “receiving displacement register”. CPU reads the received symbols from “data inputting register”. (Parallel reading, D7~D0 are read to accumulator at the same time). The speed of “receiving displacement register” is decided by “receiving clock”.

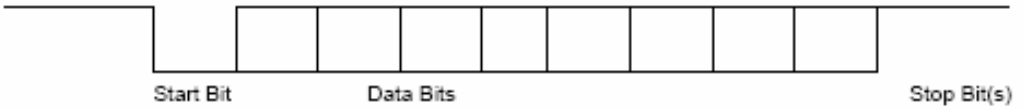
During data output, CPU send the symbols to be output (parallel) to “data

outputting register", the content of "data outputting register" to "sending displacement register". And then, the "sending displacement register" sends the data to peripheral bit by bit. The speed of "sending displacement register" is decided by "sending clock". The "controlling register" in interfaces is used to accommodate all controlling message sent to the interface by CPU. These messages decided the working mode of the interface.

The circuit which can finish "serial-parallel" exchanging as described before is called "common asynchronous receiver transmitter"（UART：Universal Asynchronous Receiver and Transmitter）. It includes doubt buffer data sending register, Parallel-serial changing equipment, double buffer data inputting register, Serial-parallel chaning equipment.

RS232 communication agreement's basic structure

Start bit is low and stop bit is high.



Baud rate is 300~115200 bit/s，8bit data bit, one or two bits stop bit, odd parity, even parity or no parity bit.

2.3.3 Experiment content

This experiment needs serial debugging software, Baud rate is 9600, sending/receiving data.

2.3.4 Experiment steps

1. Set up a new project.
2. Set up source file and programming
3. Integrate, layout, and route;
4. Download and debug.
5. Use serial line to connect PC and development board, open the serial debugging software and the Baud rate is set 9600.

2.3.5 Experiment result

Achieve receiving / sending data by the serial debugging tools. For example: send 45, and receive 45

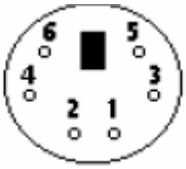
Experiment11 PS2 interface control and display experiment

2.4.1Experiment purpose

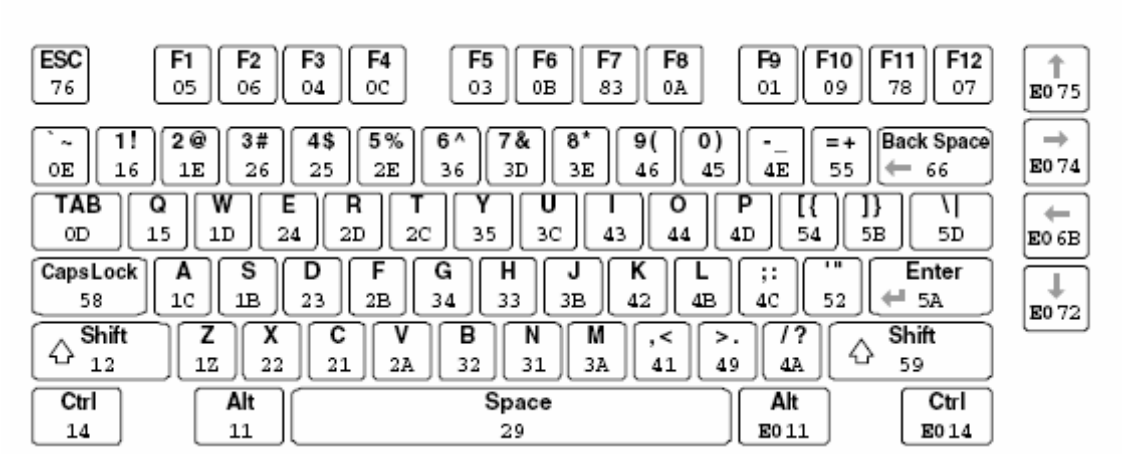
1. Learn PS2 interface agreement.
2. Learn the keyboard working theory.
3. Write program on the development board to achieve read the keyboard inputting message through interface PS2.

2.4.2 Experiment theory

This experiment is to write a program which can achieve PS /2 port features. PS/2 keyboard fulfills Two-way synchronization serial agreement, In other words, each time of send one bit data in data line and pulse in clock line, it can be read. Keyboard can send data to host. Host also can send data to devices. But the host always has priority in general lines. It can inhabit the communication from keyboard at any time if the clock is down. This experiment mainly is to achieve the data transmission from keyboard to the host. First of all, we have to know the PS/2's structure and pins features.

Plug	Socket	Pin
		1—data
		2—not achieve, reserve
		3—porwer ground
		4—power，+5V
		5—clock
Plug	Socket	6—not achieve, reserve

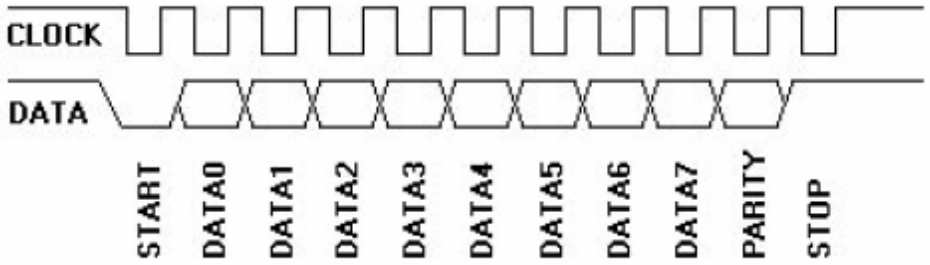
There is only one data port in upper table. To distinguish many keys, one high-efficiency distinguish way is needed. Keyboard processor spends a lot of time to scan or monitoring keyboard matrix.If it finds some keys are pressed released or hold the keyboard, it will send message pack of scan codes to the computer. There are 2 kids of scan code: "pass code" and "breaking code". When one key is pressed or hold, it will send "pass code"; when one key is released, it will send "breaking code". Each key is distributed the only "pass code" and "breaking code". In this way, the host knows the exact key by searching the only scan code. The "pass code" and "breaking code" of each key composite the scan code set. Following pictures includes the scan codes of most keys on the keyboard:



When the key is released, the keyboard will put “F0” in front of the scan code as the release signal. At the same time, some keys are extended keys. Put “E0” in front of their scan codes as beginning. When this kind of key is released, it will append “E0F0” to the scan code.

Let us know how signal inputs through keyboard by PS/2 port’s data line. First, the keyboard will check whether data line and clock line are high. Only both of them are high, you can write data. The data send from keyboard to host can be read at the clock signal’s falling (clock changes from high to low).

Keyboard mainly uses the serial agreement that each frame has 11bits:the first bit is start, be “0” forever; following 8 bits are data bits, lined from low to high; following is odd/even parity bit; last is ending bit, be “1” forever.



2.4.3 Experiment content

This experiment achieves the controlling of keyboard, LCD, RS232 and etc by programming the development board. Display the keyboard input data on the LCD, or the PC super terminal by RS232.

2.4.4Experiment result

Display the input characters on LCD.

Experiment12 USB interface read/write control experiment

2.5.1 Experiment purpose

1. Learn USB interface working theory.
2. Master CY7C68013 working timing.
3. Write the program to control the USB on the development board to read/write data.

2.5.2 Experiment theory

EZ－USB FX2 from Cypress Semiconductor is the first microprocessor integrated USB2.0. It integrates USB2.0 transceiver, SIE (serial interface engine), enhanced 8051 microcontroller and programmable peripheral interface. This original structure of FX2 permits the transmit rate to reach 56Mbytes/s that is the maximum belt width of USB2.0. In FX2, the intelligent SIE can hard process many USB1.1 and USB2.0 agreements to reduce the developing time and guarantee the USB's compatibility. GPIF（General Programmable Interface） and main/minor ports FIFO (8 bits and 16 bits data general line) provide simple and seamless connection interfaces to ATA、UTOPIA、EPP、PCMCIA and DSP.

CY7C68013 integrates following features:

● USB2.0 transceiver, SIE (serial interface engine), and enhanced 8051 microprocessor.

● Software running: 8051 starts from internal RAM, and can with the help of following ways to load programs:

（1） download through USB;

（2） load from EEPROM;

（3） through external storage device.

● Four programmable BULK/INTERRUPT/ISOCHRONOUS ports; You can choose two, three or four buffer.

● 8 bits or 16 bits external data interface.

● through programmable interface （GPIF）

（1） connect to parallel port directly, 8 and 16 bits.

（2） programmable waveform descriptors and configuration register.

（3） support several Ready input and Control output

● integrate standard 8051 core and has following enhanced features:

（1） can reach 48MHz clock.

（2） each command gets four clock circle;

（3） two USARTs；
（4） three timers/ counters;
（5） expanded interrupt system.
（6） tow data pointer.

● 3.3V power system;
● intelligent serial engine（SIE）；
● Vector USB interrupt
● Independent data buffer for SETUP and DATA pack control transmit.
● integrates I2C controller, running speed can reach 100 or 400KHz；
● Four FIFO，can connect to SIC, DSP and ect seamlessly.
● Professional FIFO and GPIF auto vector interrupt.
● Can be used in DSL Modems、ATA interface、camera、Home PNA、WLAN、MP3 player, internet and ect.

USB starting way and enumerate:

When powered, the internal logic will check the first character (0xC0 or 0Xc2) of the EEPROM which connected to the I2C general line. If it is 0xC0, it will use VID/PID/DID in EEPROM to instand internal storage value. If it is 0xC2，the internal logic will load the content of EEPROM to internal RAM. If not find EEPROM, FX2 will use internal storage's descriptor to describe the enumerator. FX2 default VID/PID/DID is 0x04B4/ 0x8613/0xxxyy.

When the first time insert the USB, FX2 will enumerate and download firmware and USB descriptor list by USB cable automatically. Then, the FX2 will enumerate again. This time, it mainly makes the definition of the device by download information. These two steps are called re-enumeration. Once the device is inserted, it works.

Program / data memory

● Internal data RAM

The internal data RAM of FX2 are divided into three different areas: LOW 128、Upper 128 special feature register(SFR) room. Low 128 and upper 125 are common RAM while SFR includes FX2 controlling and status register.

●External program memory and data memory.

FX2 has 8K chip RAM which locates in 0x0000－0x1FFF；512 bytes Scratch RAM which locates in 0xE000－0xE1FF. Though physically, Scratch RAM locates in the chip, it can be found as the external RAM by firmware. FX2 keeps data address space 7.5K（0xE200－0xFFFF as controlling / status register and port buffer.

Note: only data memory space is kept, program memory （0xE000－0xFFFF）isn't. Port buffer FX2 includes 3 64 bytes port buffers and 4K space which can be configured to different ways buffers. 3 64 bytes buffer is EP0、EP1IN and EP1OUT。EP0 is used as controlling port which is two-way port and can be IN or OUT. When it needs to control transit data, FX2 firmware read / write buffer EP0. But 8 SETUP byte data won't appear in the 64 bytes EP0 port buffer. EP1IN and EP1OUT use the independent 64 byte buffer. FX2 firmware can configurate these ports to be BULK、INTERRUPT and ISOCHRONOUS

transmission way. These 2 ports only can be visited by firmware just like EP0.It is different from big port buffer EP2、EP4、EP6 and EP8. These four port buffers are mainly used to do the data transmission to chip or out-chip which doesn't need the firmware's participation. EP2、EP4、EP6 and EP8 are high belt width, big buffers. They can be configurated in different ways to meet the belt width needs.

External interface FIFO

Big port buffers (EP2、EP4、EP6 and EP8) are mainly used to do the high-speed (480Mbits/s) data transmission. It can set up the high speed data transmission by the seamless connection between FIFO data interface and external ASIC and DSP processers. It has common interface: Slave (subordinate), FIFO (external main) or GPIF (internal main), synchronous / asynchronous clock, internal or external clock and etc.

Interrupt source:

FX2 interrupt structure enhances and expands part of the interrupt source based on the standard 8051 MCU. Following table shows the interrupt source:

FX2 interrupt	Interrupt source	Interrupt vector	Priority
IE0	INT0 Pin	0x0003	1
TF0	Timer0 Overflow	0x000B	2
IE1	INT1 Pin	0x0013	3
TF1	Timer1 Overflow	0x001B	4
RI_0 & TI_0	USART0 Rx & Tx	0x0023	5
TF2	Timer2 Overflow	0x002B	6
Resume	WAKEUP/WU2 Pin	0x0033	0
RI_1 & TI_1	USART1 Rx & Tx	0x003B	7
USBINT	USB	0x0043	8
I2CINT	I2C BUS	0x004B	9
IE4	GPIF/FIFOs/INT4 Pin	0x0053	10
IE5	INT5 Pin	0x005B	11
IE6	INT6 Pin	0x0063	12

Among them, 27 USB applicator share USB interrupt and 14 FIFO / GPIF source share INT4.

Detailed chip introduction and using description see the chip user manual.

2.5.3 Experiment content

This experiment is to set up the data transmit between FPGA and PC by USB interface which includes data reading and data writing. Testing way see the testing file.

2.5.4 Experiment steps

All experiments below are based on default user has installed the development pack EZ-USB. If the user hasn't done this, please install the EZ-USB provided in the disk and install Keil to write firmware program. And, these two samples will use VC++6.0(or VC.NET). User has to confirm the VC has been installed already.

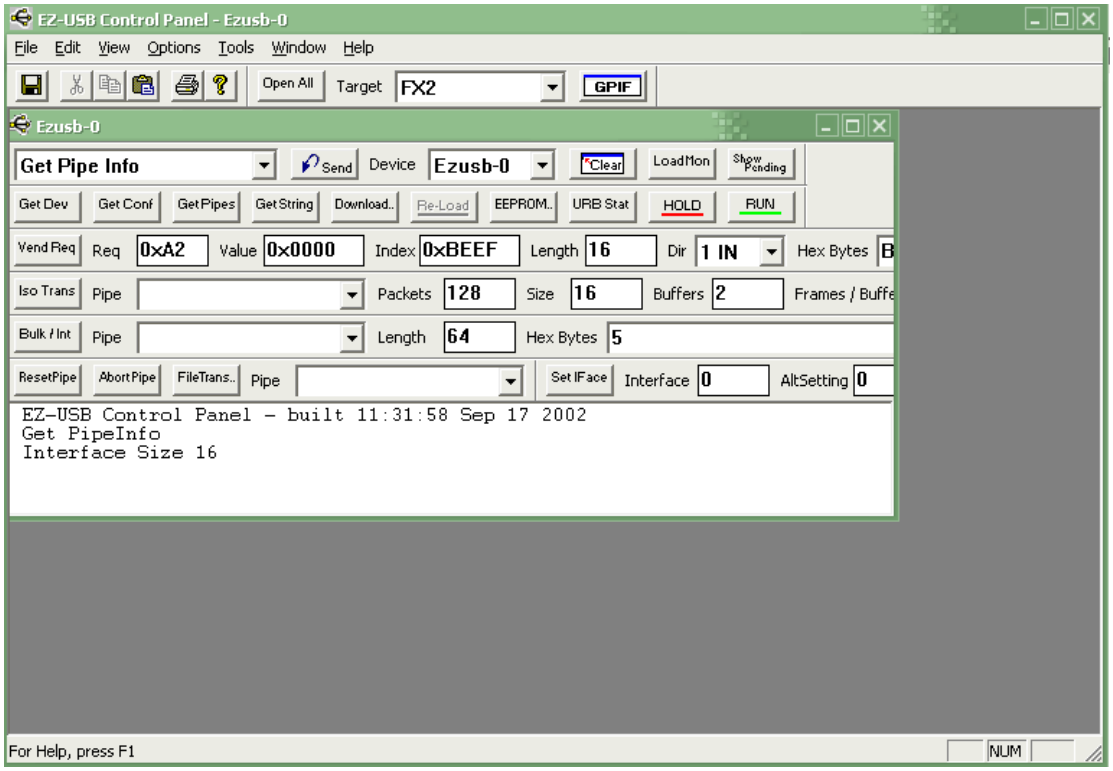
◆ BULK data transmission experiment:

Experiment steps are as following:

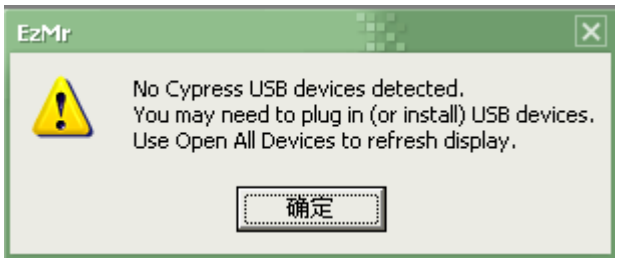
1、first of all, connect the development board to PC by USB line.

2、open /Cypress/EZ-USB Control Panel

You will see following window:



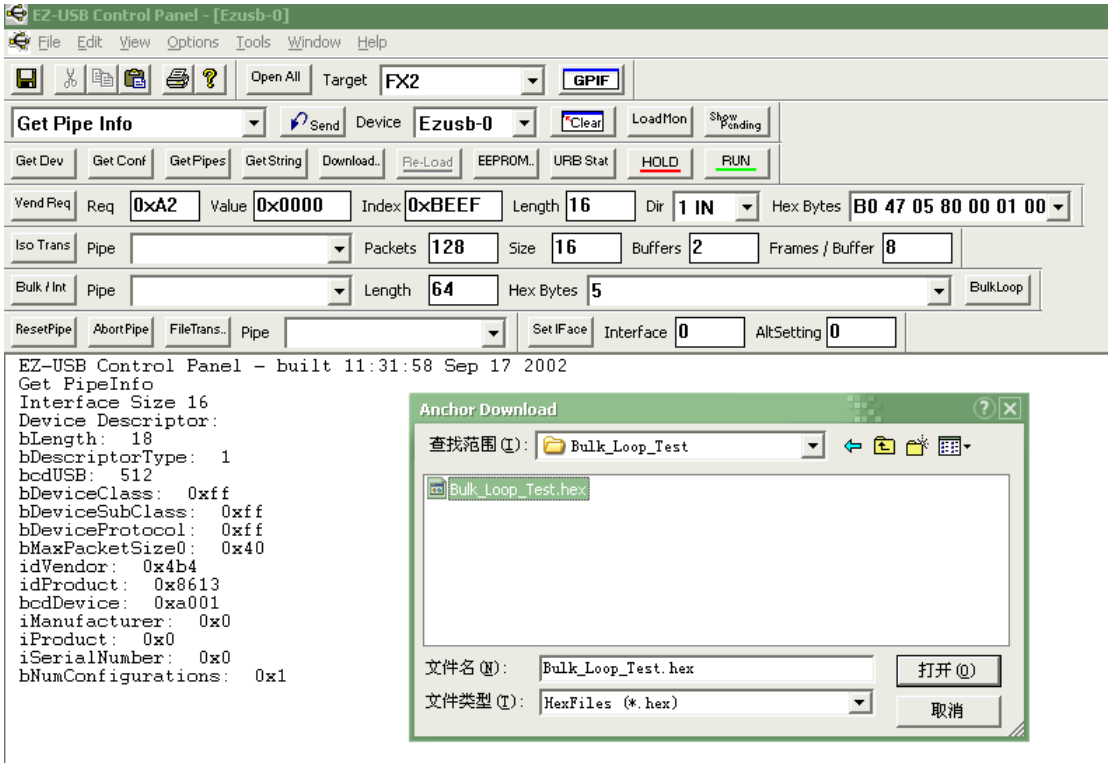
As the upper picture, the USB has been connected to PC. If following dialog comes out:



Means there is no connection. Please check whether the USB line has been inserted and the development board is powered.

1、Click download and find

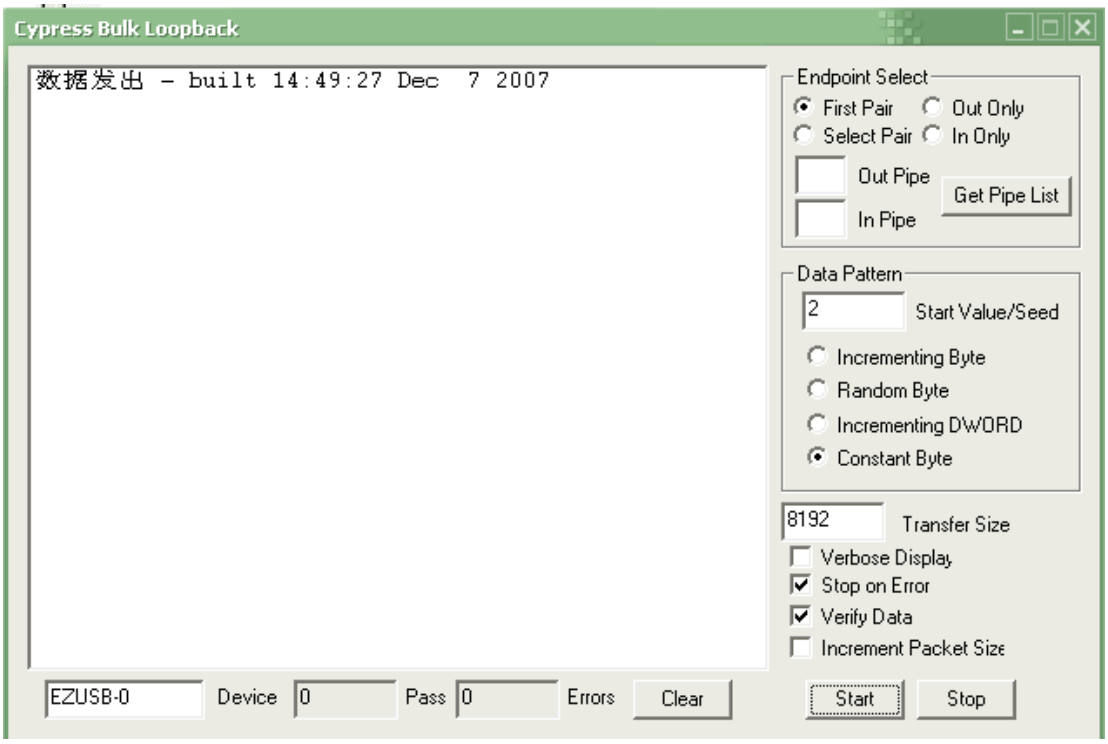
The file Bulk_Loop_Test.hex in s12_usb\Bulk transmission test \Keil firmware project\Bulk_Loop_Test\. Click download. When the download is finished, you can hear the “ding” of the USB disconnecting firstly.



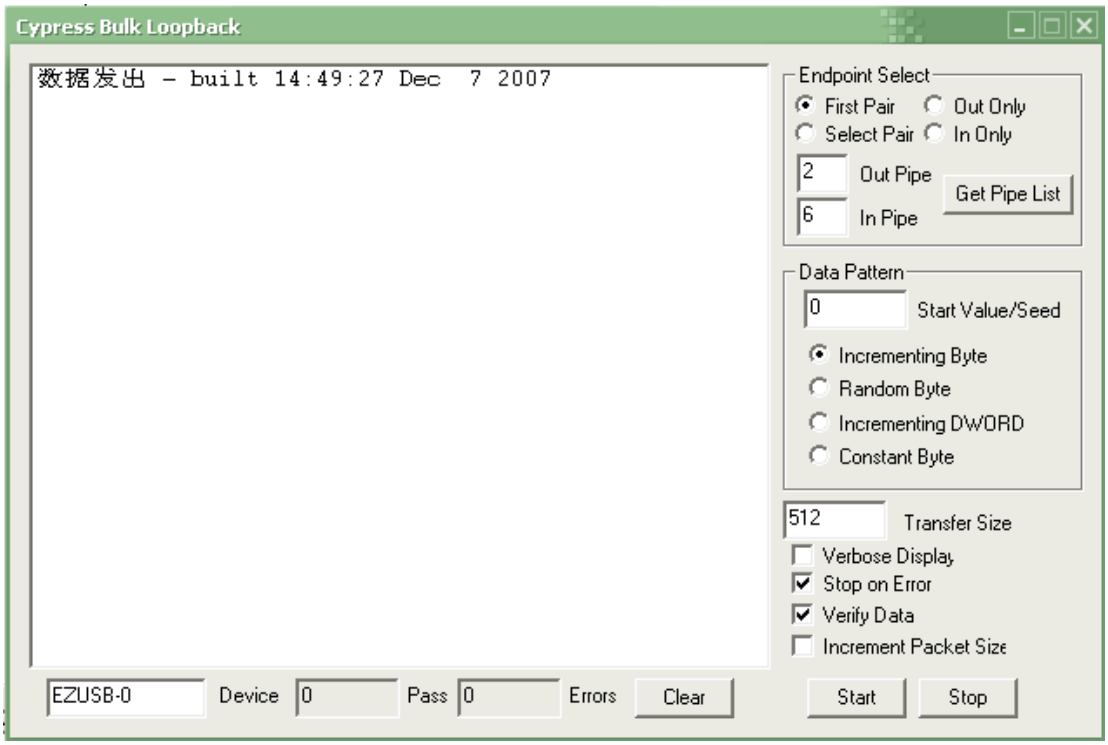
4、When the download is finished, open

The VC project in s12_usb\Bulkt transmission test \VC project \bulkloop.

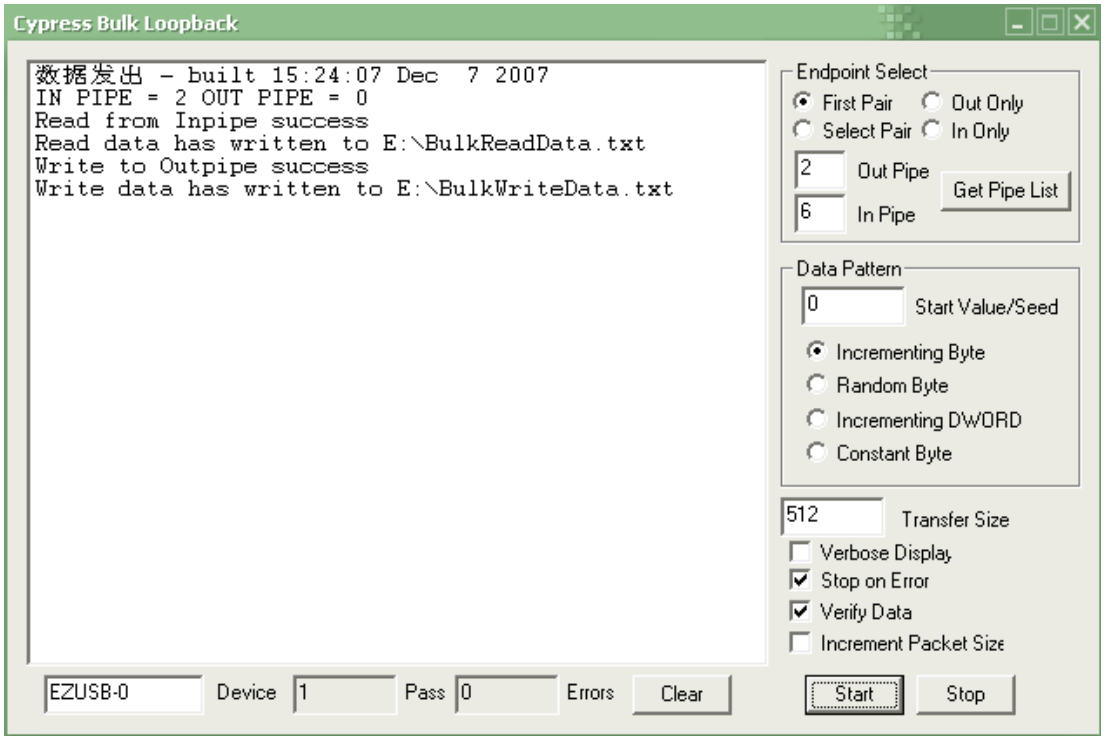
Running interface is as following:



Set Out Pipe：2，In Pipe：6，First Pair，Start Value/Speed is 0，incrementing Byte ， Transfer Size is 512,as following

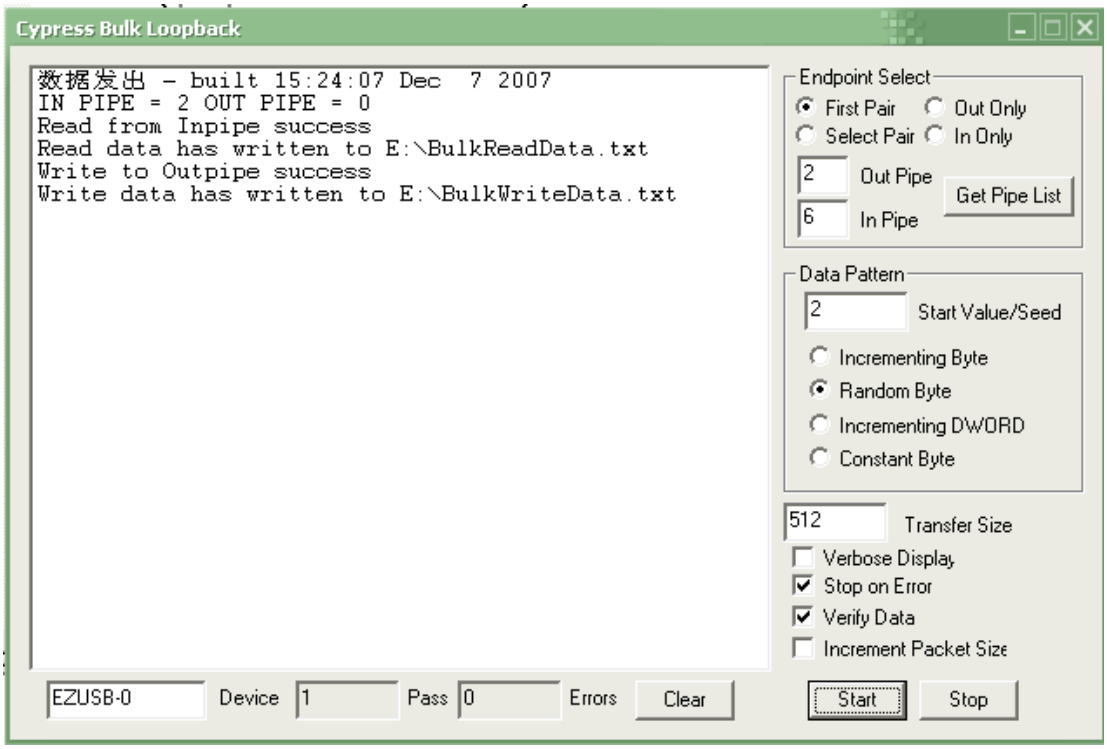


Click start and the result comes:

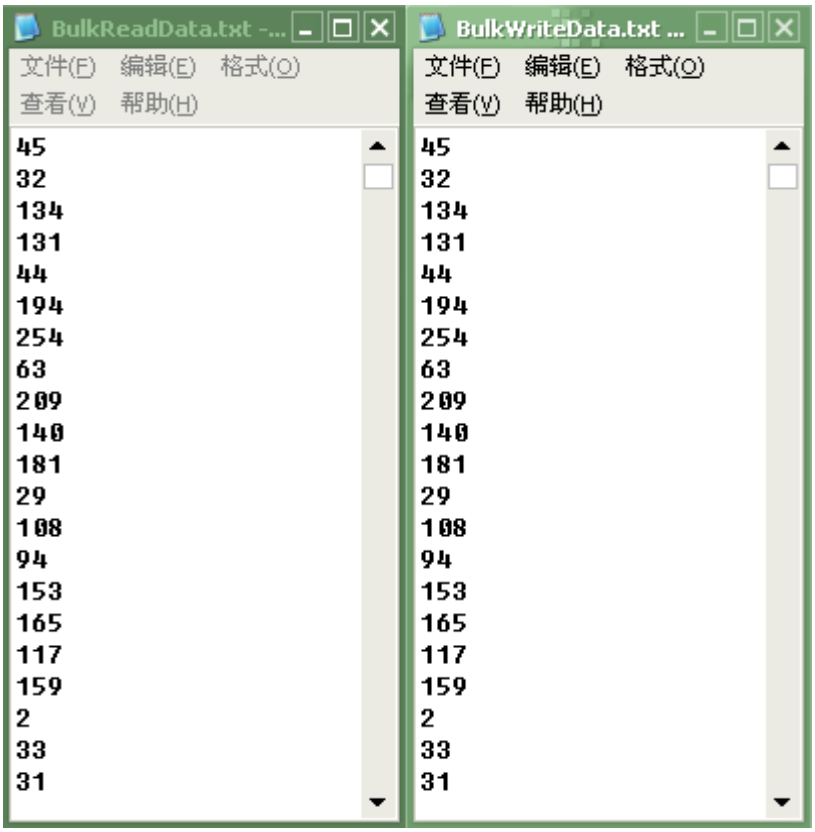


You can see, the program stores the sending data and receiving date into two files. Open these two files and see the corporations:

Data are completely the same. You can also choose random bytes that are Random Byte. Results are as following:



Compare two files as following:



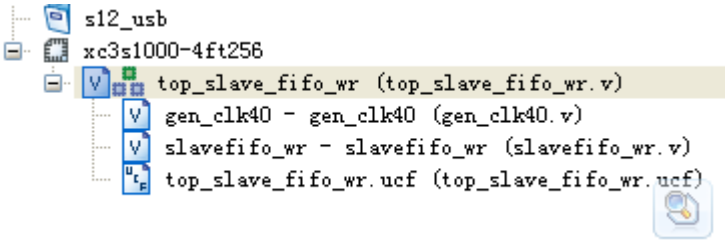
The result shows firmware program and VC program designing are correct. They can finish BULK transmission accurately. User can adjust them on the actual needs.用

◆ Slave_FIFO mode data transmission experiment

Steps are as following：

1、Open ISE project;

The project files s12_usb.ise in s12_usb\SlaveFIFO mode data transmission test \ISE project\3s1000_slavefifo



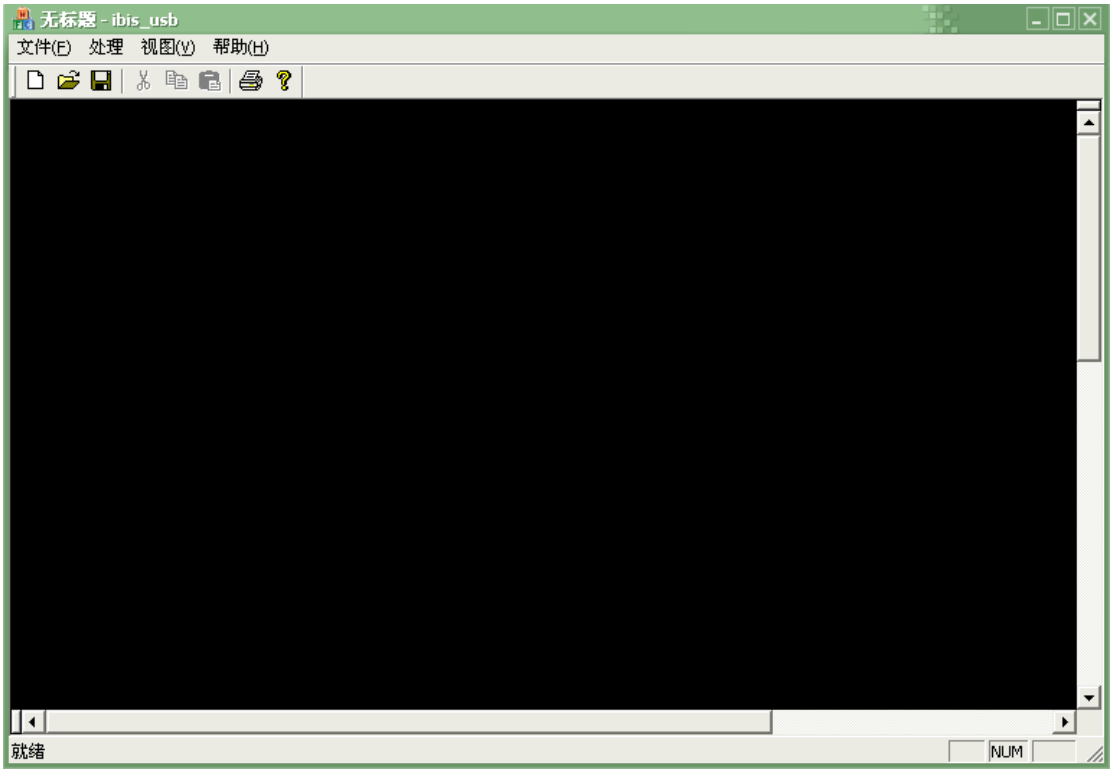
Download top_salve_fifo_wr.bit file to FPGA

2、Open Cypress\Control Panel, and download

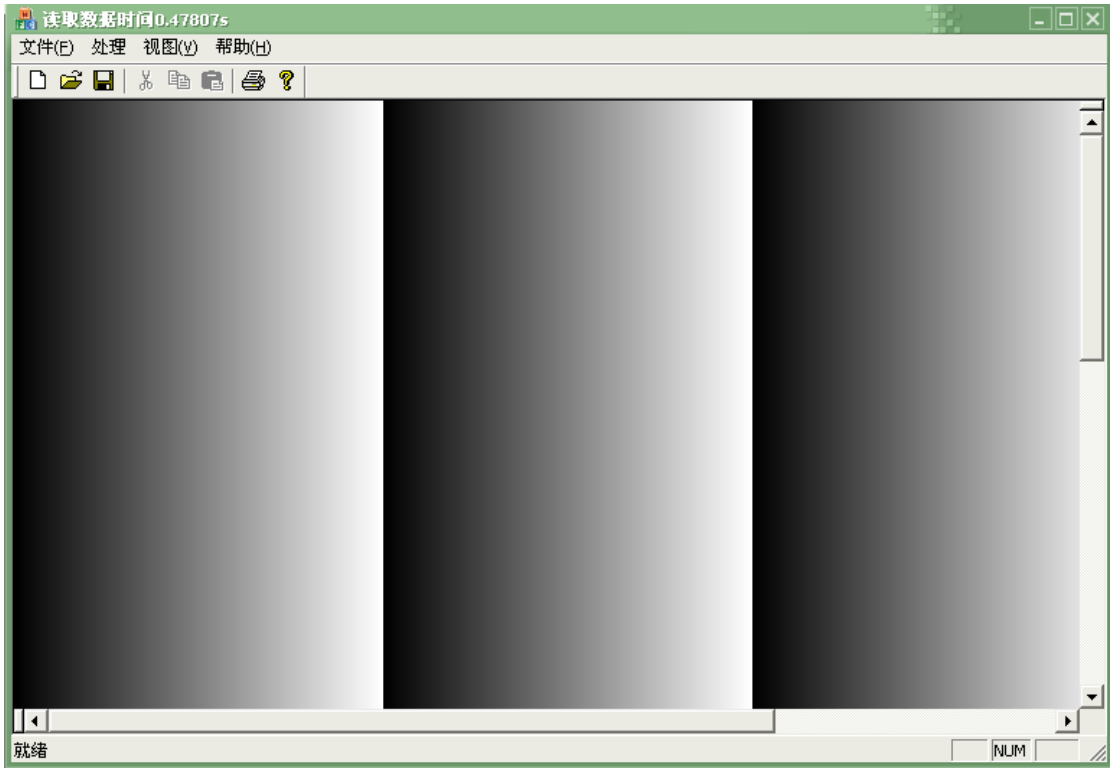
s12_usb\SlaveFIFO mode data transmission test \Keil firmware project\Slave_FIFO_rd_wr\Slave_FIFO_rd_wr.hexfile

3、Open VC project:

In s12_usb\SlaveFIFO mode data transmission test \VC.NET project \ibis_usb, the interface is as following :



Click process\read FIFO and get following result:



You can see the gradient stripes which prove the write timing of FPGA program design is correct. And you can see the read time in the title bar. This program can achieve lasting reading. Each time of read an image, the lights on development board are on/off alternately.

Here, only provide a very simple sample for user to study and using. User can do the adjustment on this base according to the actual needs.

2.5.5 Experiment result

Achieve the USB communication between FPGA and PC. And check the communication correctness on the PC. The details of EZ-USB are in the certain files in the provided project folder.

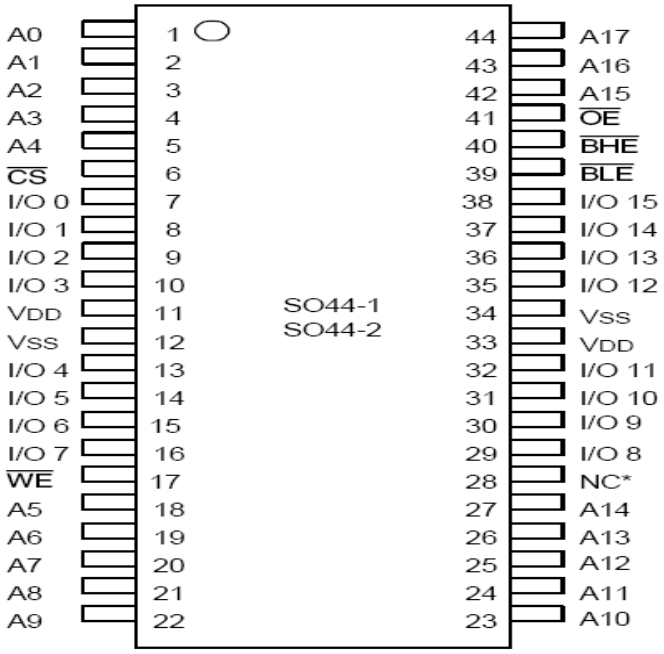
Experiment 13 SRAM read/write control experiment

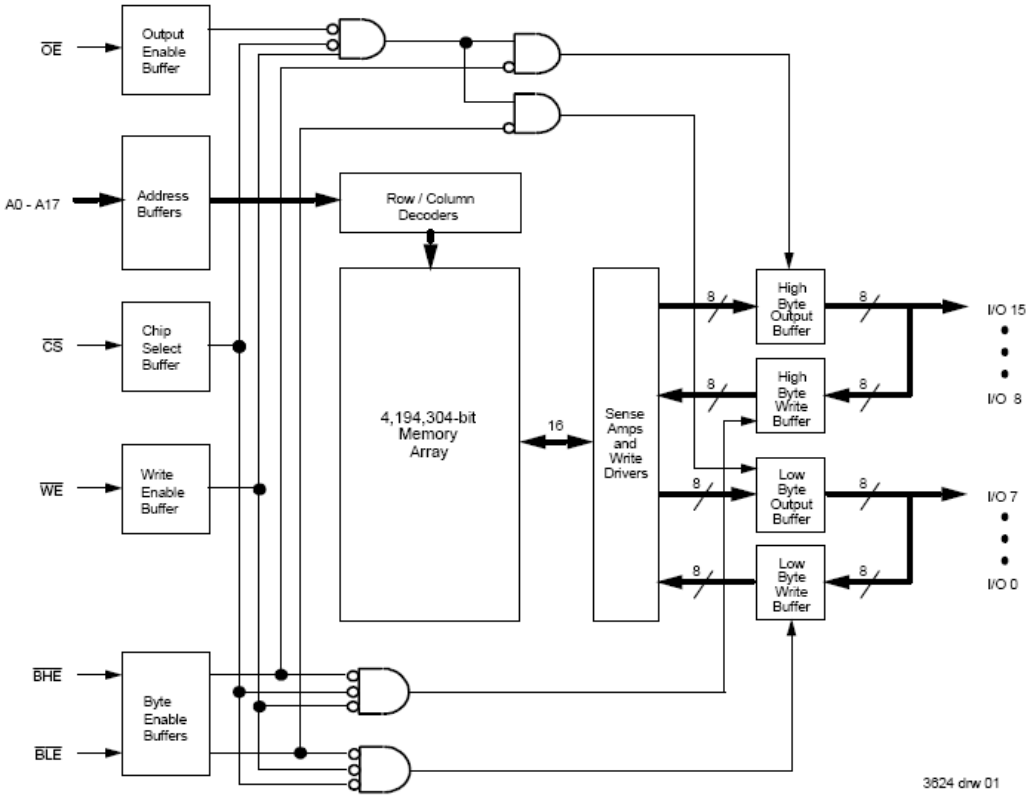
3.1.1 Experiment purpose

1. Learn SRAM memory structure
2. Handle SRAM memory read/write timing;
3. Write a program to control the SRAM read / write

3.1.2 Experiment theory

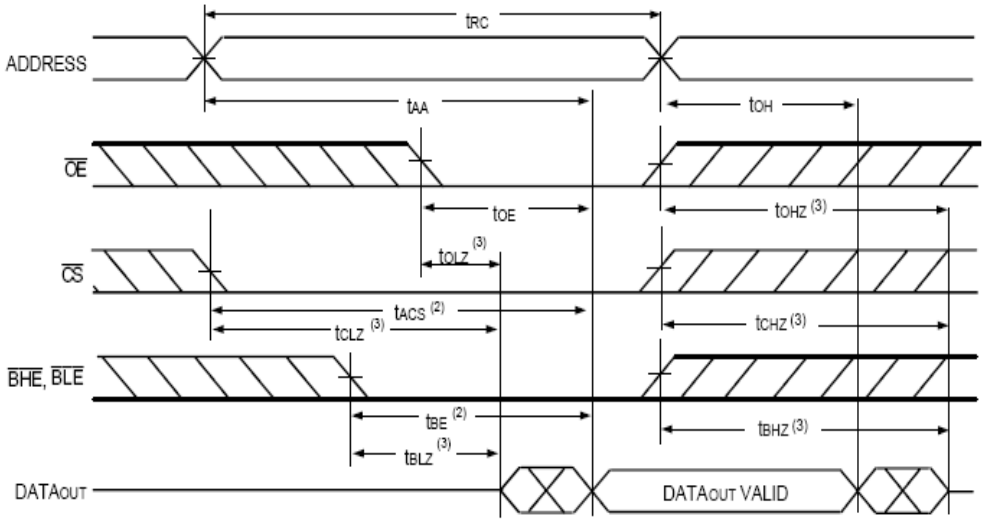
The SRAM chip used on the development board is IDT71V416S,
The external package and internal structure are as following:





SRAM is the easier reading /writing controlled way in all memory kinds which means its reading / writing timing is easier. Details are as following:

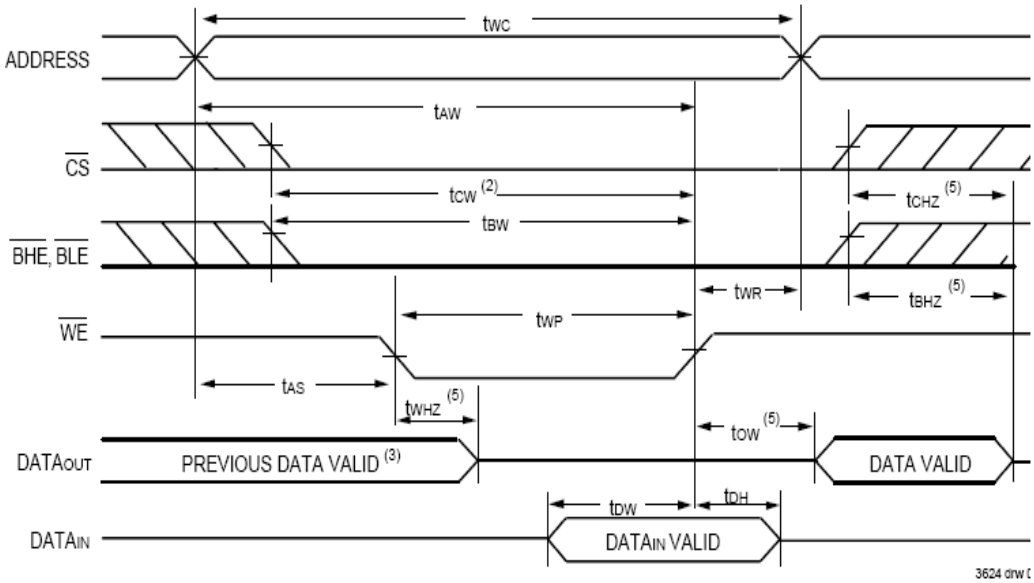
（1） Read timing：



READ CYCLE								
t_{RC}	Read Cycle Time	10	—	12	—	15	—	ns
t_{AA}	Address Access Time	—	10	—	12	—	15	ns
t_{ACS}	Chip Select Access Time	—	10	—	12	—	15	ns
t_{CLZ}[1]	Chip Select Low to Output in Low-Z	4	—	4	—	4	—	ns
t_{CHZ}[1]	Chip Select High to Output in High-Z	—	5	—	6	—	7	ns
t_{OE}	Output Enable Low to Output Valid	—	5	—	6	—	7	ns
t_{OLZ}[1]	Output Enable Low to Output in Low-Z	0	—	0	—	0	—	ns
t_{OHZ}[1]	Output Enable High to Output in High-Z	—	5	—	6	—	7	ns
t_{OH}	Output Hold from Address Change	4	—	4	—	4	—	ns
t_{BE}	Byte Enable Low to Output Valid	—	5	—	6	—	7	ns
t_{BLZ}[1]	Byte Enable Low to Output in Low-Z	0	—	0	—	0	—	ns
t_{BHZ}[1]	Byte Enable High to Output in High-Z	—	5	—	6	—	7	ns

- When do the reading, you have to set the signal #WE high.
- The address of the data to be read has to be gave following the signal #CE falling at the same time or prior to it.
- After gave the address and read controlling signal, it can read the data after period of time (normally is reading at the next clock cycle's raising)

（2） Write timing:



WRITE CYCLE								
tWC	Write Cycle Time	10	—	12	—	15	—	ns
tAW	Address Valid to End of Write	8	—	8	—	10	—	ns
tCW	Chip Select Low to End of Write	8	—	8	—	10	—	ns
tBW	Byte Enable Low to End of Write	8	—	8	—	10	—	ns
tAS	Address Set-up Time	0	—	0	—	0	—	ns
tWR	Address Hold from End of Write	0	—	0	—	0	—	ns
tWP	Write Pulse Width	8	—	8	—	10	—	ns
tDW	Data Valid to End of Write	5	—	6	—	7	—	ns
tDH	Data Hold Time	0	—	0	—	0	—	ns
tOW[1]	Write Enable High to Output in Low-Z	3	—	3	—	3	—	ns
tWHZ[1]	Write Enable Low to Output in High-Z	—	6	—	7	—	7	ns

● During writing, signal #OE can be high or low which will not effect on the operation.

● Writing operation has to give the data and address to be wrote at the signal #WE ‘s rising. Write into SRAM in next clock circle.

● In the written status $\overline{BLE}$, at least one $\overline{BHE}$ is generated at low level.

3.1.3 Experiment content

This Experiment is to control the SRAM on the development board to read the data from specified address and write the data to the specified address. And compare whether the read data is the same as the write one. If they are the same, means SRAM read/write successfully.

3.1.4 Experiment steps

1. Setup project.
2. Add source file.
3. Integrate, layout, and route;
4. Download and debug.

3.1.5 Experiment result

Read data and write data are the same. The error indicator light on the development board is off.

Experiment14 SDRAM read/write control experiment

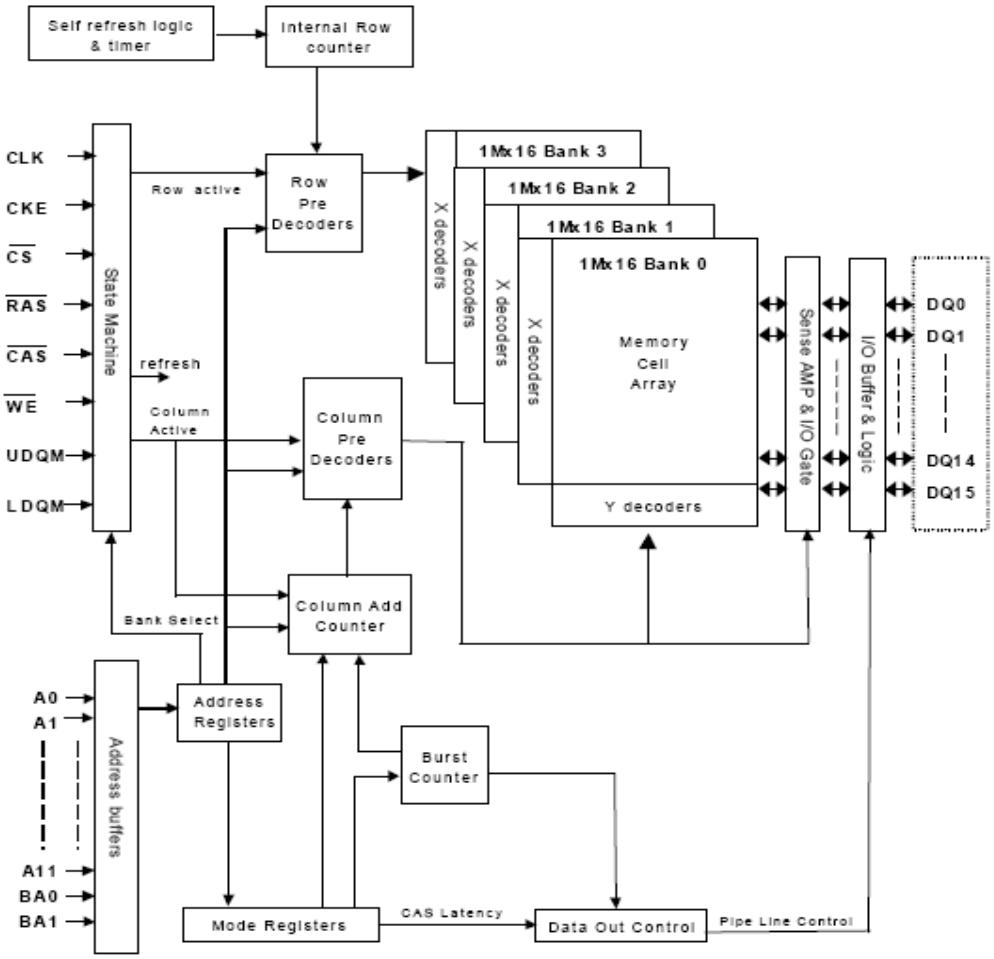
3.2.1 Experiment purpose

1. Know the internal structure of SDRAM.
2. Handle SDRAM working principle and reading and writing timing
3. Program SDRAM controller.

3.2.2 Experiment theory

In the high speed real time or none real time signal processing system, using large storage to achieve data cache is a necessary part which is also the importance and difficulty in the whole system achieving. SDRAM has advantages such as low price, high precision and high speed of reading and writing. It is the first choice for the data cache. However, the structure of SDRAM is quite different from SRAM's. Its timing controlling is more complex which limit the using range of it.

Following is the internal structure of SDRAM:



SDRAM devices' pins are separated into three parts: controlling signal, address and data. Following are the detailed definitions:

SDRAM(×16) Pin Assignment

Signal Name	Type	Description
CS	Input	Chip Enable
CLK	Input	Clock
CKE	Input	Clock Enable
RAS	Input	Row Address Strobe
CAS	Input	Column Address Strobe
WE	Input	Write Enable
DQML, DQMH	Input	Data Mask for Lower, Upper Bytes
BA	Input	Bank Address
A[0:10]	Input	Address
DQ[0:15]	I/O	Data

Usually one SDRAM includes several BANK，Each BANK's storage unit is addressing by line and row. Because of this special storage structure, SDRAM has following working features:

● SDRAM's initialization---after SDRAM is powered by 100～200μs, there

must be one initialization process to configure SDRAM ‘s mode register. Mode register’s value decides the working mode of SDRAM.

● visit storage unit --- To minus I/O pins，SDRAM reuses address line. Therefore, when read/write SDRAM, use ACTIVE to active BANK to be read/wrote firstly and latch line address. Second, when the read/write command is valid, latch row address. Once BANK is active, only after finish the pre-charge command, you can reactive the same BANK.

● refresh and pre-charge--- To improve the storage density, SDRAM uses silicon capacitor to store data. Capacitor always tends to discharge. Therefore, there must be regular refresh cycle to avoid data missing. Refresh cycle can be got by Min refresh cycle / clock cycle. Pre-charge the BANK or close the active BANK can pre-charge the special BANK and also can effect on all BANK，A10、BA0 and BA1 which used to choose BANK。

● Operation control --- SDRAM’s detailed controlling commands finishing are assisted by some specified controlling pins and address lines. CS、RAS、CAS and WR in the clock rising statues decide the detailed operation action. In some operation actions, address line and BANK choosing control line are input as assistant specifications. Because of the special storage structure, SDRAM has more operation commands which are different from SRAM that has simple read/write. Detailed operation commands are as following:

SDRAM Command Truth Table

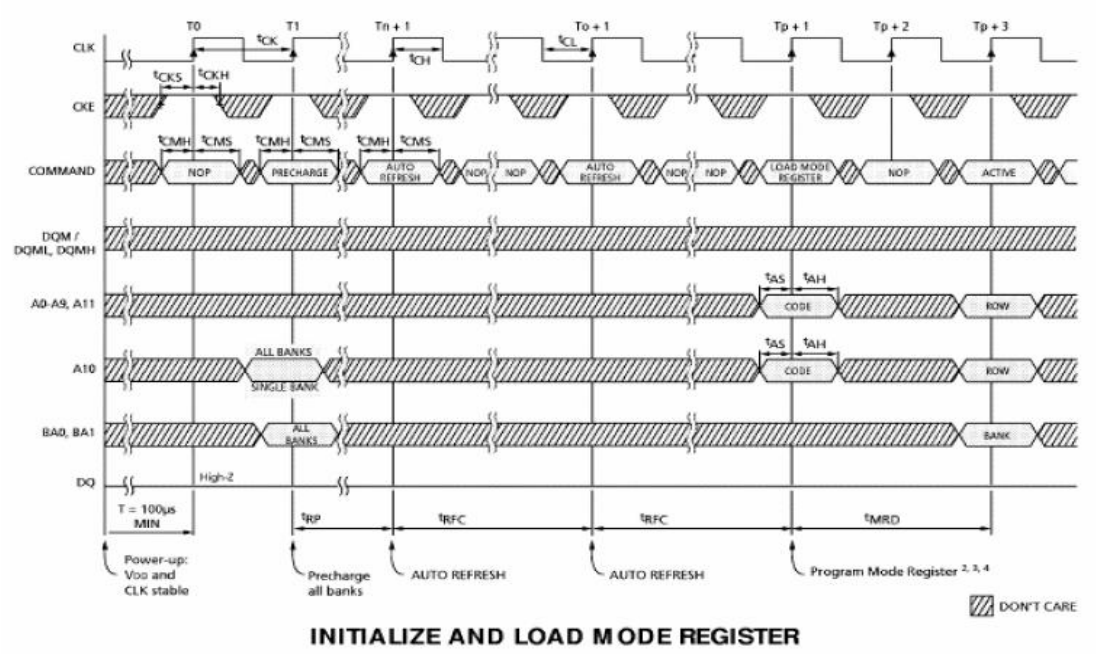
Function	Symbol	CS	RAS	CAS	WE	BA	A10	A[0:9]
Device Deselect	DSEL	H	X	X	X	X	X	X
No Operation	NOP	L	H	H	H	X	X	X
Read	READ	L	H	L	H	V	L	V
Read w/ Auto Precharge	READAP	L	H	L	H	V	H	V
Write	WRITE	L	H	L	L	V	L	V
Write w/ Auto Precharge	WRITEAP	L	H	L	L	V	H	V
Bank Activate	ACT	L	L	H	H	V	V	V
Precharge Selected Bank	PRE	L	L	H	L	V	L	X
Precharge All Banks	PALL	L	L	H	L	X	H	X
Auto Refresh	CBR	L	L	L	H	X	X	X
Load Mode Register	MRS	L	L	L	L	V	V	V

In storage family, SDRAM is a special one. It has large storage, high speed, however, at the same time it has difficulty in storage operation. There are two solutions. One is to control SDRAM’s read/ write timing directly to achieve the data’s storage and read. One is to program a SDRAM controller. Simplify the

SDRAM's read/write to SRAM form. Finish the SDRAM read/write by some commands. This experiment is to finish the SDRAM read/write on the Modelsim development board by both of two ways.

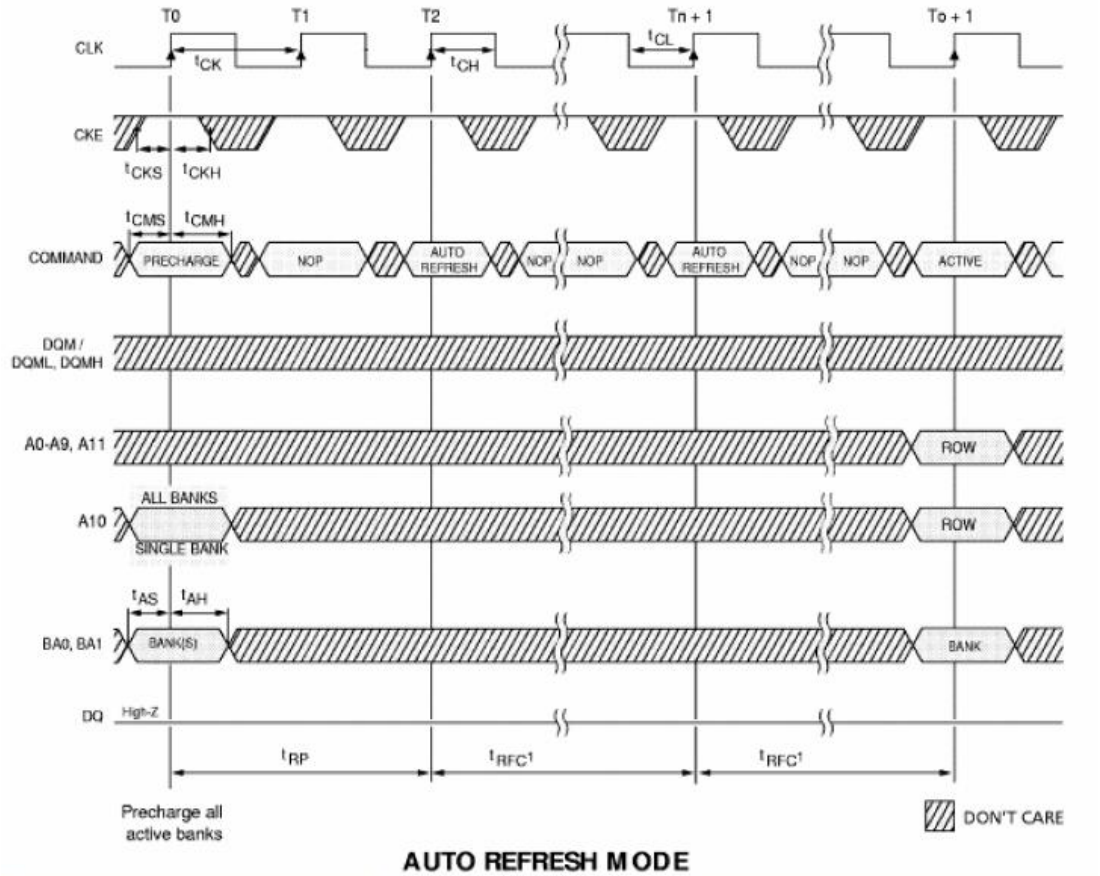
No matter which way to be used, you have to know the read/write timing of SDRAM:

A. Initialize and write register.



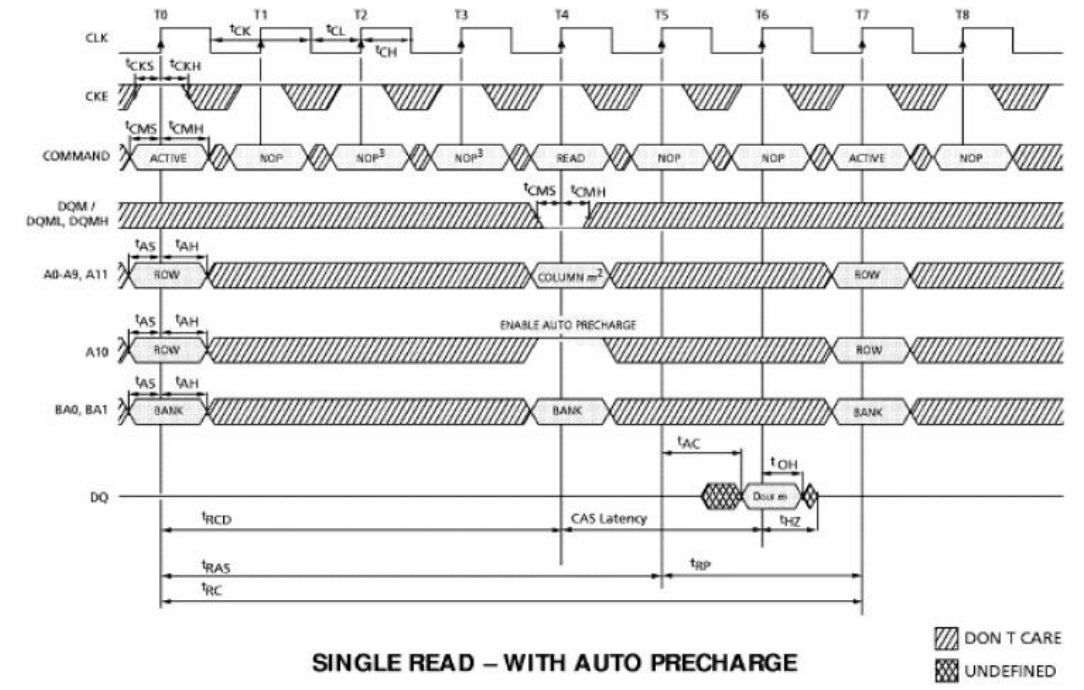
INITIALIZE AND LOAD MODE REGISTER

B. Refresh automatically



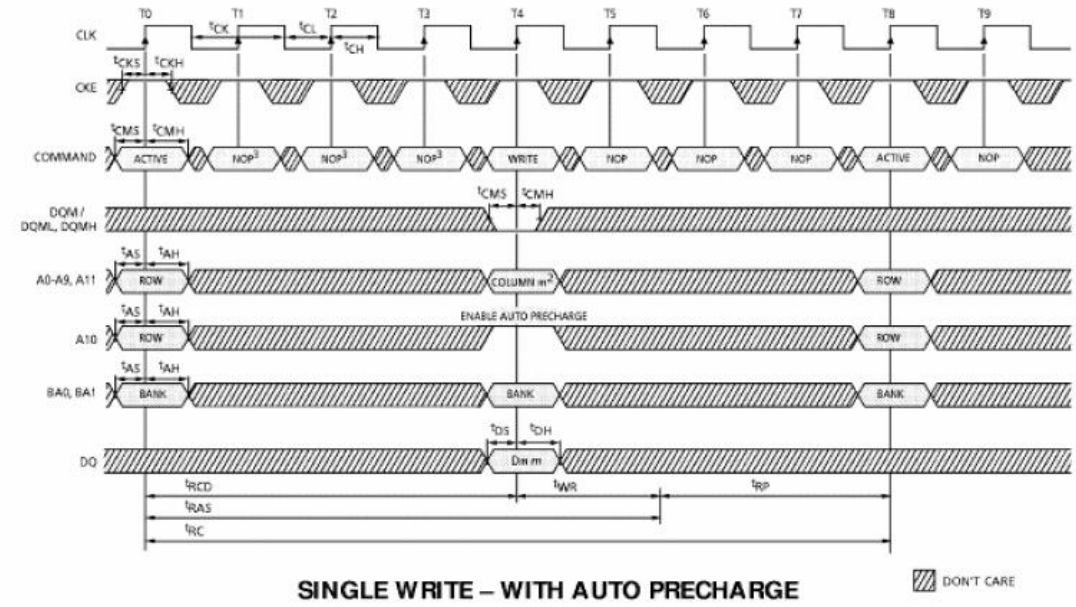
AUTO REFRESH MODE

C. Burst mode read.



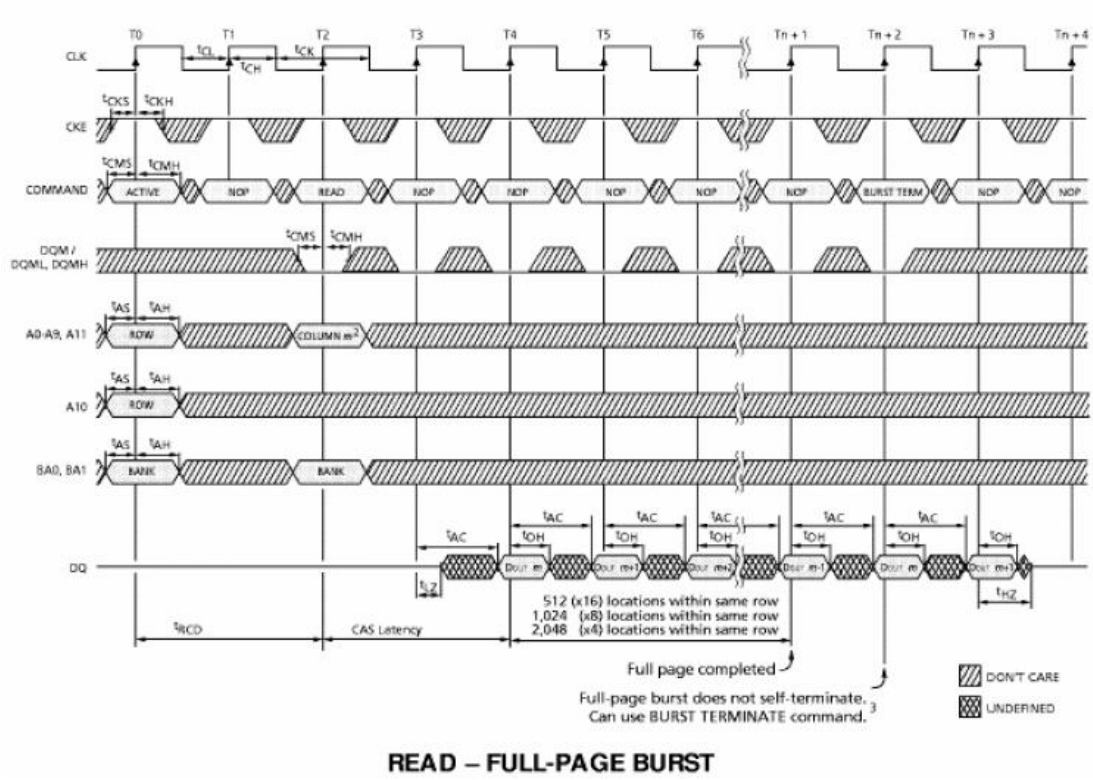
SINGLE READ – WITH AUTO PRECHARGE

D. Burst mode write.



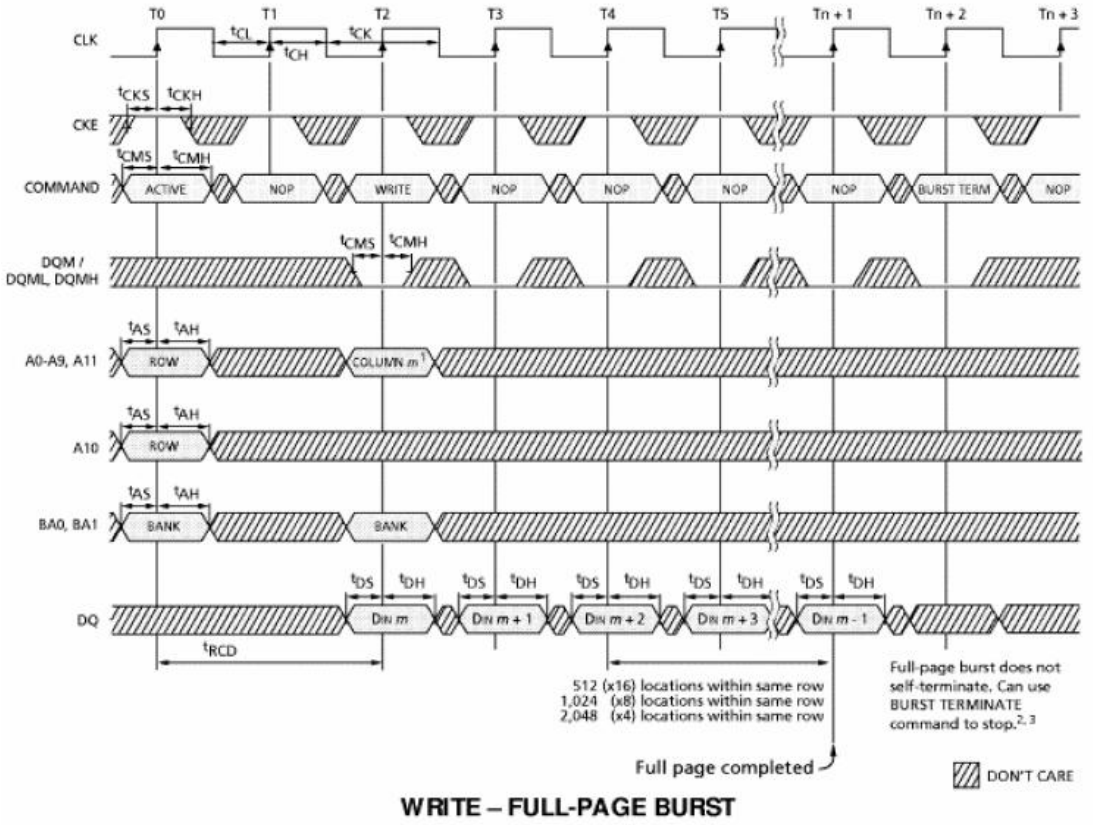
SINGLE WRITE – WITH AUTO PRECHARGE

E. Whole page read



READ – FULL-PAGE BURST

F. Whole page write



WRITE – FULL-PAGE BURST

Upper is the timing pictures of SDRAM achieving each operation. Connect them and you can achieve the SDRAM's initialization and read/write. Detailed SDRAM controlling order can be described by following picture:上

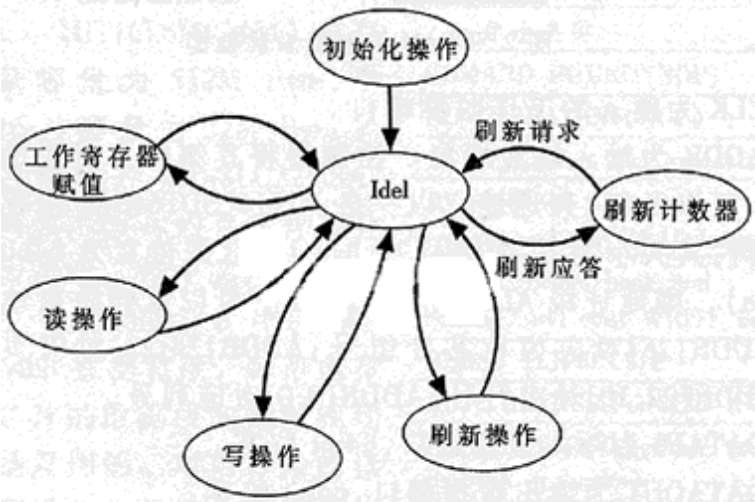
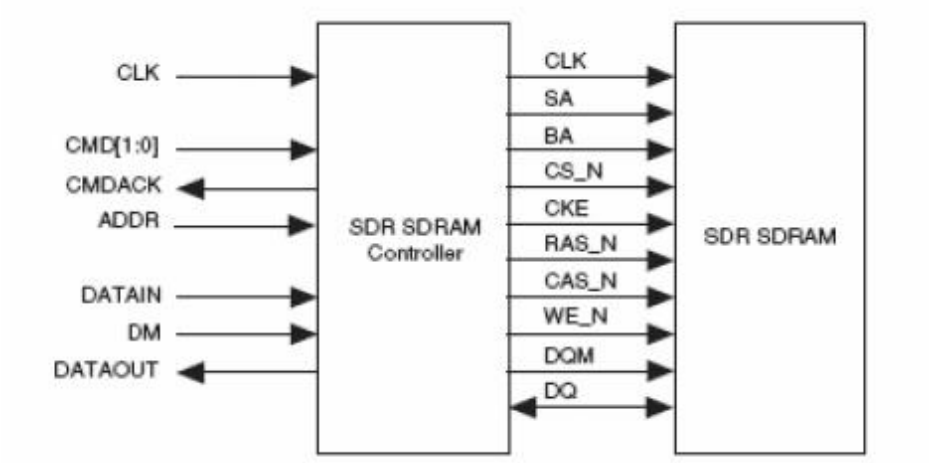
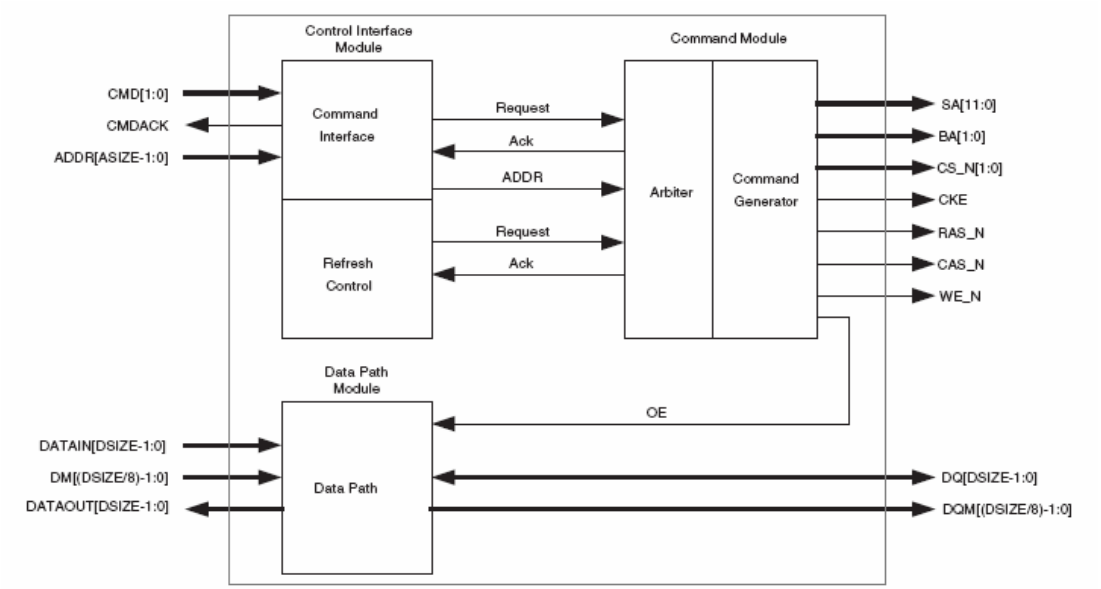


图 3 SDRAM 控制器状态转移图

Here, recommends one SDRAM controller overall design diagram and outside interface signals:



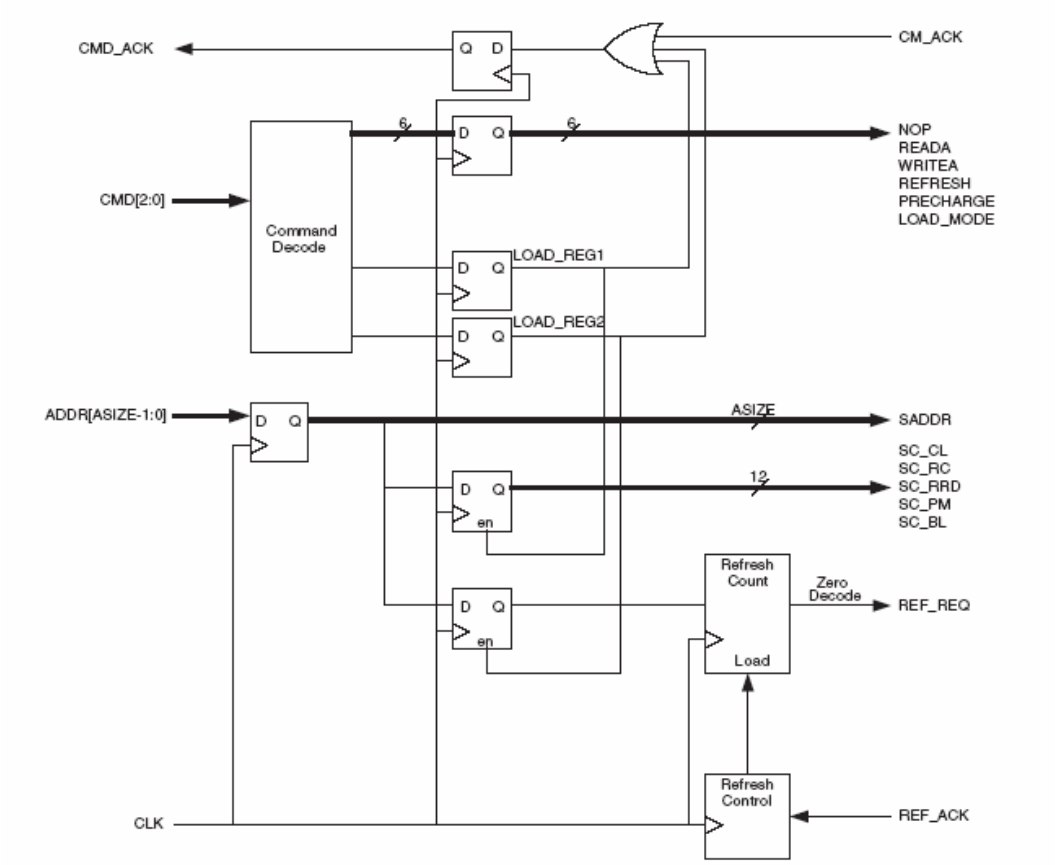
SDRAM controller and external interface schematic diagram is as upper. Signals of the controller's right port are all connected to SDRAM corresponding pins. Do not introduce here. Signals of controller's left port are system controlling port signals connected to FPGA. Among them, CLK is system clock signal, ADDR is the SDRAM address signal gave by system, DATAIN is the data signal used by the system to write into SDRAM, DATAOUT the data signal used by the system to read from SDRAM, CMD [1:0] and CMDACK are system and controller commands interactive signal, and M is data Mask signal.



SDRAM Controller Block Diagram

As upper diagram shows, SDRAM controller includes system controlling interface module, CMD command resolve and responds module and data access module. System controlling interface module is used to receive system controlling signals and generates different CMD command combination. CMD command resolve module is used to receive CMD commands and resolve codes to operation commands, and generate SDRAM operation. Data access module is used to control data's valid input/output. Following are feature details of each module:

(1) System control interface module:



Control Interface Module Block Diagram

This module includes initialization mechanism and system commands analysis mechanism. Initialization mechanism not only has to finish SDRAM's initial configuration, but also do the initial configuration of the controller which keeps the control and the external SDRAM in the same working mode. Here are the processes. The system controlled by the counter is powered to around 200μs. Do SDRAM's initial configuration firstly. One Precharge all bank command finishes all BANK's pre-charge. Then, follow several Refresh commands. Then is mode configuration command LOAD_MODE. Finish the SDRAM woke mode setting. After that, do the controller's initial configuration job. Firstly, send out command LOAD_REG1 to controller loading mode. Then, send out LOAD_REG2 command to load controller's refresh counter value. Complete the controller's initial configuration.

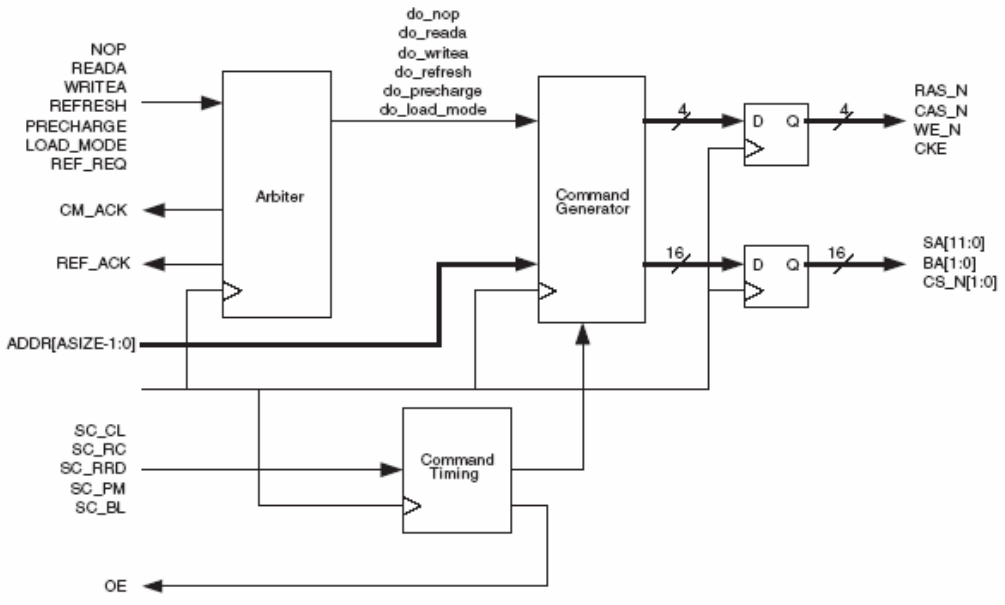
After upper processes, system command analysis mechanism can receiveand analysis system's read/write signal, address and CMDACK signal feedback from the next module. It also generates corresponding CMD command and SADDR address information to CMD command analysis module. Through program setting, achieves determining when read/write Precharge or Refresh CMD command sent out at the certain moment according to the specifications of initial configuration which simplify the system's controlling. Each time when receive CMDACK is 1, means CMD command has been sent and be valid. And have to send out NOP command

(CMD=000)。Attention that, SADDR is time-sharing reused. When initialize load mode, SADDR is used to transform the mode character content defined by user. However, in normal read/write period, SADDR is used as address transform to transit line, row and block address asked by SDRAM. More, system analysis mechanism will feedback SDRAM_FREE and FDATA_ENABLE to system user according to the status of the controller‘s operation of SDRAM.

Detailed CMD commands descriptions are as following:

Command	Abbreviation	RASN	CASN	WEN
No operation	NOP	H	H	H
Active	ACT	L	H	H
Read	RD	H	L	H
Write	WR	H	L	L
Burst terminate	BT	H	H	L
Precharge	PCH	L	H	L
Autorefresh	ARF	L	L	H
Load mode register	LMR	L	L	L

(2) CMD command resolve responds module:



Command Module Block Diagram

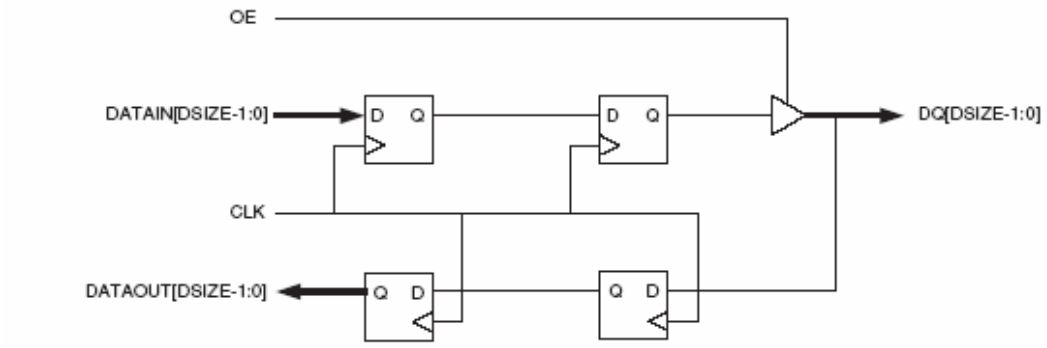
This module judges the CMD command and the result is outputting corresponding operation command signal to command response module. For example, when CMD is 001, it will output do_read signal be 1; and when CMD is 010, it will output do_write signal be 1. At the same moment, only one valid operation command can be output.

Besides, this module includes the mode register which can be used to precharge some certain mode specifications. Mainly, there are three types: First is the SDRAM mode controlling register. In the command LOAD_MODE , send this register into SDRAM mode register to control SDRAM’s working mode. The second is SDRAM controller’s specification register (LOAD_REG1) which makes the SDRAM controller’s working mode match the external

SDRAM devices' working mode. The third is SDRAM refresh cycle controller register. This register precharge the auto-refresh counter value defined by user which is used for the precharge of SDRAM's refresh cycle. The precharge value of three types register above is sent by SADDR when the system controlling interface module is initializing. According to the operation command from CMD command analysis module, this module make the action which meets SDRAM read/write specifications to achieve user's expectations. It gives data choosing signal OE to control data path module (in writing, OE is 1, while in reading OE is 0). Besides, this module processes system non-reuse address ADDR to reuse address SDRAM and sends to SA, BA time-sharingly. In the program, actually, CMD commands of WRITEA and READA imply command ACTIVE. Therefore, when the module receives command do_write or do_read, it will do activation firstly. And then does read or write after CAS delay required by initial configuration. For example, during the initialization, mode requires CAS＝2，BURST LENGTH=PAGE，and after receives do_write=1 from command interface module, it will do activation and gives line address (sends RAS_N=0，CAS_N=1，WE_N=1，SA=raddr). After 2 clock delay, it does write and gives row address (sends RAS_N=1，CAS_N=0，WE_N=0，SA=caddr)。

Besides, after receives each kind of operation commands, this module will responds to CMD command analysis module and cmdack signal is 1. Finally, the responds will be sent to system controlling interface module's CMDACK and signal is 1. If there is no operation command, cmdack=0，CMDACK signal is 0。

(3) Data path module



Data Path Module Block Diagram

This module accepts OE signal's controlling and makes synchronization of data in/out and corresponding operation command. When OE is 1, data can be written to SDRAM by DQ pin. When OE is 0, data can be read from DQ pin of SDRAM.

3.2.3 Experiment content

1．Design to program a SDRAM controller and check it's availability on the

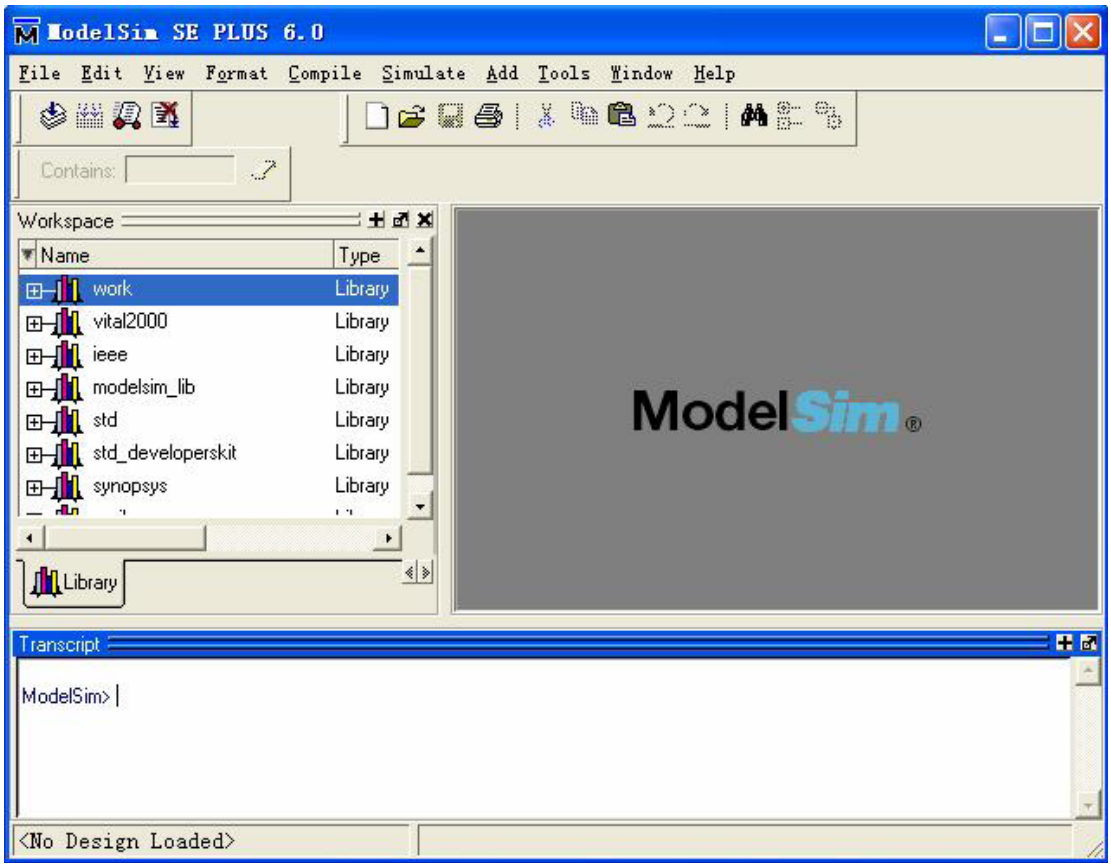
simulation platform ModelSim

2. Write programs to control SDRAM read/write on development board.

3.3.4 Experiment steps

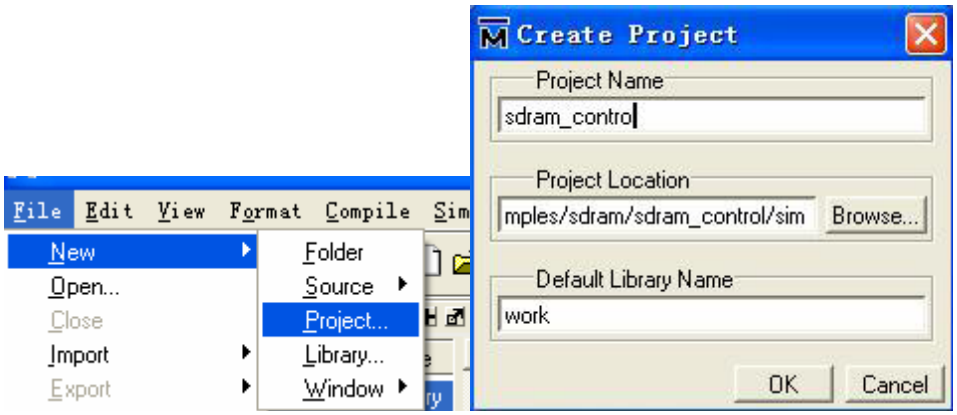
1. ModelSim simulation part

（1） open ModelSim

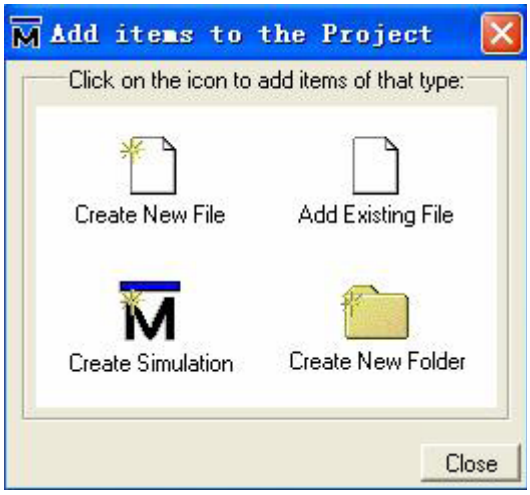


（2） set up a new project and add source file or file wrote by user.

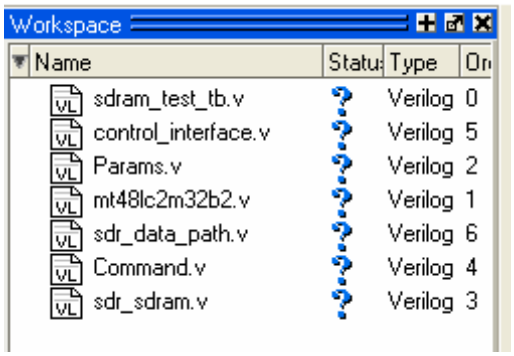
choose “File ⟶ New ⟶ Project…”



Fill project’s name and address in upper dialog. OK.

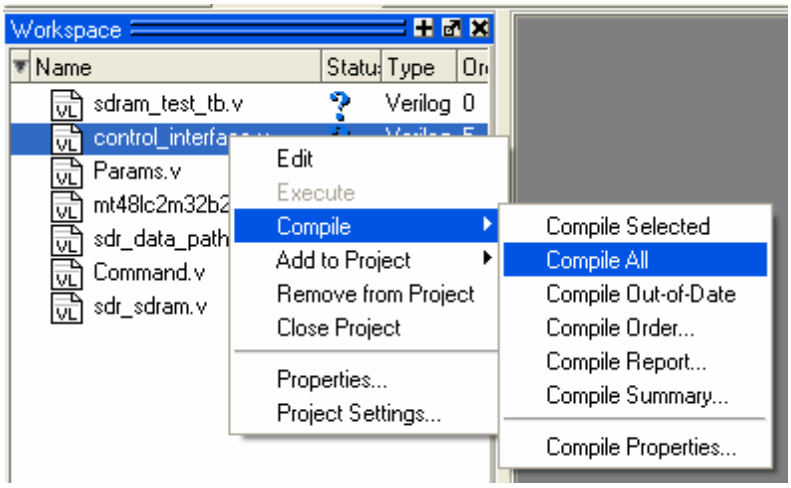


If you want to set up new file, choose Create New File. If you want to add the exits file, choose Add Existing File. Add or write source file, and you can see they are displayed in Workspace:

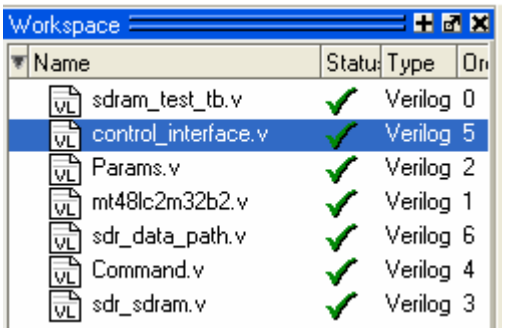


（3） Compile

Right click on any source file, choose “Compile”

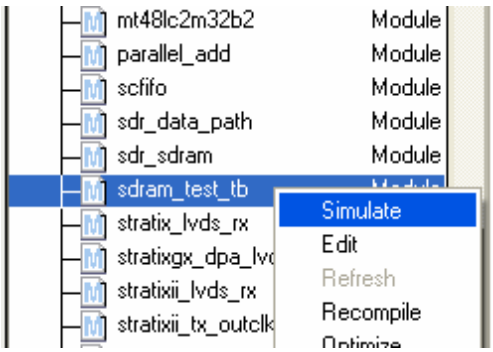


ModelSim will program all files automatically. All mistakes found by it will be list in following dialogue. Here, double click mistake, and ModelSim will open the file which includes it automatically and finds the location of the mistake. If the programming passes, the blue question mark near source file will change to green checkmark.



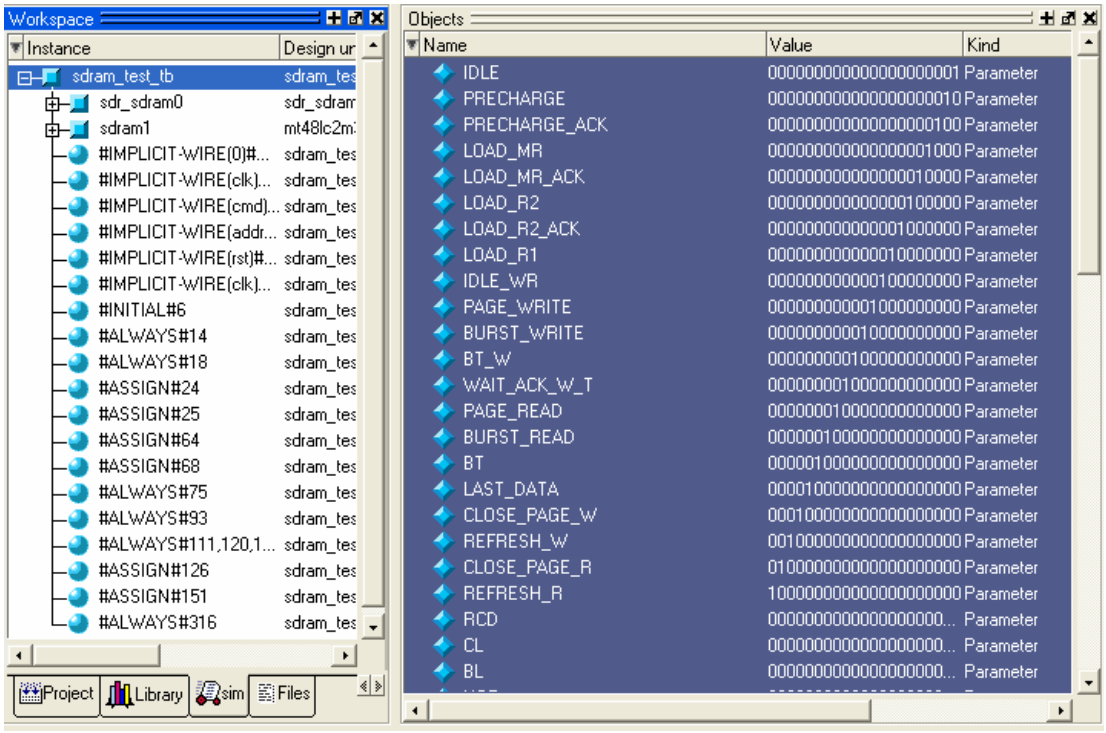
（4） Simulate

Choose check page Library in Workspace. Click the plus mark on the left of Work. In the pop-up submenu, find simulation module. Double click or right click to choose Simulate and ModelSim will run simulation automatically.

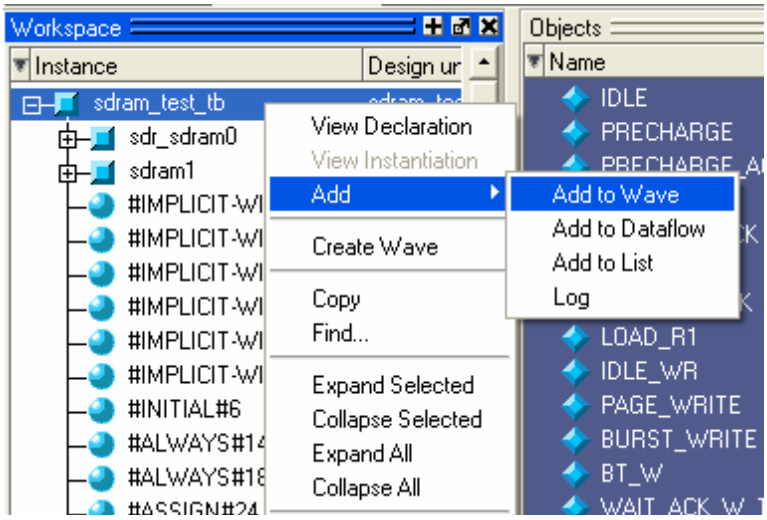


（5） watch waveform

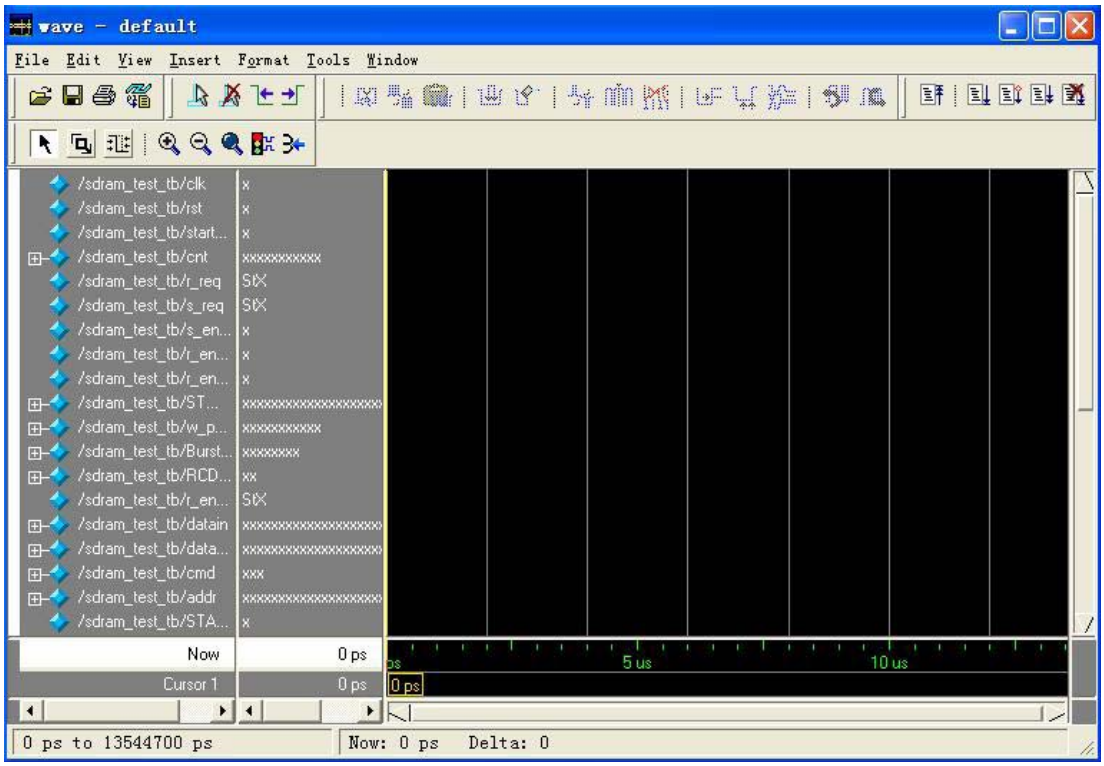
Choose Sims check page in Workspace:



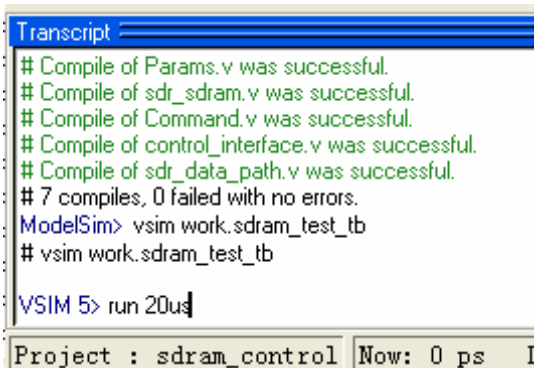
Right click top testing module and choose “Add Add to Wave”.



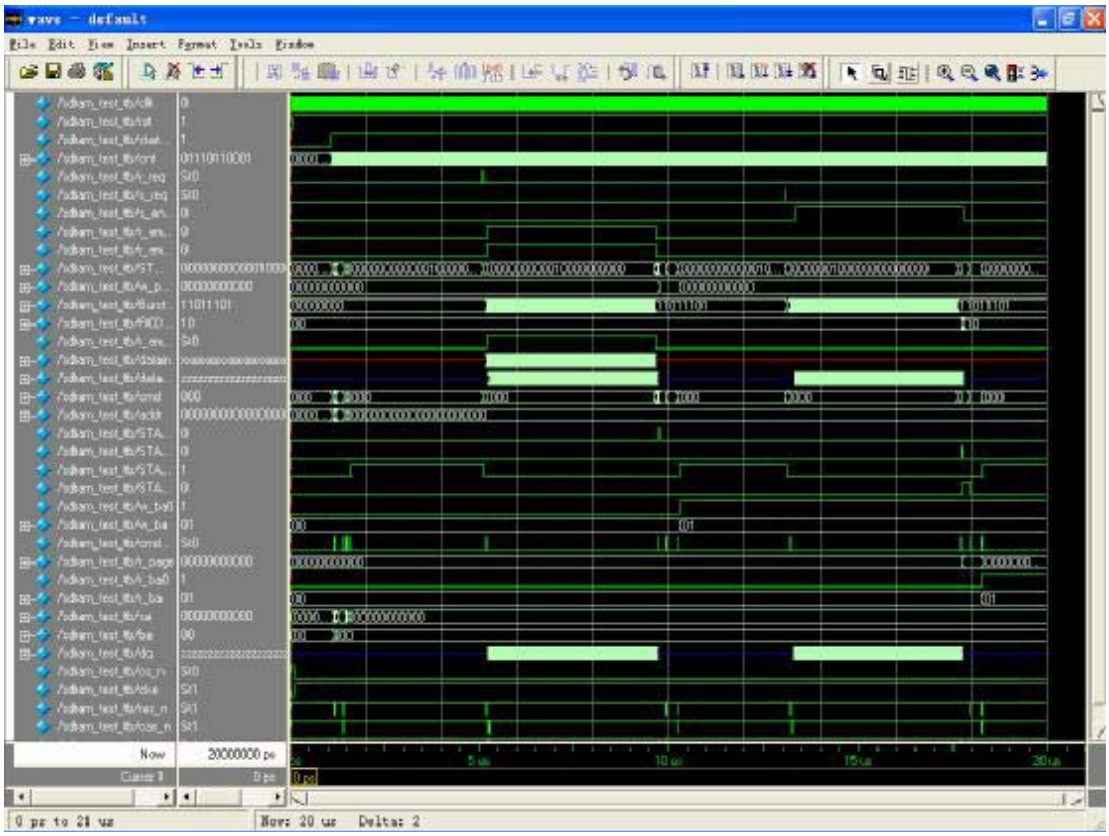
ModelSim will open a waveform simulation interface automatically and add all registers and interfaces of top testing module:



Back to ModelSim的 interface, and type “run 20us”in command input window.



Few seconds after “Enter”, you can see the simulation result as following:



2．Development board testing part

（1） set up a new project.

（2） add source files

（3） integrated, pin defined, and route

（4） download and debugging

3.2.5 Experiment result

LED shinning on the development board means SDRAM’s read dates and writing dates are the same. Working properly.

Experiment15 FLASH read/write control experiment

3.3.1 Experiment purpose

1. Learn the structure and working principle of Flash memory.
2. Master NOR Flash's sequence of reading and writing.
3. Program to control the Flash's reading and writing on development board.

3.3.2 Experiment theory

1、Brief introduction of FLSH

In 1988, Intel first developed the NOR Flash technology, which broke the monopoly of EPROM and EEPROM. And during the next year 1989, Toshiba published the structure of NAND Flash, highlighting lower cost per bit and higher performance, and it can upgrade easily through interface as a disc.

NOR is characterized by its (XIP eXecute In Place), with which the application program can operate in Flash memory directly, rather than read the code into system RAM. With high transmission efficiency, NOR brings high cost benefit in small capacity of 1-4MB. But the low writing and erasing speed greatly affects its performance. NAND structure can provide ultra high cell density; achieve high storage density, and high writing and erasing speed. The difficulty of applying NAND lies in the flash management, and the need of special system interface.

The Am29lv320DB on development board is a NOR Flash memory; it has the following features:

1、 Small size, great capacity; can reach more than 10MB now.

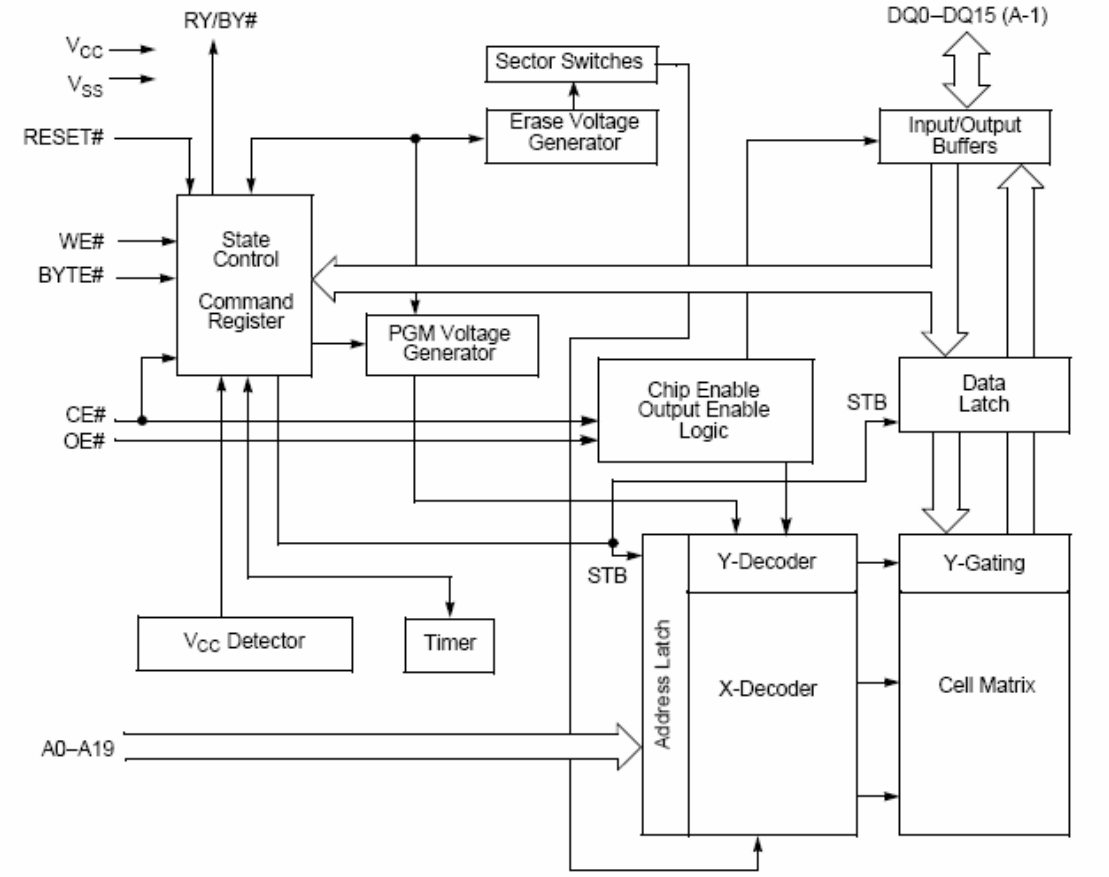
2、 Save the data when power off; the data can be kept for 10-100 years.

3、 With separate address and data bus, it can read the data quickly through bus, so it has the same reading speed as static RAM, and can be used not only as data storage, but also program storage.

4、 Write operation has to be completed in Bytes or words through instruction sequence, each Byte or word needs more than 10μs.

5、 Erasure also has to be operated in blocks through instruction sequence; the usual weight of one block is 64K; the erasure of each block needs more than 10ms.

Internal structure of Flash:



Pin functions table：

Operation	CE#	OE#	WE#	RESET#	Addresses (Note 1)	DQ0–DQ7	DQ8–DQ15	
							BYTE# = V_{IH}	BYTE# = V_{IL}
Read	L	L	H	H	A_{IN}	D_{OUT}	D_{OUT}	DQ8–DQ14 = High-Z, DQ15 = A-1
Write	L	H	L	H	A_{IN}	D_{IN}	D_{IN}	
Standby	V_{CC} ± 0.3 V	X	X	V_{CC} ± 0.3 V	X	High-Z	High-Z	High-Z
Output Disable	L	H	H	H	X	High-Z	High-Z	High-Z
Reset	X	X	X	L	X	High-Z	High-Z	High-Z
Sector Protect (Note 2)	L	H	L	V_{ID}	Sector Address, A6 = L, A1 = H, A0 = L	D_{IN}	X	X
Sector Unprotect (Note 2)	L	H	L	V_{ID}	Sector Address, A6 = H, A1 = H, A0 = L	D_{IN}	X	X
Temporary Sector Unprotect	X	X	X	V_{ID}	A_{IN}	D_{IN}	D_{IN}	High-Z

The read operation of Flash doesn't need to write control word; with only address can it output the data. While the write operation is relatively complicated; you have to write control word first, and the details are as follows:

Command Sequence (Note 1)			Cycles	Bus Cycles (Notes 2–5)											
				First		Second		Third		Fourth		Fifth		Sixth	
				Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data	Addr	Data
Read (Note 6)			1	RA	RD										
Reset (Note 7)			1	XXX	F0										
Autoselect (Note 8)	Manufacturer ID	Word	4	555	AA	2AA	55	555	90	X00	01				
		Byte		AAA		555		AAA							
	Device ID, Top Boot Block	Word	4	555	AA	2AA	55	555	90	X01	22C4				
		Byte		AAA		555		AAA		X02	C4				
	Device ID, Bottom Boot Block	Word	4	555	AA	2AA	55	555	90	X01	2249				
		Byte		AAA		555		AAA		X02	49				
	Sector Protect Verify (Note 9)	Word	4	555	AA	2AA	55	555	90	(SA) X02	XX00				
											XX01				
		Byte		AAA		555		AAA		(SA) X04	00				
											01				
CFI Query (Note 10)		Word	1	55	98										
		Byte		AA											
Program		Word	4	555	AA	2AA	55	555	A0	PA	PD				
		Byte		AAA		555		AAA							
Unlock Bypass		Word	3	555	AA	2AA	55	555	20						
		Byte		AAA		555		AAA							
Unlock Bypass Program (Note 11)			2	XXX	A0	PA	PD								
Unlock Bypass Reset (Note 12)			2	XXX	90	XXX	00								
Chip Erase		Word	6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
		Byte		AAA		555		AAA		AAA		555		AAA	
Sector Erase		Word	6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30
		Byte		AAA		555		AAA		AAA		555			
Erase Suspend (Note 13)			1	XXX	B0										
Erase Resume (Note 14)			1	XXX	30										

Detailed time specifications see chip manual.

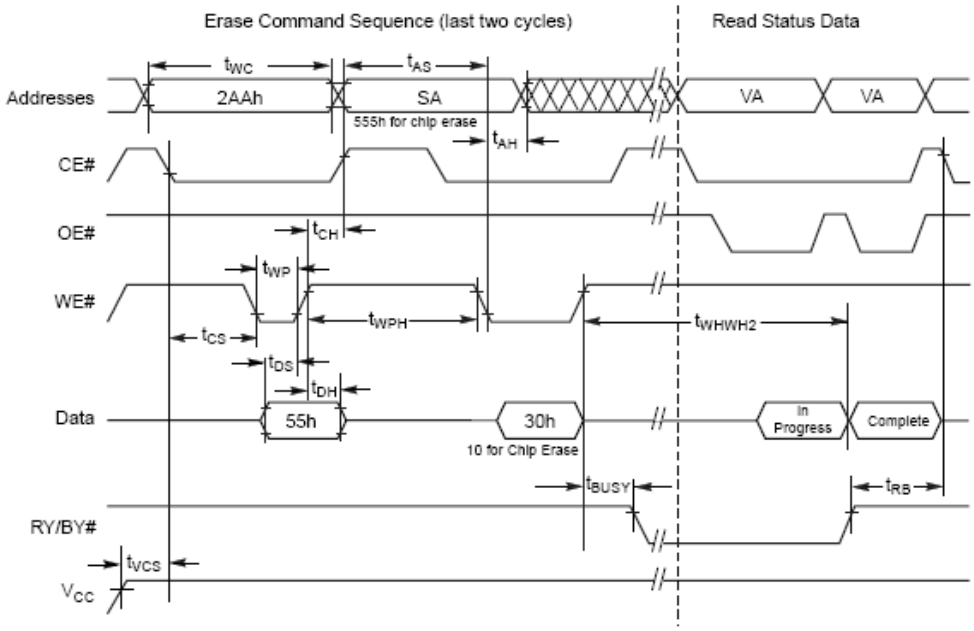
2、Specific examples:

Here we will verify FPGA's controlling of Flash reading and writing through achieving a simple function. The detailed process is as following: after system is powered and reset, through commands, do erase operation to the whole FLASH. Then, write data into FLASH by writing commands. And read the data wrote into the FALSH by reading commands. If the data written in keeps the same as read out one, the light on the development board will be on.

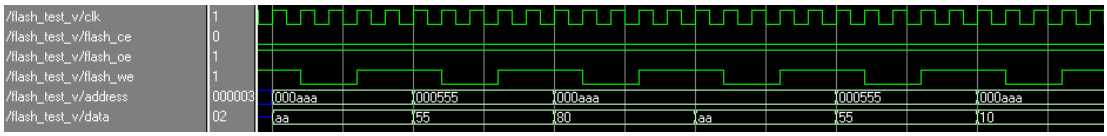
Simulation result:

1）、Erase operation corresponds to the *Chip Erase* order in the chart. Timing requirements:

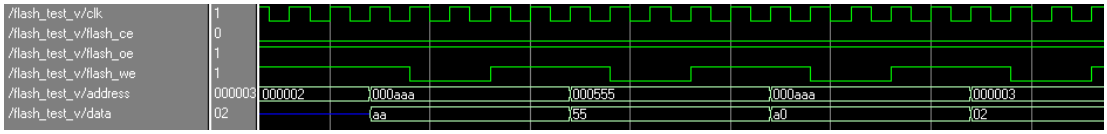
AC CHARACTERISTICS



There are two kinds of erase operations: Sector erase and Chip erase; we adopt chip erase here. It should be noticed that data is latched to the corresponding address during the rising edge of WE.
Simulation result:



As is shown in the table, *address* corresponds to *Addr* in the chart, and *data* refers to *DATA.* According to the table, put the data into the corresponding place. Pay attention to that the data must be stable when the rising edge comes on WE.
2）、Write operation. Corresponding to the Programmed operation in the upper table, FLASH itself has status machine to the FLASH controlling. It will confirm the change of status machine according to the operation code given by user and achieve related operation. In default situation, after system is powered and reset, FLASH is in the status that can read data. Therefore, without any command, you still can read data from FLASH. Only give the FLASH address will be ok. To other operations, you have to give the related operation code. Following is the write operation (Program) simulation result:

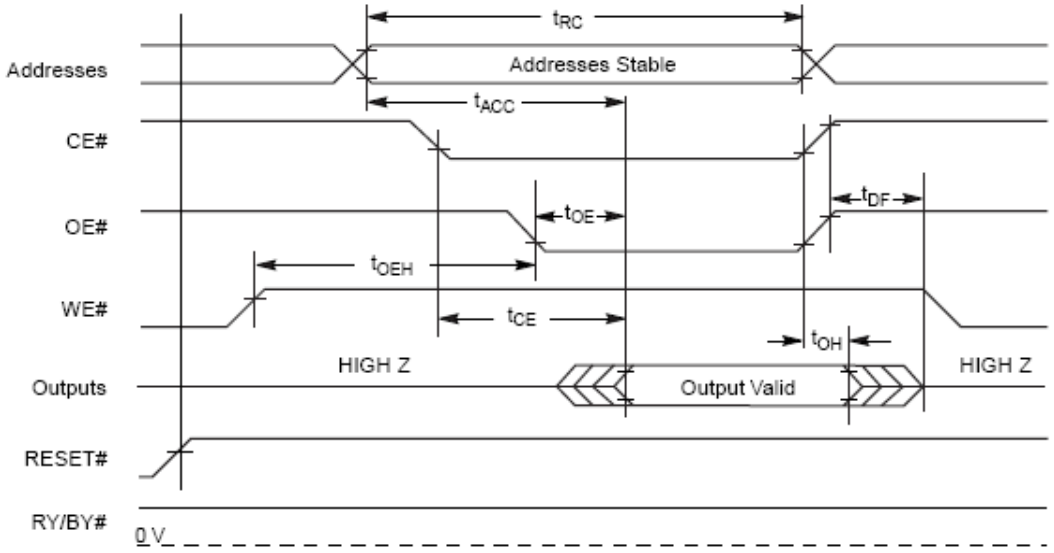


As is shown in the figure, the first three addresses and data is the operation code; the last one is data to write and the corresponding address. Every time the data is written, the state machine of Flash will skip to the state of data access, and we can read the data directly at this time. If there is other

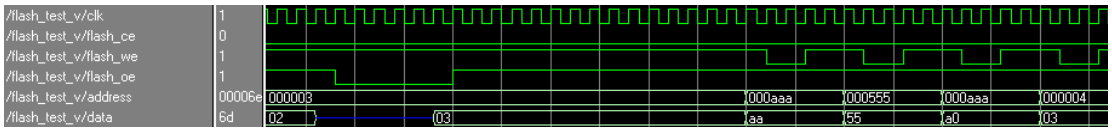
data to write, we have to resend the operation code.

3）、Read operation corresponds to *READ* in the above figure.

Read operation on Flash is very convenient; the address of data to read is the only thing needed. Timing requirements are as follows:



Simulation result:



The figure shows: *address* is the address of read; the simulation imitates that after sending OE signal, there will be a period of time before the output comes out, and the next data will be written in after the previous one is read out.

3.3.3 Experiment content

This experiment is to control the reading and writing of Flash through Verilog, and verify if the read data is the same as write data on development board.

3.3.4 Experiment result

On the development board, the corresponding LED turns on means FLASH works normally.

Experiment16 Data flow control experiment

4.1.1 Experiment purpose

1. Know basic idea of data flow.
2. Know the basic theory of data flow controlling which includes popline, serial/parallel, FIFO, Ping-Pong read/write.
3. Handel the achieving way of each data flow processing way.

4.1.2 Experiment theory

Data flow controlling is the difficult often met in data signal processing. In designing, designed speed is not enough or the resources used in designing is over the maximum content of FPGA will effect on the designed cycle. This is the problem in the relation of speed and area we often mentioned. To solve it, we need useful way to control the data flow. Here we introduce four ways: popline, serial/parallel, FIFO and Ping-Pong read/write. Their features and using ways are as following:

（1） popline

Popline processing is a useful way in high speed designing. If the designed processer flow is separated to some steps and the whole data processing is “single flow” which means no feedback or interaction and the output of previous is the input of the next, we can consider using popline designing way to improve the system working frequency.

The popline design structure diagram is in diagram 3. The basic structure is: connect n operation steps which are suitable separated in series. The most important feature and request of popline operation is that each step of data flow is continuous from time. Suppose each step is passing a D trigger (means use register to hit a beat), the popline operation is similar to a shift register. Data get through the D register in turn and finish each step’s operation.

The key of popline design is the reasonable arrangement of the whole design timing. It asks the reasonable separation of each operation step. If the previous operation time equals to the next one, the design will be the simplest. The previous output directly imports to ten next inputs. If the previous operation time is longer than the next one’s, you need to do the certain buffer of the previous output data and then import to the next inputs. If the previous output time is shorter than the next step’s operation time, you have to copy the logic to divide the flow, or in the previous step use store, after-treaterment.

Or the next step data will be overflow.

The popline processing way is often used in WCDMA designing like RAKE, receiver, searcher, leading to capture and etc. The reason why popline processing has high frequency, is that it copies processing module. It is a concrete embodiment of the thought area for speed.

There are many ways of popline achieving. The most direct way is to use many always blocks in one module. Each always stands for one step of the whole program. In this way, when the trigger edge comes, each always blocks do one operation. And these steps compose a simple popline. Details see the following program:

```
module pipeline(cout,sum,ina,inb,cin,clk);
output[7:0] sum;
output cout;
input[7:0] ina,inb;
input cin,clk;
reg[7:0] tempa,tempb,sum;
reg tempci,firstco,secondco,thirdco,cout;
reg[1:0] firsts,thirda,thirdb;
reg[3:0] seconda,secondb,seconds;
reg[5:0] firsta,firstb,thirds;
always @(posedge clk)
begin
tempa=ina; tempb=inb; tempci=cin; //input data buffer
end
always @(posedge clk)
begin
{firstco,firsts}=tempa[1:0]+tempb[1:0]+tempci;
//first level add (low 2 bytes)
firsta=tempa[7:2]; //data buffer which have participated in caculate
firstb=tempb[7:2];
end
always @(posedge clk)
begin
{secondco,seconds}={firsta[1:0]+firstb[1:0]+firstco,firsts};
//second level add (add 2 to 3 bit)
seconda=firsta[5:2]; //data buffer
secondb=firstb[5:2];
end
always @(posedge clk)
begin
{thirdco,thirds}={seconda[1:0]+secondb[1:0]+secondco,seconds};
//third level add(add 4 to 5 bit)
thirda=seconda[3:2]; //data buffer
thirdb=secondb[3:2];
```

```
end
always @(posedge clk)
begin
{cout,sum}={thirda[1:0]+thirdb[1:0]+thirdco,thirds};
//forth level add(two higher bits add)
end
endmodule
```

Example:

Here we give an example that use detailed application popline to design the adder. It is to achieve the additon of 8 symbol numbers. It uses three-level popline to calculate the addition of 8 9-bit numbers with symbols. The input of each level needs to be expanded first. For example, to the first level popline, data_reg1[0] is the result of data_in0 after is bit expanded:. data_reg1[0] = {data_in0[8],data_in0}. After three-level popline, you can get the final result

（2） serial/parallel, parallel/serial exchange.

Parallel communication: data are transferred in several parallel channels in groups at the same time. For example, several binary bits which compose 1 character codes are transferred in several parallel lines separately. Each bit uses separate line. Parallel communication is very normal especial in two devices which are close to each other. The most common example is the communication between PC and external device, like print cable. Other examples include the communication between CPU, memory and device controller. There is no advantage when the parallel communication is used in long distance connection. First, using several lines in long distance is more expansive than using one. Another problem is about the demand time of bit transform. When the distance is short, bits sent in multi-channel can be received almost in the same time. However, in the long distance, resistances in wire obstruct the transform more or less. Therefore, the bits cannot reach in the same time which brings troubles to the receiving port.

Serial communication: data flow is transferred in one channel by the serial way which means transfer all bits one by one in one line. This way brings additional complexity to sending device and receiving device. The sending way must clear the sending order. For example, when send 8 bits of one character, sending part has to decide which one to be sent first, the high bit or the low one. In the same way, the receiver has to know where to put the first bit in the received purpose byte. If two parties of serial communication can no keep the same in bits order; there will be error in data transmission. 。

As the sending and receiving part only need one transfer channel in serial communication, it is cheaper and easier to achieve, and in long distance connection, it is more realizable, it is the widely using way now a days. However, it sends one bit each time, so the speed is slower.

Serial/parallel exchange is an important skill in FPGA design. It is the often used way in data flow processing. It also reflects the idea of the exchange

between area and speed directly. There are many ways of achieve the serial/parallel exchange. According to the data order and quantity, you can use register, RAM and ect. to achieve. In the Ping-Pong operation diagram before, it used DPRAM to achieve the serial/parallel exchange of data flow. And as used DPRAM, the data buffer can be opened larger. To the lower quantity design, you can use the register. If there are no special requirements, you need to use synchronization timing design to achieve the serial/parallel exchange. For example, in the exchange Serial to Parallel, the high bit is in the front and can be achieved by the following program:

```
prl_temp<={prl_temp,srl_in};
```

Here, prl_temp is the parallel output buffer register, and srl_in is serial data input.

To the serial/parallel exchange has the special order; you can use case statement judgments to achieve. Status device is used for the complex ones. As the serial/parallel change is simple, we do not explain it here.

Parallel/serial exchange is also an important skill of FPGA design. It uses the expense of step to get the advantage in area. It is often used in the FPGA and other devices' interfaces. You can save the resource of FPGA pin that occupied by data transmission. Example, the I2C controller uses parallel/serial exchange. It sends one 8bit data in 8 cycles. Each single cycle (SCL) sends one bit (SDA) only. Detailed achievement can follow the program:

```
if( 3'b000 != n ) n <= n – 3'b001 ;
else n <= 3'b111 ;
assign dout = din[n] ;
```

As mentioned before, the introduction of parallel/serial and serial/parallel exchange can be achieved by dual-port ARM. Through controlling the bit width of read/write dual-port RAM, achieve the exchange. E.G.: write RAM by the bit width of 8bit and read RAM by the bit width of 1bit (here, the read speed has to be 8 times of write speed). At this time, RAM achieves parallel/serial exchange. In opposite, write RAM by the bit width of 1bit and read at 8bit. It finished the serial/parallel exchange module.

（2） FIFO

FIFO is short for First In First Out. It is the first in first out data buffer. The difference between it and the normal memory is that it doesn't have the external read/write address line. It can be used very easily. However, the shortage is that it only can write / read data in order. The data address is finished by internal read/write pointer add 1 automatically. It can do like the normal memory that uses the address line to decide read or write the specified address.

Some important specifications of FIF:

● FIFO width: that is THE WIDTH we often read in English material. It stands for the data bit of one time operation of FIFO.

● **FIFO depth**：THE DEEPTH. It stands for that the FIFO can store several N bit data (if width is N)

● **full mark**：When FIFO is full or to be full, one signal is sent by FIFO status circuit which can stop FIFO written operation from writing to FIFO that may cause overflow.
● **empty mark**：When FIFO is empty or to be empty, one signal is sent by FIFO status circuit which can stop FIFO read operation from reading data of FIFO that may cause underflow.
● **read clock**：read operation follows it. Read when the clock edge comes.
● **write clock**：write operation follows it. Write when the clock edge comes.
● **read pointer**：point to the next address to be read. Add 1 after read automatically.
● **write pointer**：point to the next address to be wrote. Add 1 after write automatically.

（read/write pointer is read/write address. But, this address cannot be chose freely. It is continued）

According to the FIFO working clock area, we can separate FIFO to synchronous FIFO and asynchronous FIFO. Synchronous FIFO means the clock of read and write is the same one. Read/ write operation occurs when clock edge comes. Asynchronous FIFO means the read clock is different from write clock. Read clock is independent from the write one.

The difficulty of FIFO design is how to judge the FIFO empty/full status. To guarantee the correction of data's read and write, and avoid the overflow or underflow, you cannot do write operation when FIFO is full. And you cannot do read operation when the FIFO is empty. How to judge the status is the core of the FIFO design. As the synchronous FIFO is hardly be used, here, we only describe the asynchronous FOFI empty/full mark's generation.

In the design used trigger, you cannot avoid to meet the metastable problem (we won't introduce the metastable here. You may read the related information). Metastable cannot be deleted in the circuit refers to trigger. You can lower the possibility of it as much as you can. One way is to use Gray code. In Gray code, there is only one bit change in the two neighbor symbols (binary codes, in many situations, are many symbols change at the same time). It could avoid the metastable status when the counter and clock are synchronous. The shortage of Gray is that it only can defines the depth of 2^n and cannot defines FIFO depth freely as binary codes because Gray code has to cycle one 2^n, or it isn't the actual Gray code. Another way is to use redundant trigger. Suppose P is the probability of one trigger's metable status, the probability of two serial-level-connected triggers is P^2. However, it will cause the addition of delay. The metastable status will cause FIFO mistakes; the value of read/write clock sampling address is different from the actual one. All these will lead the address mistake of write or read. Consider the delay usage, empty/full mark occurs not only when the FIFO is really empty/full. It may occur before FIFO is empty/full. It is ok if it can guarantee there is no overflow or underflow.

In our actual design, we use FIFO mostly in place that the asynchronous

data is synchronized. For example, when input/output's delay (among chips, PCB lines, some driver interface's delay and etc.)can not be tested, or may changed, you need to set up the synchronization which can use one sync enable or sync signal to make the data be saved by RAM or FIFO. and achieve the data synchronized purpose.

Following is the method of save data in RAM or FIFO. See data clock with data sending provided by the previous level as write signal and write them into RAM or FIFO. Then, use this level's sampling clock (normal is the main clock of data processing) to read them out. The key of this way is the reliability of the writing data to RAM or FIFO. If using sync RAM or FIFO, there must be a guiding signal with data sending which has the fix relationship with data relatively dely. This signal can be the valid guiding to data, and also can be the clock beat by the previous level. To low speed data, you also can use asyn sampling RAM or FIFO. However, we do not suggest you use it.

（4） Ping-Pong structure operation.

"Ping Pong Operation" is the process skill that often is used in data flow controlling.

Ping-Pong operation's processes are as following: input data flow distributes the data flow time-equally to two data buffers by "input data chosen unit". Data buffer module can be any storage module. The often used are dual-port RAM(DPRAM)、single-port RAM(SPRAM)、FIFO and etc. in the first buffer cycle, cache the input data flow to "data buffer module 1"; in the second buffer cycle, cache the input data flow to "data buffer module 2" by switch the "input data chosen unit" and send the data in first cycle cached by "data buffer module 1" to "data flow calculate processing module" to calculate after chose by "input data chosen unit". In the third buffer cycle, through the switch of "input data chosen unit", cache the inept data flow to "data buffer module 1" and at the same time, switch the second cycle data cached by "data buffer module 2" by "input data chosen data" and send to "data flow calculate processing module" to calculate. Do in this cycle.

The most important feature of Ping-Pong operation is that it can send the cached data flow to "data flow calculate processing module" to calculate and process without stop by the switch of "input data chosen unit" and "output data chosen unit" with each other by beat. Take the Ping-Pong module as a whole body. Watch data on the two ports of this module. Input and output data flows are continuous without any stops. So, it is very suitable to the popline processing of data flow. Therefore, Ping-Pong operation is often used in popline calculation to finish the seamless buffer and processing.

The second feature of Ping-Pong operation is that it can save the buffer area. Like in the WCDMA baseband application, one frame is composed by 15 slots. Sometimes, it needs to delay one whole frame for a slot to process. The direct way, is cache this frame data, delay one slot to process. At this time, the length of this buffer is as 1 frame's. Suppose data rate is 3.84Mbps,

the length of 1 frame is 10ms; the length of buffer area is 38400 bits. If using Ping-Pong operation, only need to define two RAM (single port RAM) which can cache 1 slot data. When write data to one RAM, read from the other one and send them to processing unit. At this moment, each RAM's capacity only needs to be 2560 bits. The total of tow RAMs is only 5120 bits.

Besides, skillful use of Ping-Pong operation can make the low speed module to process high speed data flow. Data buffer module uses dual-port RAM and introduced a preprocessing module. This data preprocessing module can do several calculations according to demands, such as in WCDMA designing, it can do the job of input data flow's dispreading, descrambling and remove rotation. Suppose the speed of data flow input from port A is 100Mbps, Ping-Pong buffer cycle is 10ms and following are analysis of data rate at each port.

Speed of data flow input at port A is 100Mbps. In the first buffer cycle 10ms, through "input data chosen unit", pass B1 and reach DPRAM1。B1's data flow rate is 100Mbps, too. DPRAM1 has to write 1MB data in 10ms. In the same, in the second 10 ms, data flow is switched to DPRAM2. □B2's data rate is 100Mbps also. DPRAM2 is written in 1MB data in the second 10 ms. In the third 10ms, data flow is switched to DPRAM1 and DPRAM1 is written in 1Mb data.

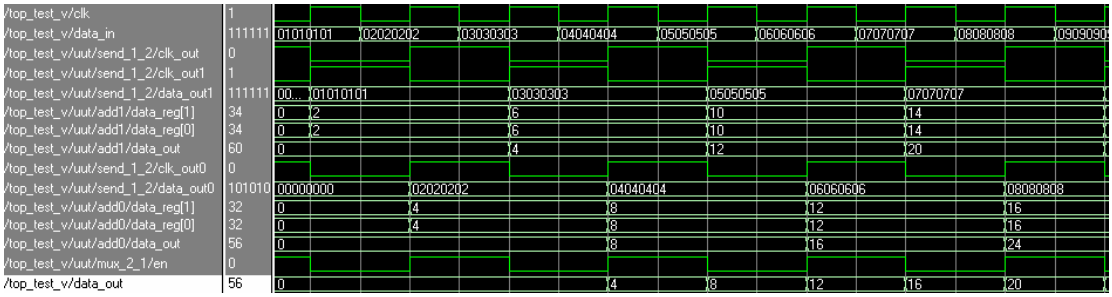
Analysis carefully, you will find, until the third buffer cycle, the time of data left to DPRAM1 read and sent to "data preprocessing module1" is 20ms in total. Some engineers puzzled why the time is 20ms. This came from: first, in the 10ms that write data to DPRAM2 in the second buffer cycle, DPRAM1 can do read operation. Besides, from the 5^{th} ms in the first buffer cycle(the moment that the absolute time is 5ms), DPRAM1 can write data to the address after 500K and at the same time , read data from address0. When reach 10ms, DPRAM1 just finished 1MB data writing and has read 500K data. In this buffer period, DPRAM1 read 5ms. In the third buffer cycle, from 5^{th} ms(the moment that the absolute tie is 5ms), similarly, it can write data to address after 500K and at the same time read data from address0. Read another 5ms. Therefore, before data saved in the first cycle of DPRAM1 is covered completely, DPRAM1 can read 20ms in the most and the data to be read is 1MB. Therefore, the data rate at port C is 1Mb/20ms=50Mbps. And lowest data throughput of "data preprocessing module1" is 50Mbps. Similarly, the lowest data throughput of "data preprocessing module 2" is 50Mbps also. In other words, through Ping-Pong operation, the timing of "data preprocessing module "pressure is reduced. The required data process rate is only half of the input data rate.

The substance of achieve process high speed data by low speed module is: achieve the serial/parallel exchange by the buffer unit DPRAM, and use data preprocessing module 1" and "data preprocessing module 2" to process divided data. It is the withdrawals of the exchange between area and speed.

Example：

Here, use Ping-Pong operation to achieve an added. Its features are as following: input one 32 bits data in each clock and take it as the 4 8-bit data. Calculate the sum of these 4 data. Here use Ping-Pong operation to achieve. In the first clock cycle, send input data to addition 1 unit. Do the addition. In the second clock cycle, write the input data to addition 0 unit (addition unit is still achieved by popline). In the third clock cycle, write data to addition 1 unit again, at the same time, get the calculation result of the one sent to addition 1 unit in the first. In the forth clock cycle, output the calculation result of addition 1 unit and addition 0 unit. In the fifth clock cycle, get calculation result of output addition 0 unit. Cycle in this way, get the continue results at the output port finally.

Its simulation result is as following:



See in the picture：data_in is the 32 bits data output. Each clock inputs one data. data_out1 is the data sent to addition 1 unit after chose by the data chosen unit. The upper data_out is the calculation result of addition 1 unit. data_out2 is the data sent to addition 0 units after chose by the data chosen unit. The lower data_out is the calculation result of the addition 0 unit. The bottom data_out is the final calculation result. We can see that, from data input to the final result, three clock cycles have been passed. In these three cycles, the first cycle sends the data it read to addition 1 unit. The second cycle sends the data it read to addition 0 unit, and the addition 1 unit does the calculation at the same time. The third clock gets the calculation result of addition 1 unit and addition 0 unit does calculation at the same time. And output the calculation result of addition 1 unit until the forth cycle. Cycle in turn and get the continuous output.

4.1.3 Experiment content

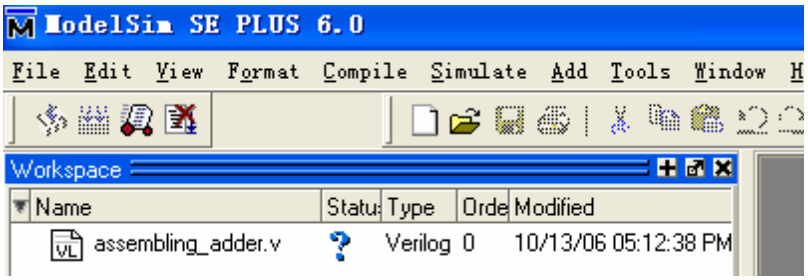
This experiment mainly is used to learn and master four kinds of data flow controlling ways: popline, serial and parallel converter, FIFO and Ping-Pong operation. Program separately to achieve their functions:

- design one popline adder with symbols.
- design a 1:8 serial and parallel converter
- design a interface FIFO(width bit 8，depth bit 128）
- program a adder by Ping-Pong operation.

4.1.4 Experiment steps

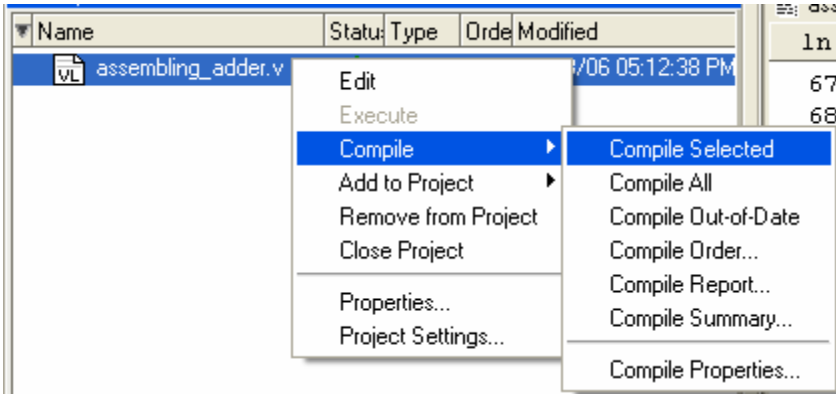
This experiment mainly is doing the simulation in the ModelSim working environment. Detailed steps are as following:

1. Set up project
2. Load files



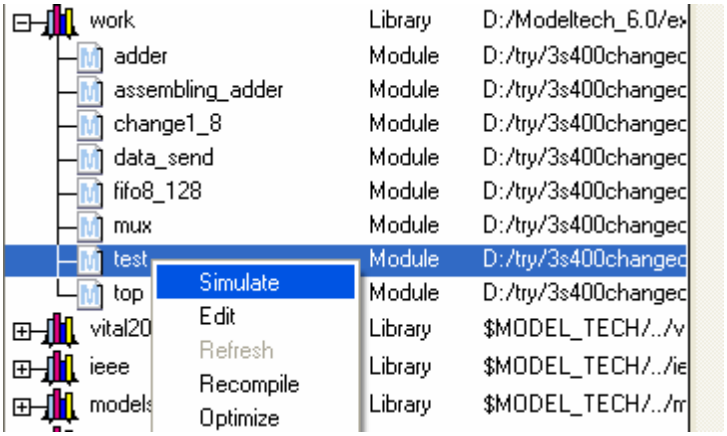
3. Compile

在Project 页



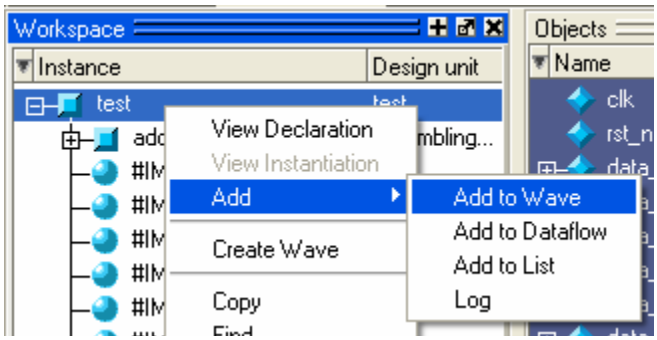
4. Simulate

In page Library



5. Add waveform

In page Sim

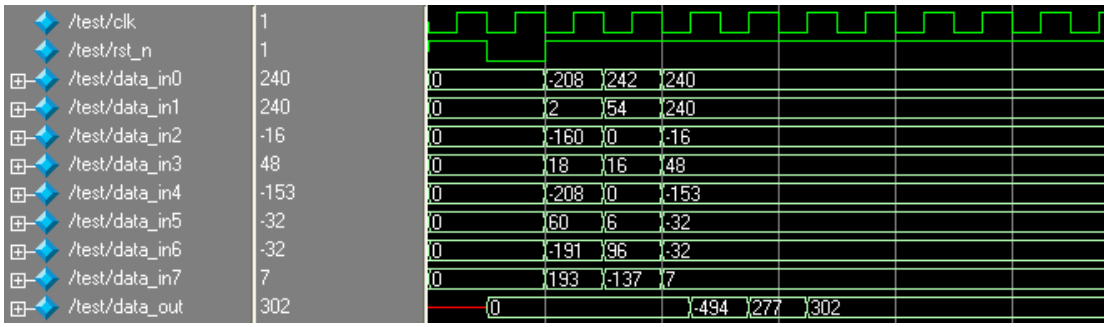


6．simulation

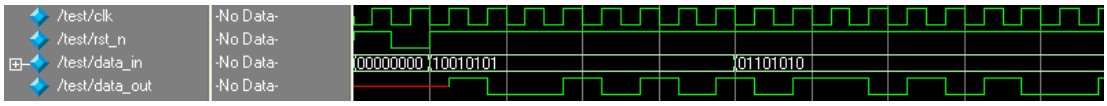
In pop-up waveform window, choose  in the tool bar.

4.1.5 Experiment result

（一）popline adder

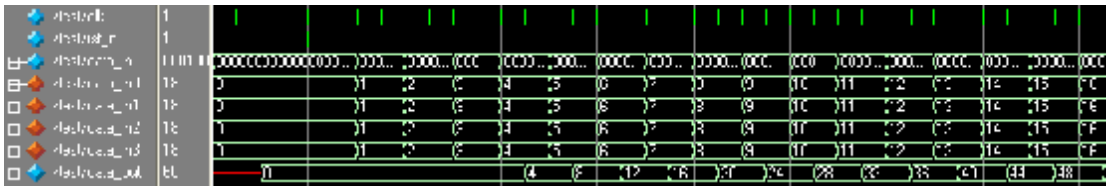


（二）serial and parallel converter



（三）FIFO

（四）Ping-Pong operation



Experiment17 MicroBlaze control LED experiment

5.1.1Experiment purpose

1. Know the structure of Micro Blaze
2. Learn the usage of Platform
3. Know the working principle of OPB bus
4. Master the basic usage of Micro Blaze

5.1.2 Experiment theory

The MicroBlaze32 soft-core processor of Xilinx Company is the fastest soft processing solution in this field. The standard peripheral set, which supports Core Connect bus, provides the designers with compatibility and reuse ability. Running in the 150MHz clock, Micro Blaze processor can provide 125D-MIPS performance. It's very suitable for designing complicated systems aiming at network, telecommunication, data communication, embedded type, and consumer market.

（1）Micro Blaze structure

Micro Blaze is the microprocessor IP core based on FPGA of Xilinx Company. Together with other peripheral IP core, it can design the programmable SOPC. Micro Blaze processor is an independent 32-bit instruction and data bus, adopting RISC framework and Harvard Architecture. It can execute the programs stored in on-chip memory and external memory at full speed and visit the data there.

● Internal structure

Inside Micro Blaze, there are 32 32-bit purpose registers and 2 32-bit special function registers —— PC pointer and MSR state EFLAGS. In order to improve performance, Micro Blaze also has instructions and data cache. All the instructions are 32 bits in length; there are 3 operands and 2 addressing odes. Instructions can be divided into logical operation, arithmetic operation, branch, memory read/write, special instructions, etc. The assembly line of instruction execution is parallel line, which can be classified into three categories: fetching, decoding, and execution.

● Core Connect technology

Core Connect is the on-chip bus communication chain developed by IBM, which makes it possible to connect several source chips nuclear into a complete new chip. Core Connect technology makes integration much easier and enables the reuse of processor, system, and peripheral core in the

standard production platform design, achieving higher overall system performance.

As is shown below, Core Connect bus architecture includes PLB, OPB, 1 bus bridge, 2 arbiters, and 1 DCR bus. Xilinx will provide all the embedded processor users with IBM Core Connect permission, since it is the basis of all the Xilinx embedded processors' design. Micro Blaze processor uses the same bus as IBM PowerPC, which serves as peripheral. Although Micro Blaze soft processor is totally independent of Power PC, it enables the designers to choose the method of operation on chip, including embedded PowerPC, and share its peripheral.

● Core Connect architecture——OPB

The kernel can visit low speed and low performance system resources through OPB. OPB is a completely sync bus; its function lies in a single bus layer, rather than connect to the processor core directly. The OPB interface provides separated 32-bit address bus and 32-bit data bus. With the help of *PLB to OPB* bridge, the processor can visit peripheral through OPB, and in turn, as OPB bus controller, the peripheral can visit the storage through PLB with the help of *OPB to PLB* bridge.

● Core Connect architecture——Processor Local Bus（PLB）

PLB interface provides commands and data side with independent 32-bit address and 64-bit data bus. The A device embedded with PLB interface can be connected with the B device to read and write data through PLB signal, which is supported by PLB. Each A device is linked to PLB through independent address bus, read data bus, and write data bus. While PLB B device is linked to PLB through shared but separated address bus, read data bus, and write data bus. Therefore, to each data bus, there is complicated transmission control and status signal.

In order to permit the A device to get the bus own ship through competition, there is a central judgment institution authorizing the visit to PLB. And this judgment institution is of enough flexibility to provide all kinds of priority.

● Core Connect architecture——Device Control Register Bus（DCR）

Device control Register Bus（DCR） is designed for the data transmission between CPU general purpose register （GPR）and the slave logical device control register of DCR.

● （2）Development of Micro Blaze

Application EDK (Embedded Development Kits) can develop Micro Blaze IP Core and build embedded system. The tool kit integrated the hardware platform generator, the software platform generator, the simulation model generator, software compiler, software debugging aids, etc; EDK provides the integrated development environment XPS (Xilinx platform studio) so as to use all the system tools to finish the whole procedure of embedded system development. EDK is also embedded with some peripheral interface IP cores,

such as LMB,OPB bus interface, UART, interrupt controller, timer and so on, with which we can build a relatively complete embedded micro processor system.

The embedded system designed on FPGA can be classified into 5 grades. Among these, the IP core can be developed on the lowest layer of hardware resource, or set up the hardware development part of embedded system by using developed IP core; The software development includes the development of IP core device driver, application interface (API), and application layer (algorithm).

Build the basic embedded system by utilizing Micro Blaze, which can be connected with various kinds of peripheral IP core through standard bus interface- LMB bus and OPB bus IP core.

Each IP core provided by EDK has corresponding device driver and application interface, so the users can program their own application software and algorithm routine by simply using the related function library. As for the IP cores developed by users, they have to program corresponding drivers and interface function themselves.

（3） Application of Micro Blaze

Usually the *Micro processor* + *coprocessor* structure is adopted in software radio system: the micro processor mainly completes the work of system communication and baseband processing by DSP, and the coprocessor mainly complete the bottom algorithm of synchronization and preprocessing on FPGA. It is relatively simple to adopt baseband processing algorithm in this topic. Replace the DSP with application software processor so that the whole system can be designed within one piece of FPGA, which can simplify system structure and improve system’s overall performance.

For example, There are two tasks for the system on FPGA——send and receive data. As for sending, FPGA first complete the initialization of hardware algorithm, then receive serial data and save it into two-part SRAM; System hardware algorithm part does the baseband processing to these data and sends the result to DA converter. As for receiving, after receiving the data from DA converter, FPGA will do baseband processing to the data and save it into two-part SRAM; after that, send these data back to A device.

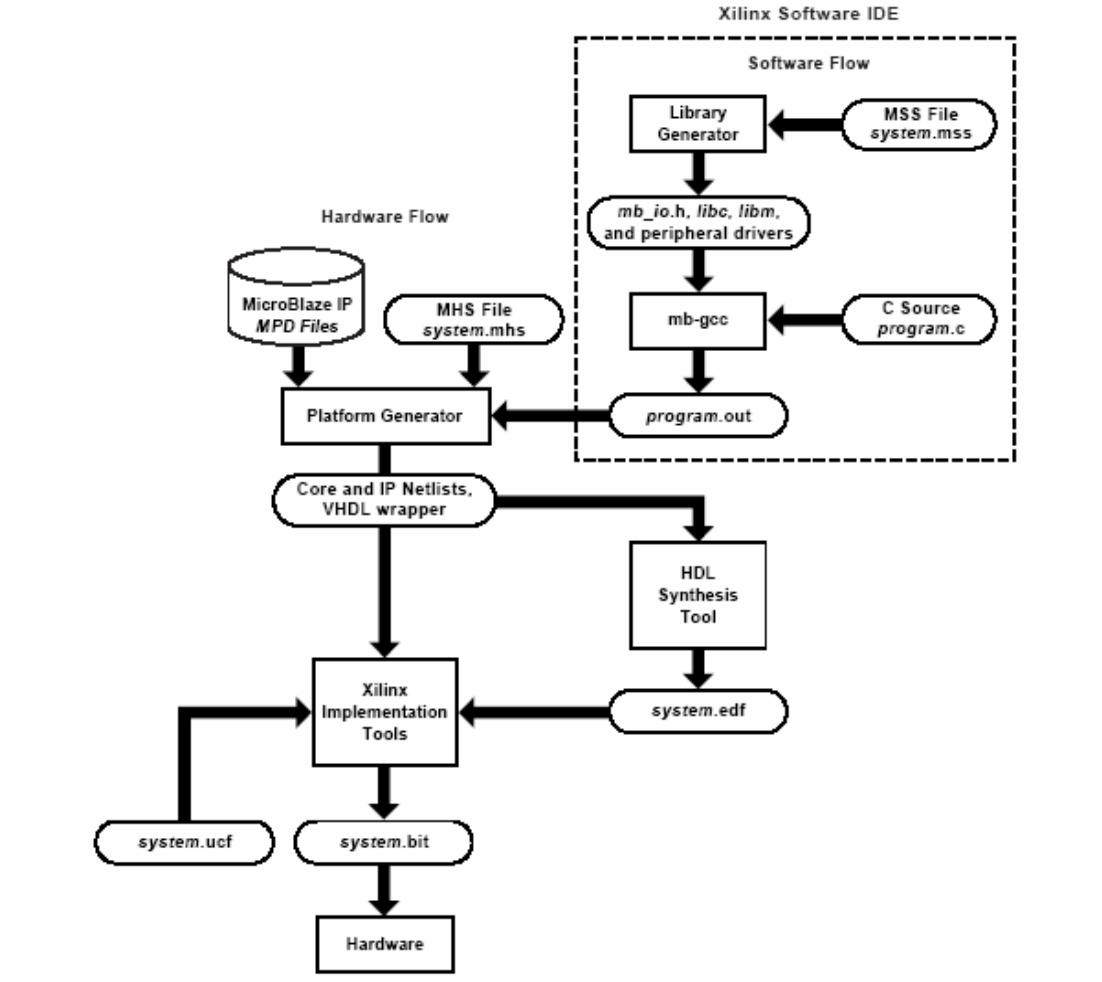
The system hardware can be designed under XPS integrated development environment of EDK development kits. Add IP core, connect system, and set each parameter in this environment. Since the hardware algorithm module in the system is not standard module, the project should be set in sub module way. Use the platform generator, according to MHS document, to generate NGC document of embedded system sub module. Afterwards, in the ISE design environment, connect the NGC documents with hardware algorithm module through GPIO port exteriorly so as to constitute the hardware module of the whole application system.

Each peripheral IP module on EDK has its own software function library. Add the needed header files of peripheral function library into program by

Libgen tools so that the peripherals can be operated and controlled through calling these functions.

Designing the embedded system by adopting FPGA and Micro Blaze realizes the function of multi-piece ASIC and narrows the receiver volume greatly, making it easy for the system to achieve miniaturization and integration. Utilizing hardware to achieve the capture and Frequency hopping synchronous algorithm can accelerate the capture and tracking speed. The experiment result proves the design of FPGA system feasible. With high-capacity SDRAM disposed in system, high speed communication interface such as Ethernet and USB added, and real-time operation system run on the processor, it can be built into a relatively complete embedded system based on FPGA, which is quite promising in fields such as network, communication and consumption.

The design procedure of software and corresponding hardware development on FPGA of Xilinx Company is as follows:



5.1.3 Experiment content

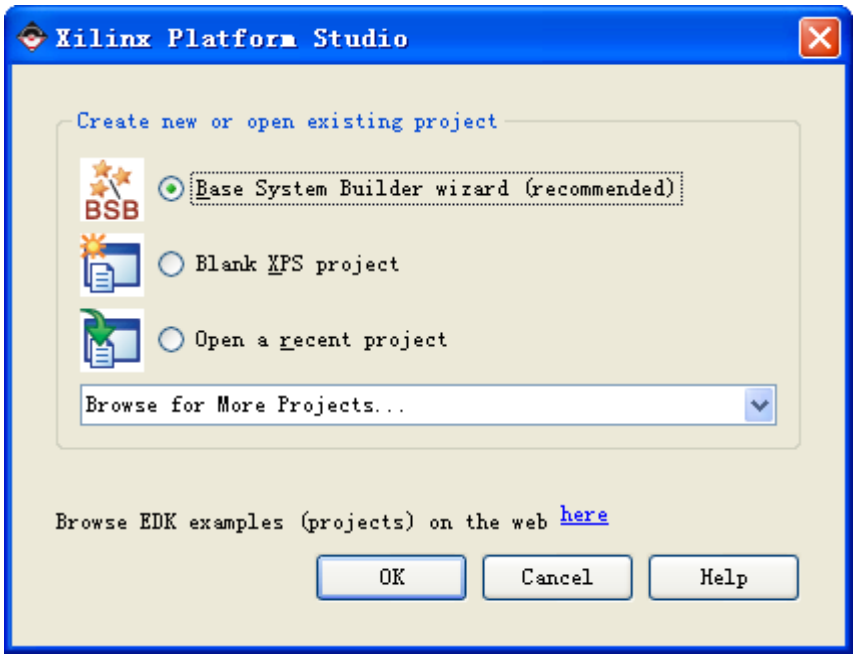
This experiment controls 8LEDs on development board through using processer MicroBlaze.

5.1.4 Experiment steps

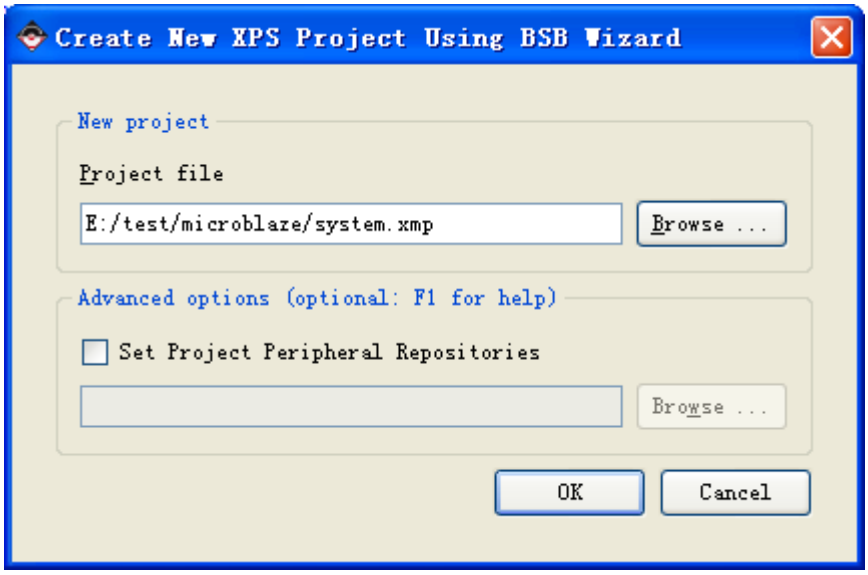
1. Start Xilinx Platform Studio

Start ⟶ All program ⟶ Xilinx Platform Studio 11 ⟶ Xilinx Platform Studio

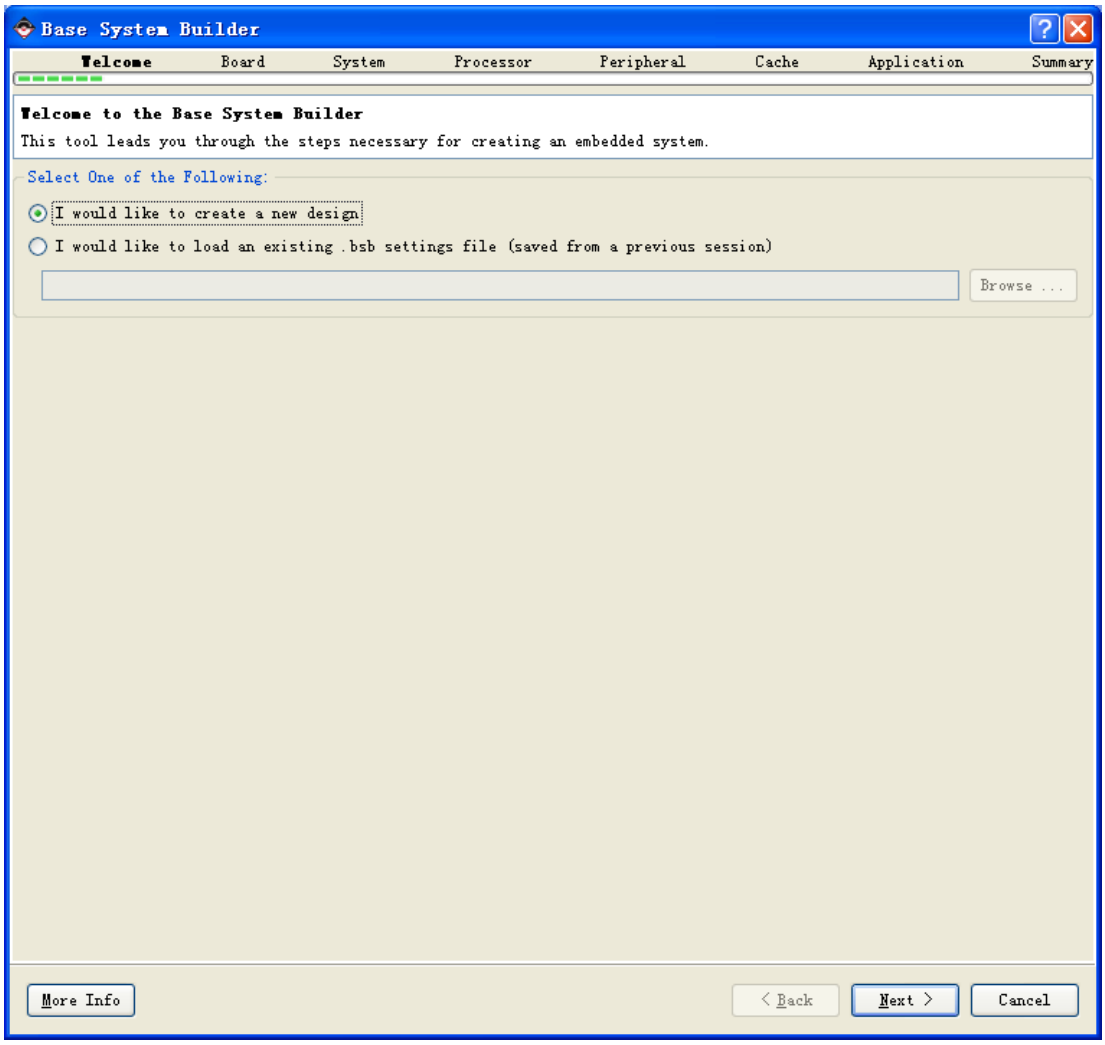
2. Set up a system



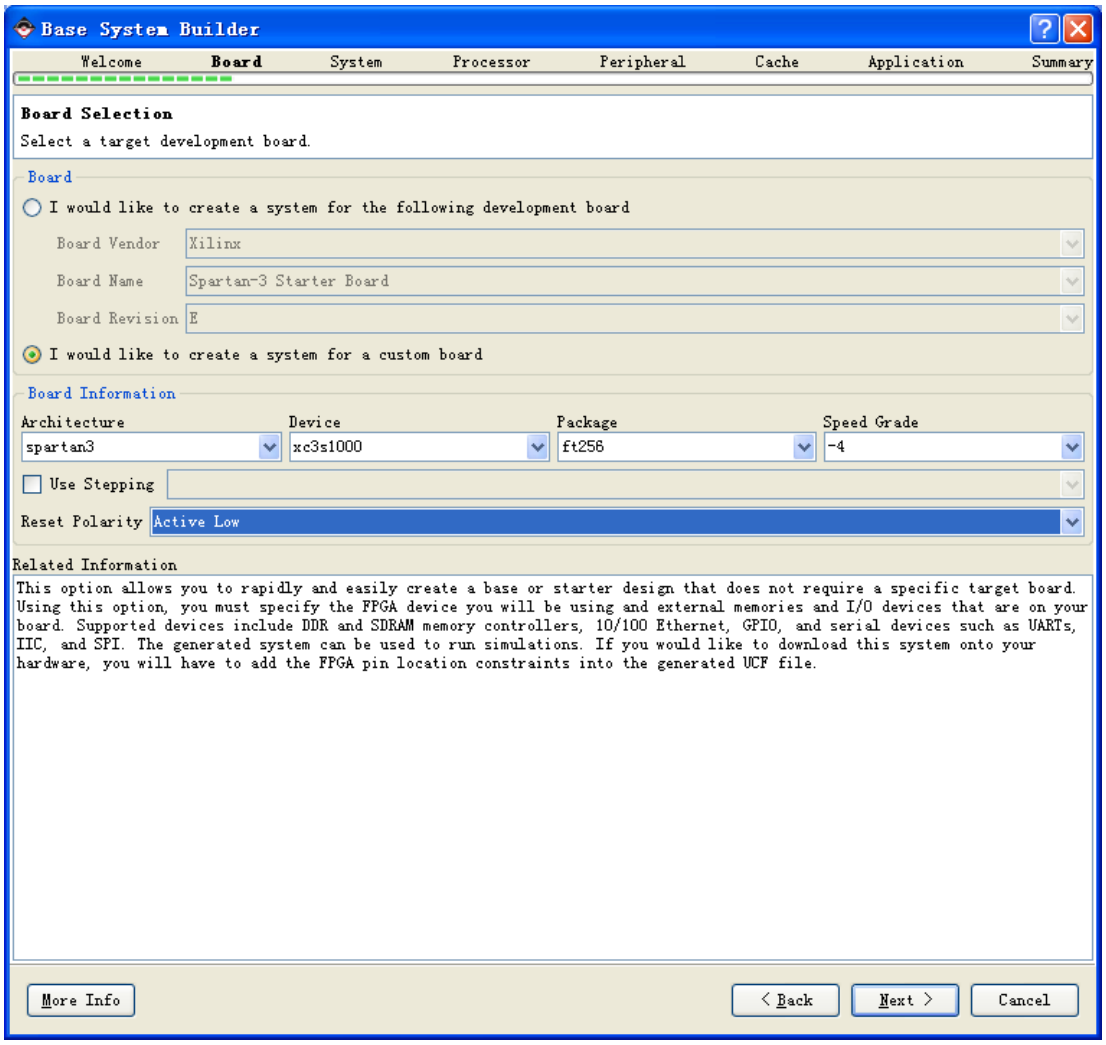
Choose OK；



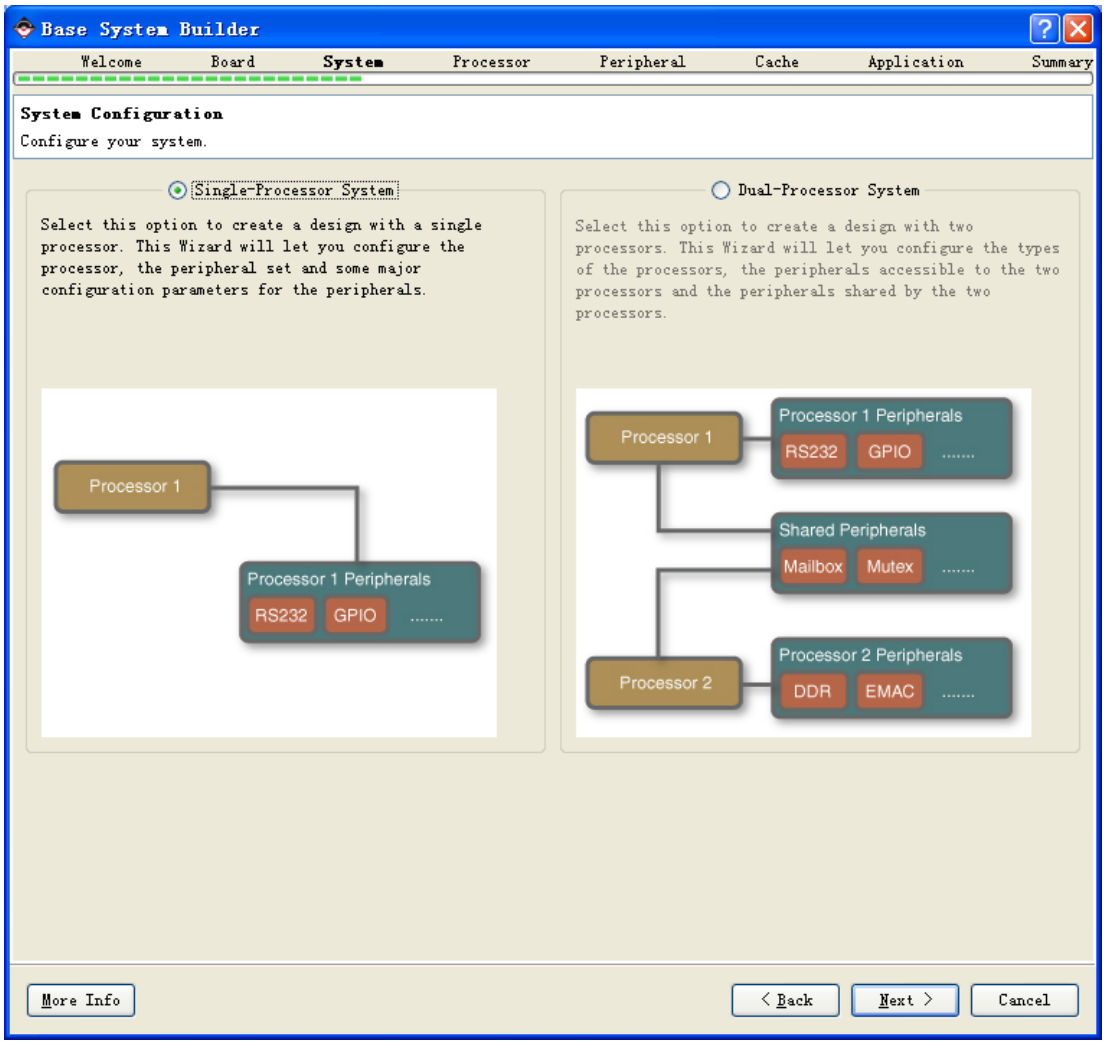
Choose the path of system to be generated in Project File. Choose OK；



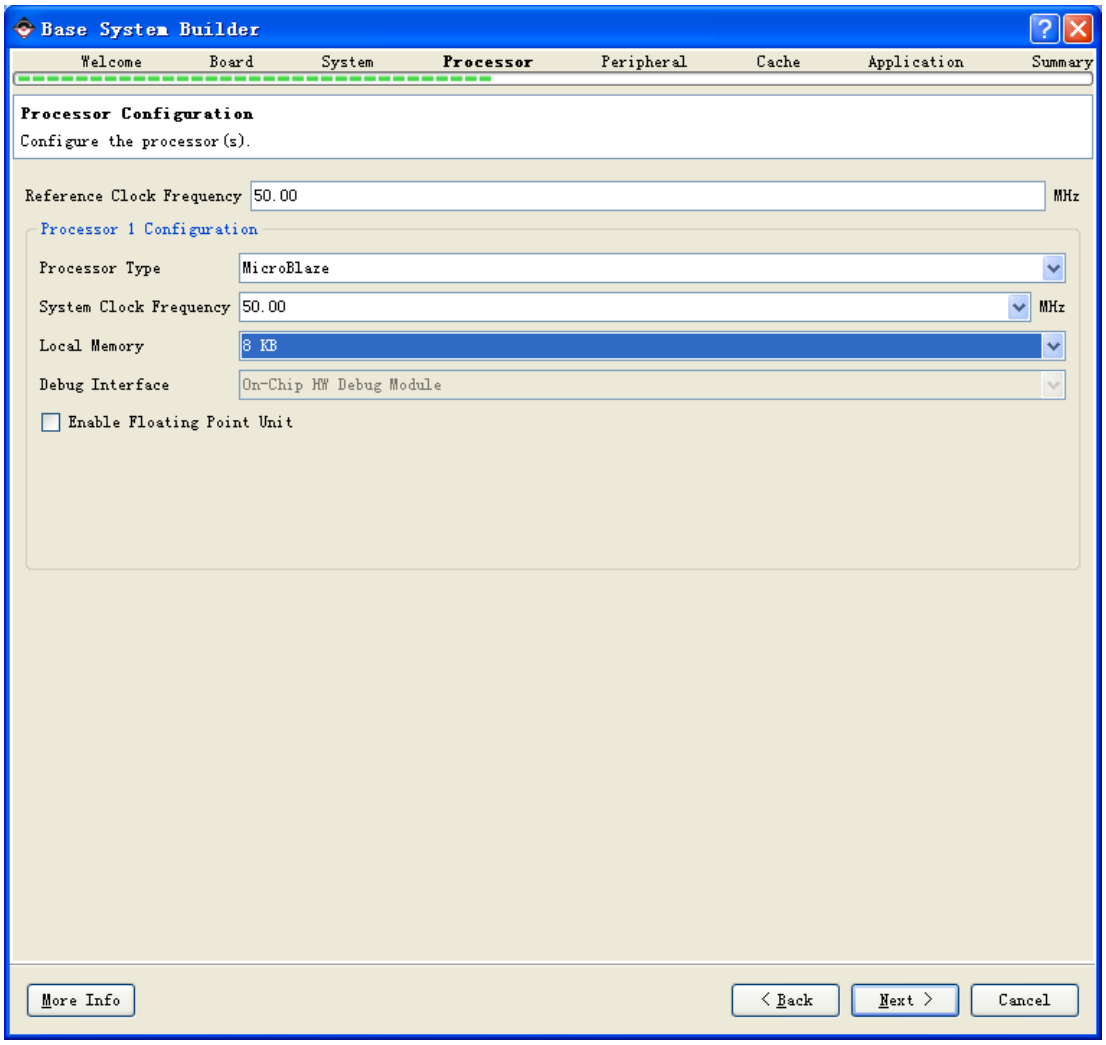
Here, choose set a new design, Next>



Choose I would like to create a system for a custom board and development board's FPGA type: spartan3 xc3s1000 ft256 -4 , Reset. Choose Active LOW。Next>



Choose Single-Processor System and click Next>

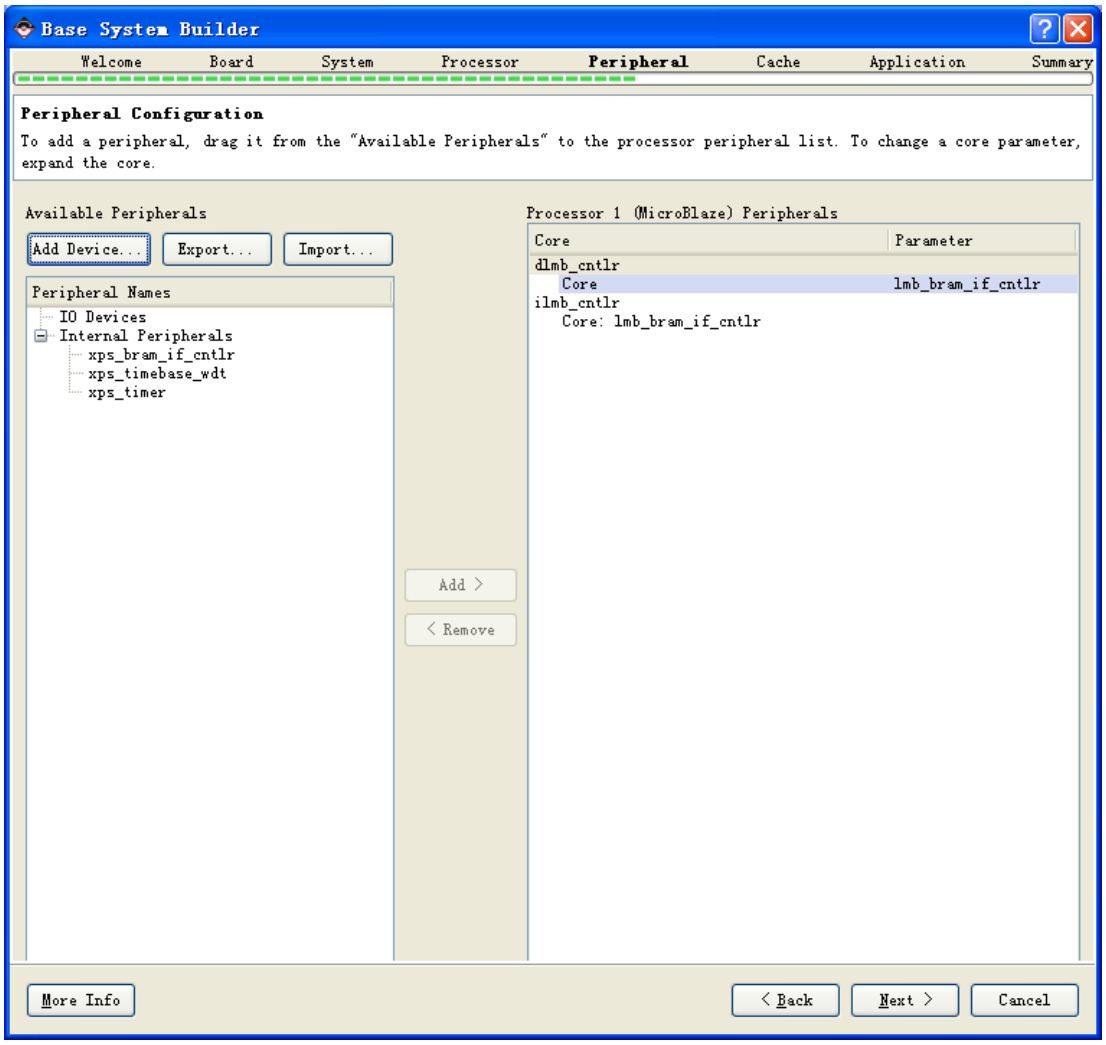


There is only one 50MHz clock on the development board. Therefore, here fill 50.00

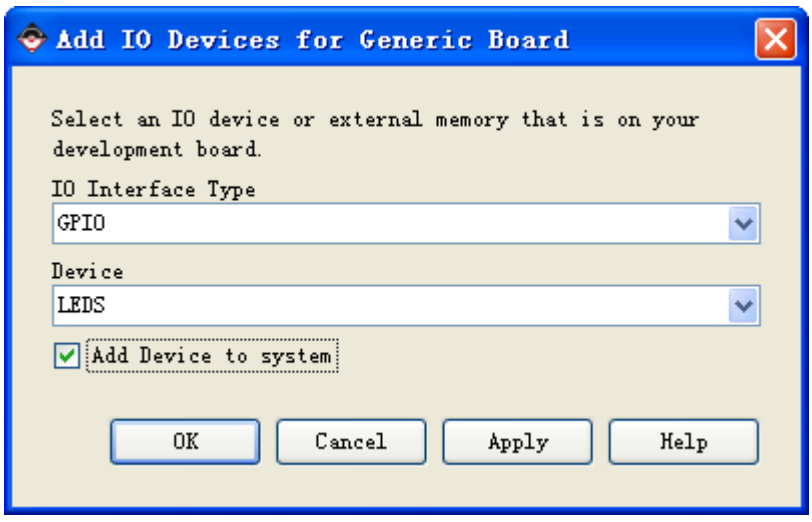
Processor Type : MicroBlaze

Local Memory : 8 KB

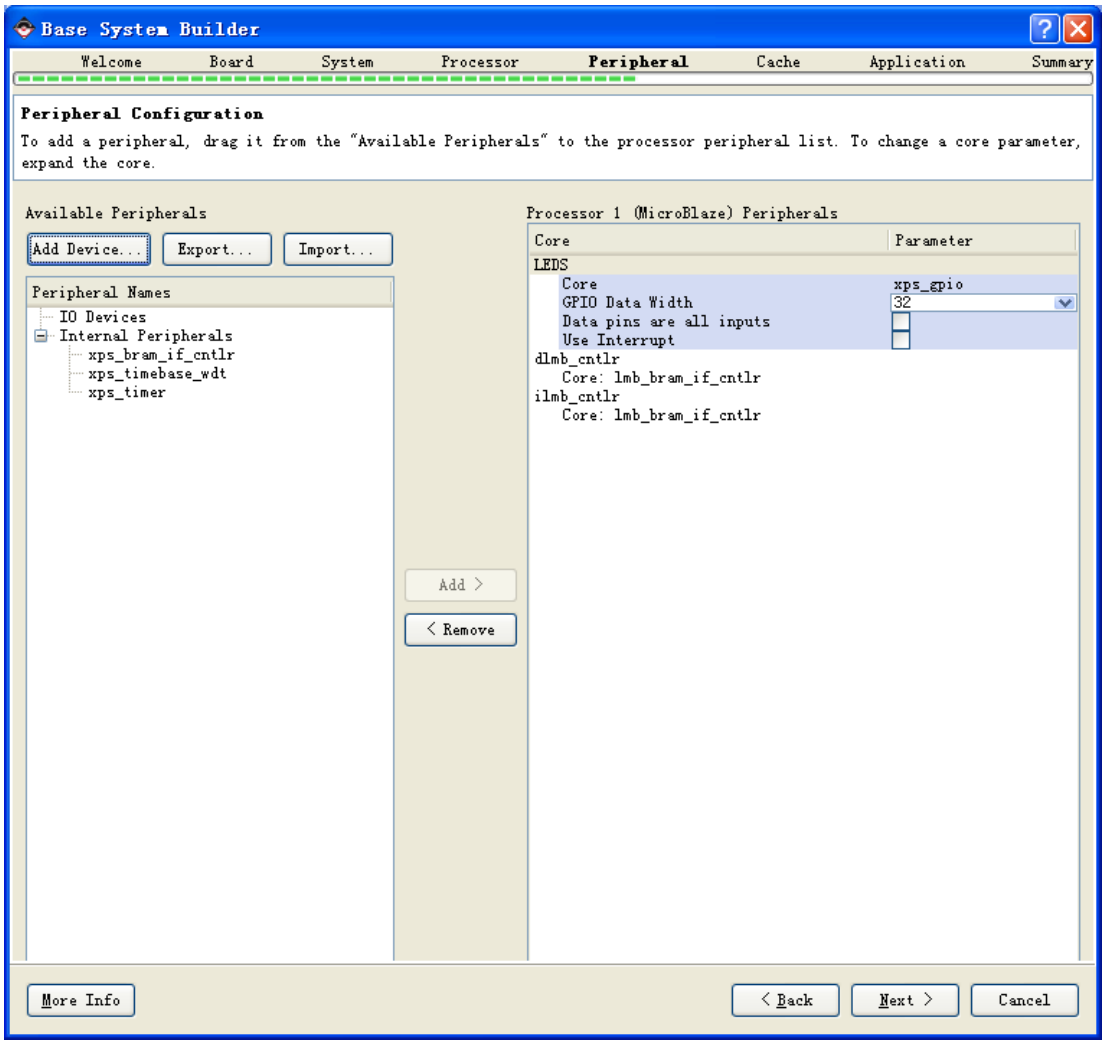
Next>



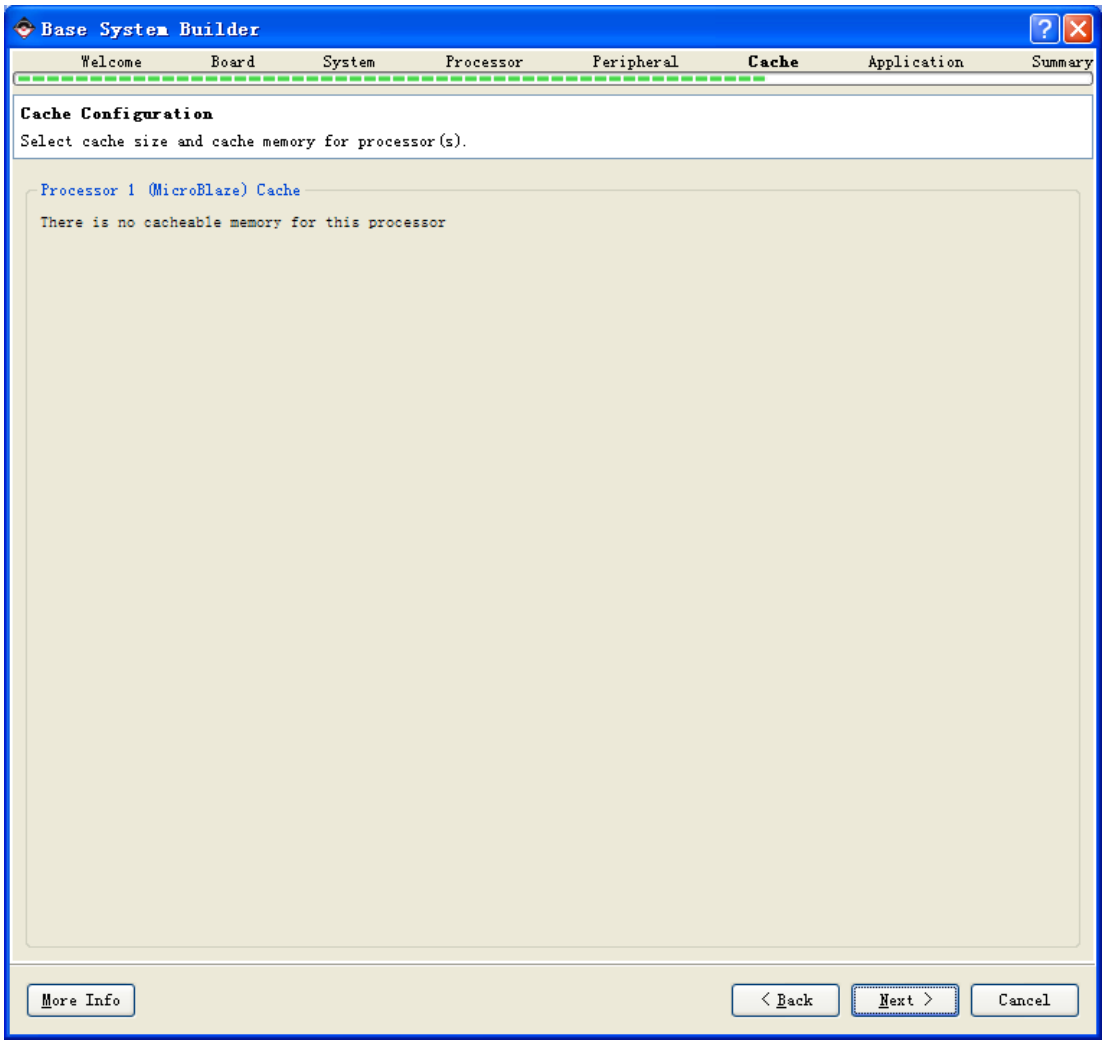
Add the port connects to the external and choose Add Device…



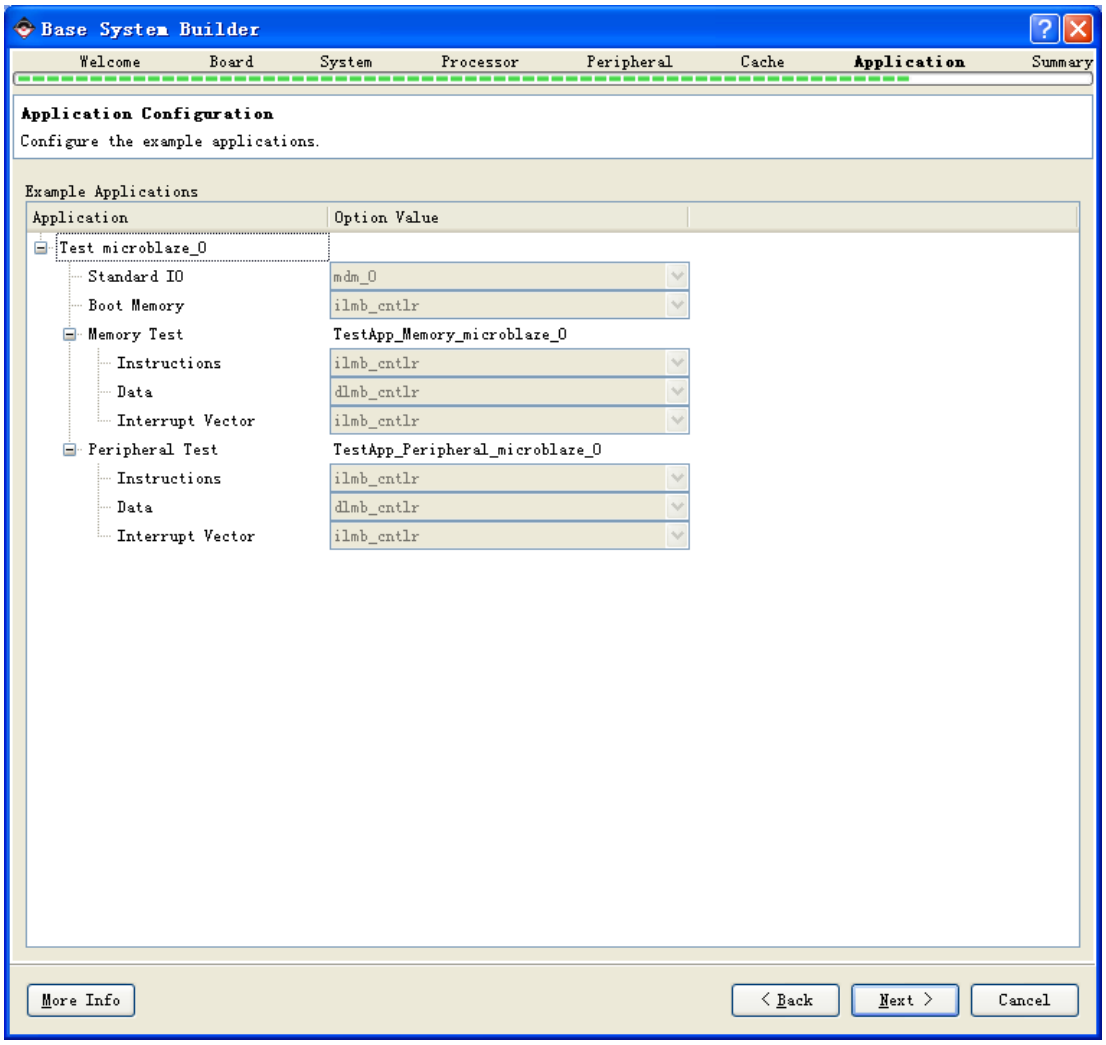
In pop-up dialogue choose IO Interface Type : GPIO， Device : LEDS，OK



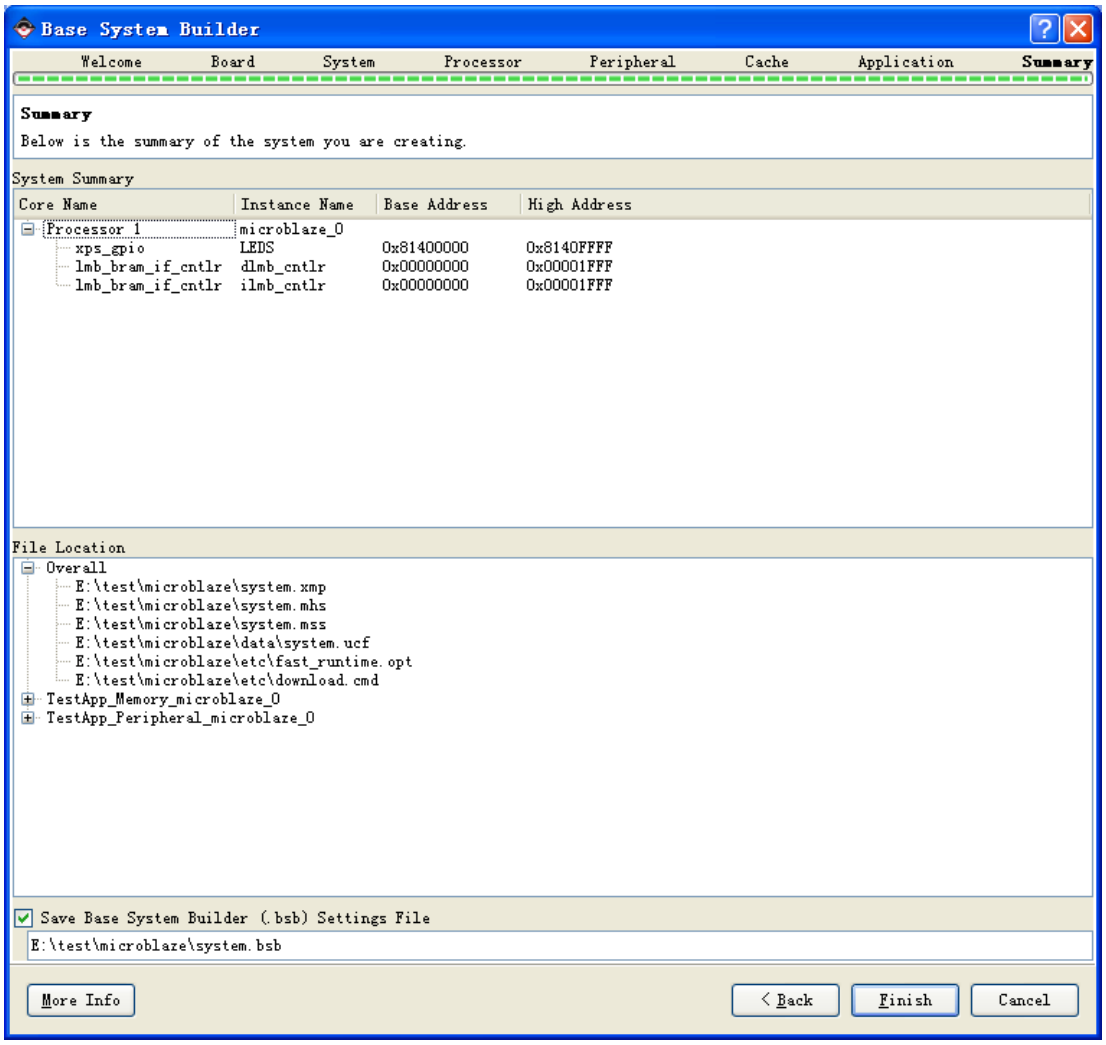
This experiment only needs to add LEDS. Choose Next>


Base System Builder
Welcome
Board
System
Processor
Peripheral
Cache
Application
Summary
Cache Configuration
Select cache size and cache memory for processor(s).
Processor 1 (MicroBlaze) Cache
There is no cacheable memory for this processor
More Info
< Back
Next >
Cancel

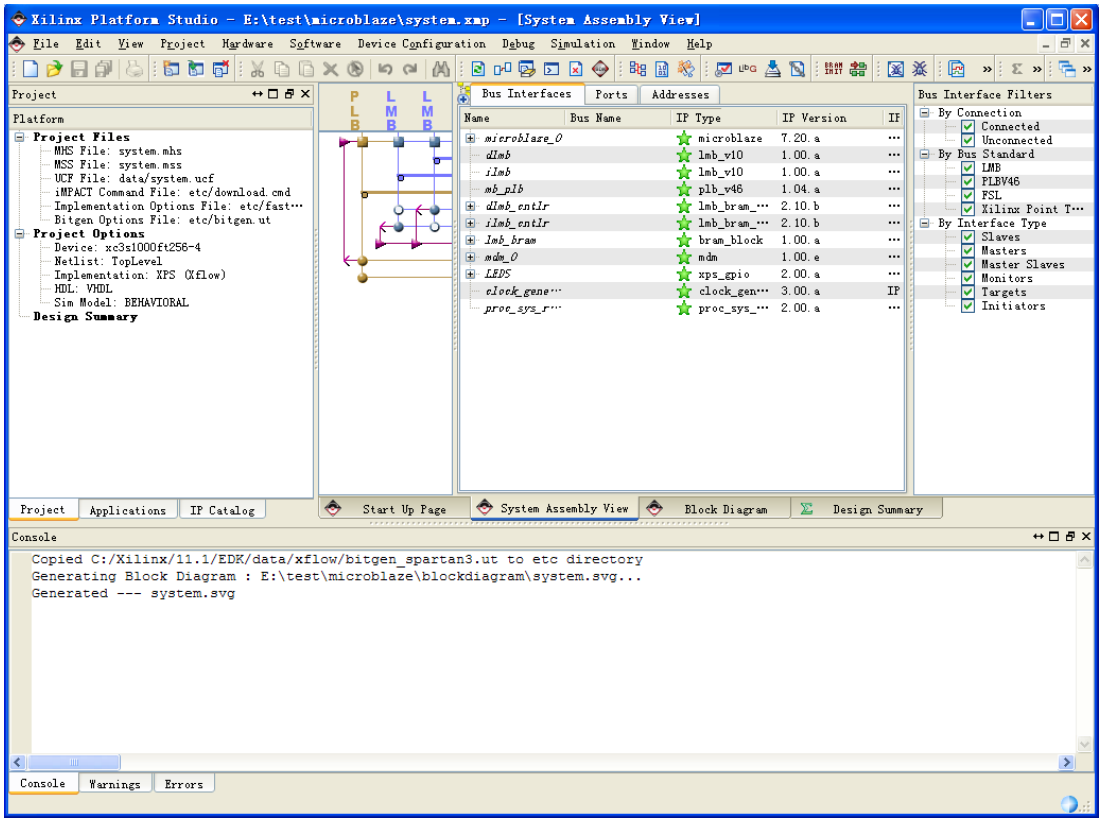
Next>


Base System Builder
Welcome
Board
System
Processor
Peripheral
Cache
Application
Summary
Application Configuration
Configure the example applications.
Example Applications
Application
Option Value
Test microblaze_0
Standard IO
mdm_0
Boot Memory
ilmb_cntlr
Memory Test
TestApp_Memory_microblaze_0
Instructions
ilmb_cntlr
Data
dlmb_cntlr
Interrupt Vector
ilmb_cntlr
Peripheral Test
TestApp_Peripheral_microblaze_0
Instructions
ilmb_cntlr
Data
dlmb_cntlr
Interrupt Vector
ilmb_cntlr
More Info
< Back
Next >
Cancel

Next>


Base System Builder
Welcome
Board
System
Processor
Peripheral
Cache
Application
Summary
Summary
Below is the summary of the system you are creating.
System Summary
Core Name
Instance Name
Base Address
High Address
Processor 1
microblaze_0
xps_gpio
LEDS
0x81400000
0x8140FFFF
lmb_bram_if_cntlr
dlmb_cntlr
0x00000000
0x00001FFF
lmb_bram_if_cntlr
ilmb_cntlr
0x00000000
0x00001FFF
File Location
Overall
E:\test\microblaze\system.xmp
E:\test\microblaze\system.mhs
E:\test\microblaze\system.mss
E:\test\microblaze\data\system.ucf
E:\test\microblaze\etc\fast_runtime.opt
E:\test\microblaze\etc\download.cmd
TestApp_Memory_microblaze_0
TestApp_Peripheral_microblaze_0
Save Base System Builder (.bsb) Settings File
E:\test\microblaze\system.bsb
More Info
< Back
Finish
Cancel

Finish

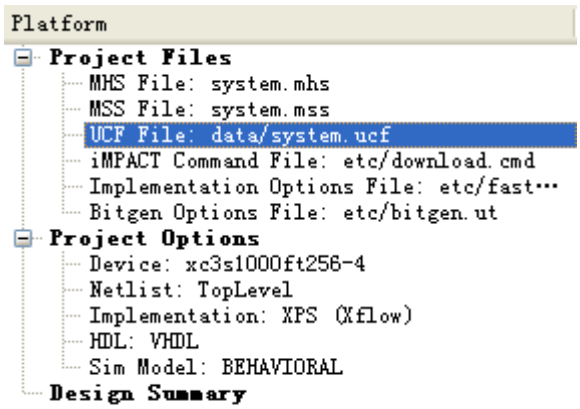


In Bus Interfaces, you can change property by double click item. Each interface related to the process is list in Ports. The upper External Ports lists external ports (pins need to be defined). In Addresses, you can modify peripheral devices' physical address. The programming after the address stands for this device

This experiment needn't modify.

4. Set download related

To download the generated processer to the development board, first of all, you have to define pins.



In Project Files submenu, double click UCF File: data\system\ucf. In the right working area, you can see following files:

Net fpga_0_LEDS_GPIO_IO_O_pin<0> LOC=;

Net fpga_0_LEDS_GPIO_IO_O_pin<1> LOC=;

Net fpga_0_LEDS_GPIO_IO_O_pin<2> LOC=;
Net fpga_0_LEDS_GPIO_IO_O_pin<3> LOC=;
Net fpga_0_LEDS_GPIO_IO_O_pin<4> LOC=;
Net fpga_0_LEDS_GPIO_IO_O_pin<5> LOC=;
Net fpga_0_LEDS_GPIO_IO_O_pin<6> LOC=;
Net fpga_0_LEDS_GPIO_IO_O_pin<7> LOC=;
Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 50000 kHz;
Net fpga_0_clk_1_sys_clk_pin LOC=;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC=;

Here list only 8 GPIO as there are only 8 LED on the development board. First, delete the "#" in front of the pin (the line starts with "#" is notes). Second, put the pin number between equal sign and semicolon in each line. After modification, you can see:

Net fpga_0_LEDS_GPIO_IO_O_pin<0> LOC=A5;
Net fpga_0_LEDS_GPIO_IO_O_pin<1> LOC=A7;
Net fpga_0_LEDS_GPIO_IO_O_pin<2> LOC=A3;
Net fpga_0_LEDS_GPIO_IO_O_pin<3> LOC=D5;
Net fpga_0_LEDS_GPIO_IO_O_pin<4> LOC=B4;
Net fpga_0_LEDS_GPIO_IO_O_pin<5> LOC=A4;
Net fpga_0_LEDS_GPIO_IO_O_pin<6> LOC=C5;
Net fpga_0_LEDS_GPIO_IO_O_pin<7> LOC=B5;
Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 50000 kHz;
Net fpga_0_clk_1_sys_clk_pin LOC=T9;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC=K14;

Save file and finish the IO definitions.

In Project Files submenu, double click iMPACT Command File:etc\download.cmd. In the right working area, you can see following files:

setMode -bscan
setCable -p auto
identify
assignfile -p 5 -file implementation/download.bit
program -p 5
quit

Here, you need to change 1 to 2 in line 4 and 5 because in the download chain of our development board, JTAG download in on the second. There are no other modifications required.

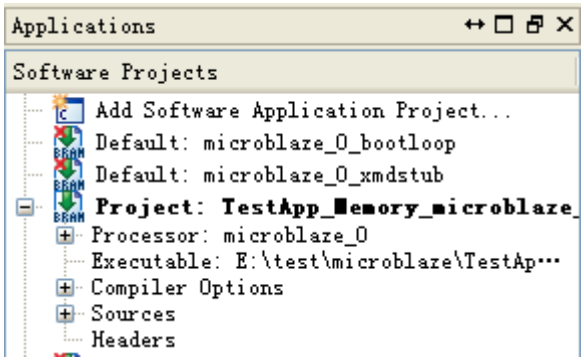
After modification, you can see:

setMode -bscan
setCable -p auto
identify

assignfile -p *2* -file implementation/download.bit
program -p *2*
quit
Save file.

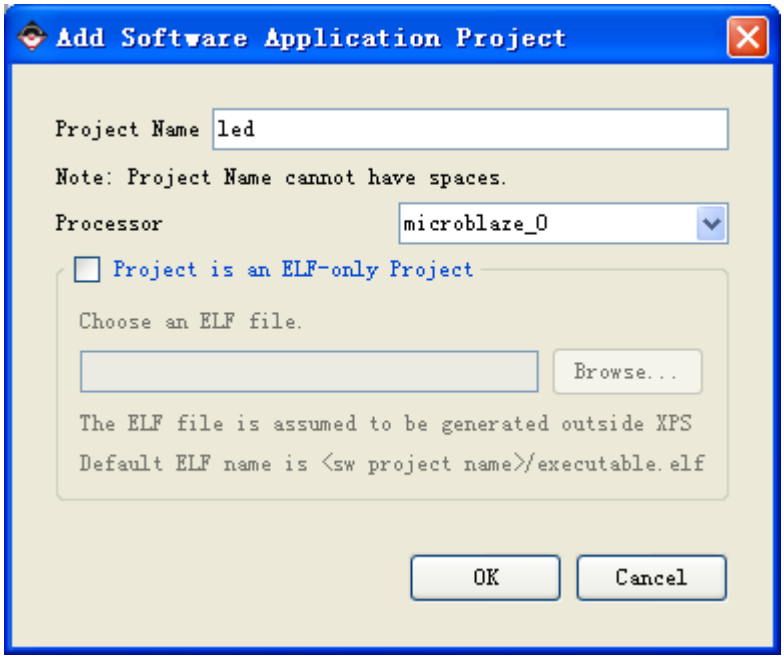
5. Write program C

After hardware design, comes software program. Choose Application page:

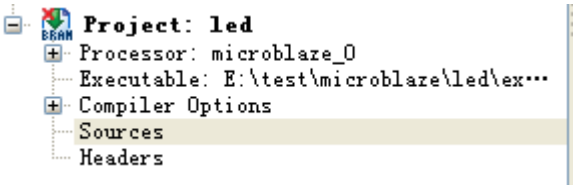


In upper picture, you can see, Platform add a storage testing program for the processer automatically. This program uses serial to output results which will be introduced in next experiment. Here, we have to re-write a C program. Detailed operations are as following:

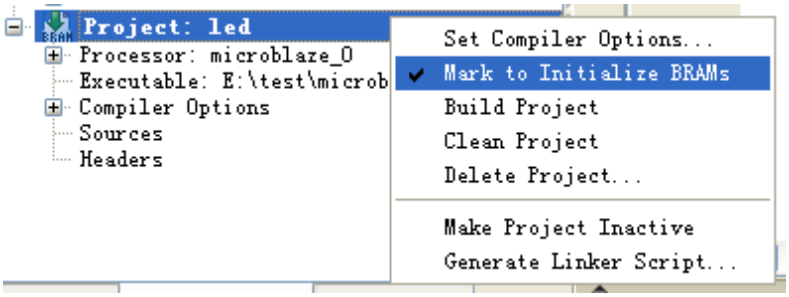
First, double click Add Software Application Project and then add a new project:



Write project name and used processer in upper dialogue. After OK, you can see the new project has been added in the list:



Now, you need to click right on the new set storage testing project. Choose Mark to Initialize BRAMs in the pop-up submenu. Activate this project because Platform allows only one project be active.



In upper column, right click on Source and choose Add New File to set a new C file for the project. Content is as following:

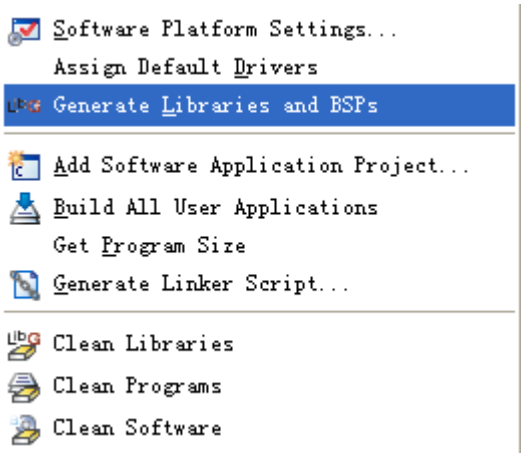
```
#include "xparameters.h"
void main(){
int *i;
i = 0x81400000 ;
(*i) = 0xff000000 ;
}
```

Save file。

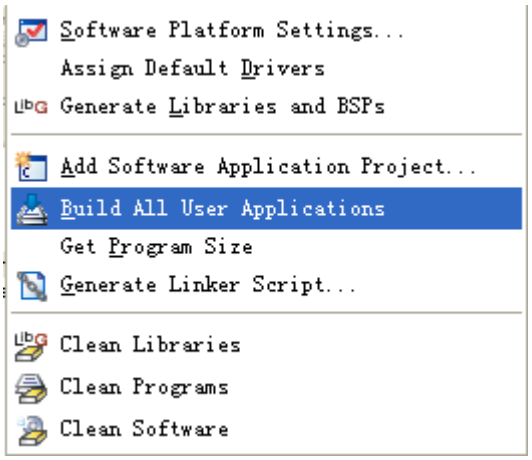
6. Download

First, operate on the software and in the tool bar choose Software：

Program C project: choose Generate Libraries and BSPs and will generate drivers of peripheral devices' and driver library. Configure STDIN/STDOUT and generate interrupt processing mechanism.



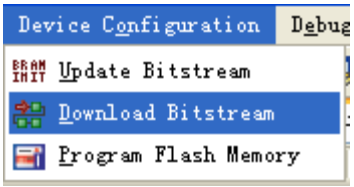
Then choose Build All User Applications to program project C.



Second, operate on hardware and in tool bar choose Hardware：
Generate Net list

Generate Bit stream

Then can do Download and in the tool bar choose Device Configuration：
Download Bit stream



5.1.5 Experiment result

When the download is finished, you can see LED on the development board turns on.

Experiment18 MicroBlaze control serial communication experiment

5.2.1 Experiment purpose

1. Master the usage of Xilinx Platform Studio
2. Learn the usage of Platform Studio EDK
3. Write program C, and control serial ‘s output.

5.2.2 Experiment theory

OPB UART Late is a serial controller provided by EDK for MicroBlaze. Features are as following:

- One sending pipe and one receiving pine (full-duplex).
- 16 characters’ sending and receiving FIFO
- Data bytes in character are configurate （5-8）
- Configurate parity is odd parity or even parity.
- Configurate Baud rate

UART Late provides four registers. In programming, user can achieve the serial communication features by controlling the internal data information:

UART Late Register

寄存器	地址映射	功能说明
Receive FIFO	UART_BASE_ADDRESS+0	从接收 FIFO 中读字符
Transmit FIFO	UART_BASE_ADDRESS+4	写字符到发送 FIFO 中
Status Reg	UART_BASE_ADDRESS+8	可读操作
Control Reg	UART_BASE_ADDRESS+12	可写操作

Status Reg Status Register

位数	名称	描述	复位值
0-23	保留位	未使用	0
24	PAR_ERROR	奇偶校验错误	0
25	FRAME_ERROR	贞错误	0
26	OVERUN_ERROR	益处错误	0
27	INTR_ENABLED	中断使能	0
28	TX_FIFO_FULL	发送 FIFO 满	
29	TX_FIFO_EMPTY	发送 FIFO 空	
30	RX_FIFO_FULL	接收 FIFO 满	
31	RX_FIFO_VALID_DATA	接收 FIFO 空	

Control Reg Control Register

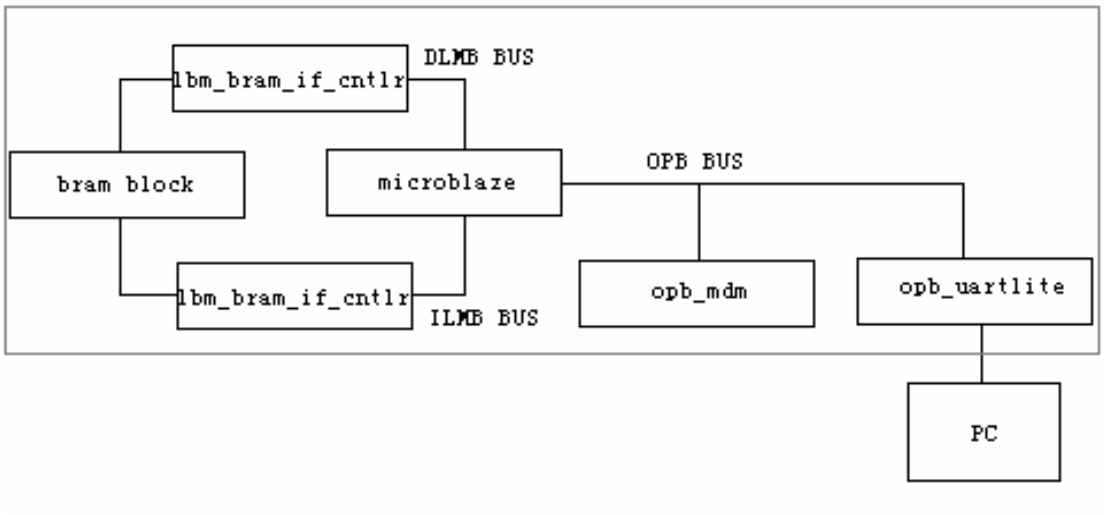
位数	名称	描述	复位值
0-26	保留位	未使用	0
27	ENABLE_INTR	UART 中断使能	0
28-29	保留位	未使用	0
30	RST_RX_FIFO	复位接收 FIFO	0
31	RST_TX_FIFO	复位发送 FIFO	0

When UART Lite interrupt enable is not set yet, if any of the following condition is met, the interrupt occours:

（1） If there is valid character existing in receiving FIFO, interrupt keeps activation until accept FIFO is empty.

（2） When sending FIFO from non-empty to empty, after send the last character of FIFO, the interrupt will be activated and keeps one clock circle.

Detailed design and achieving plan is as following:

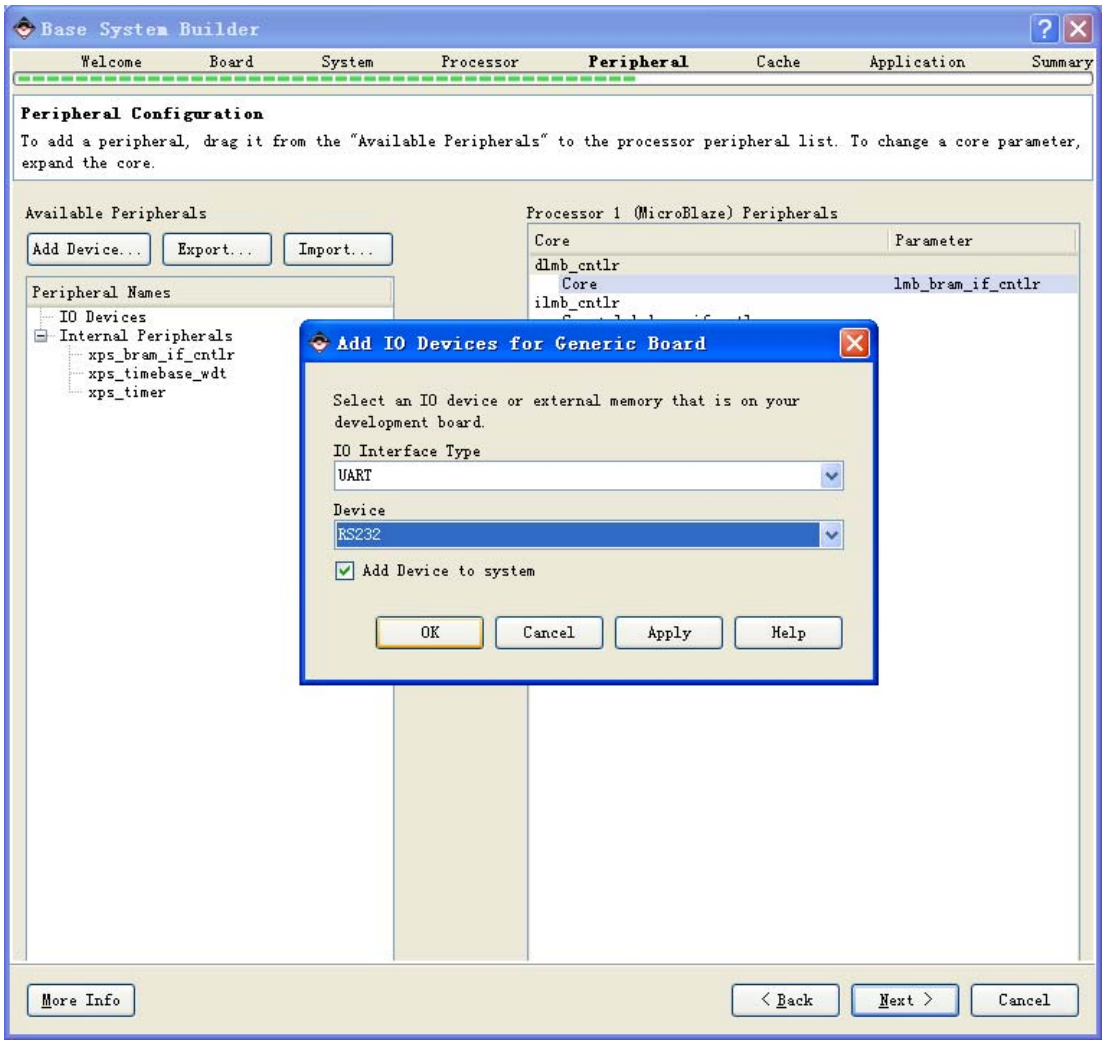


5.2.3 Experiment content

1. Test bram by MicroBlaze.
2. Use Platform to control serial output

5.2.4 Experiment steps

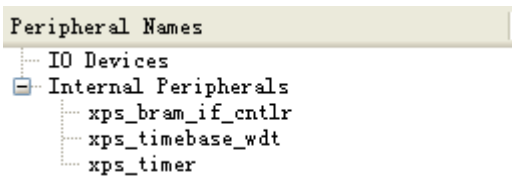
1. Set up a processer follow the experiment steps as last one. The difference is when choosing the peripheral device, select the UART and bram. Detailed operations are as following:



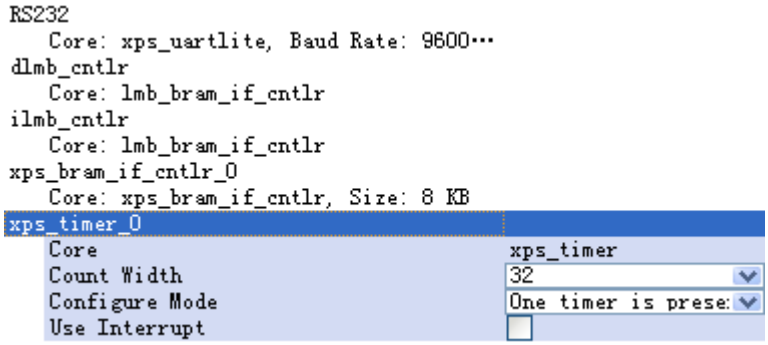
Choose UART， RS232，OK

In upper dialog, you can adjust the RS232 interface's property. Data Bits is 8bit. Baud rate chooses 9600. Parity chooses Odd.

And you have to add devices in the system.

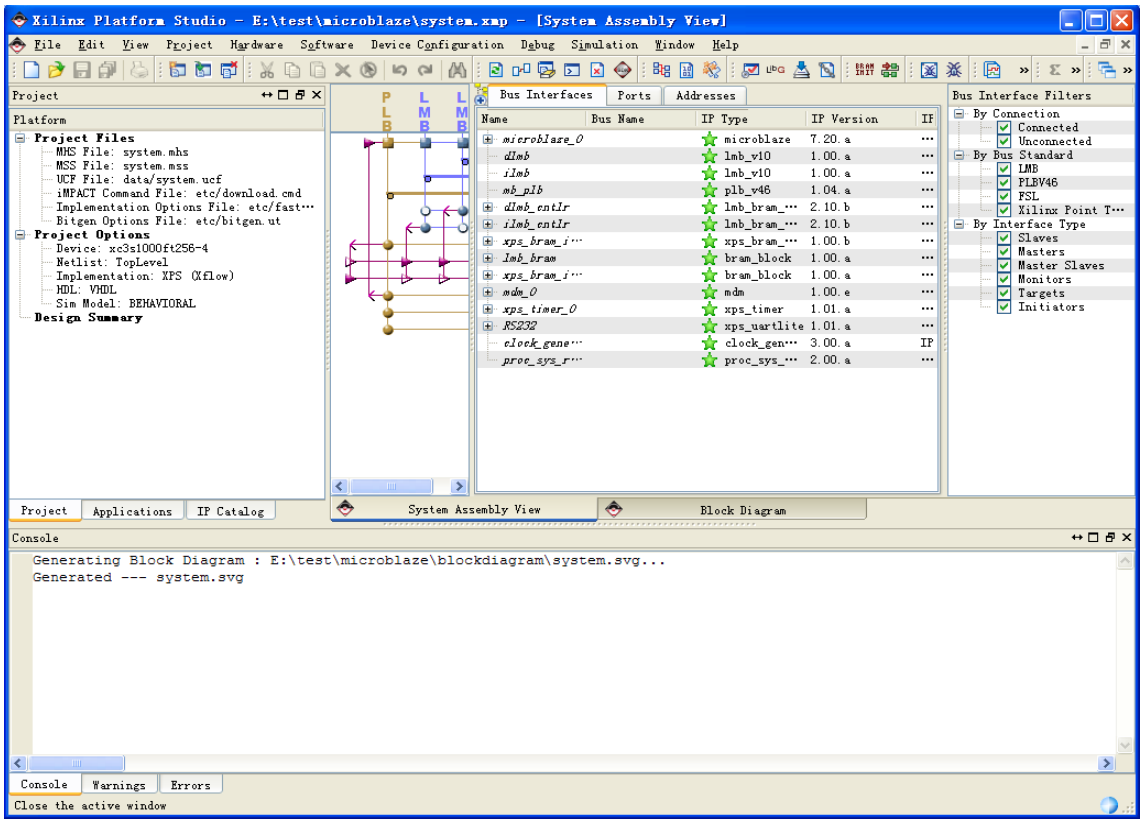


Add xps_bram_if_cntlr and xps_timer and adjust the property.



The added xps_bram_if_cntlr as upper is the interface controller on the OPB general and chip BlockRAM. Xps_timer is the clock controller of OPB general line. Choose One timer is present。

Following experiment steps just follow the steps as before which can generate one processor.



2．Adjust system property.

Double click the iMPACT Command File in the Platform window: etc\download.cmd. Change 1 to 2 in the line 4 and 5 as in our downloading chain, JTAG mode download is in the second. Other setting does not need adjustment.

After adjusting, it is as following:

setMode -bscan

setCable -p auto

identify

assignfile -p *2* -file implementation/download.bit

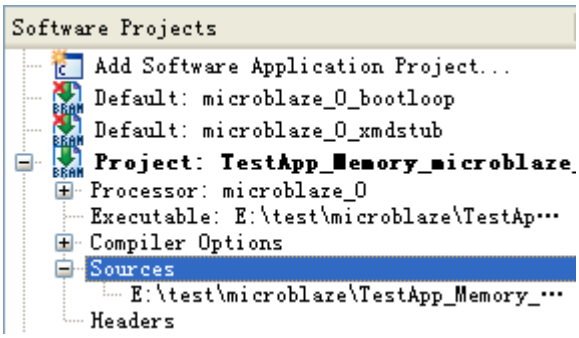
program -p *2*
quit
Save file.

Double click UCF File in the window Platform: data\system.ucf. Define the pins as following:

Net fpga_0_RS232_RX_pin LOC=C2;
Net fpga_0_RS232_TX_pin LOC=C1;
Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 50000 kHz;
Net fpga_0_clk_1_sys_clk_pin LOC=T9;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC=K14;
Save file.

3. Add module C

In the left project setting area, choose Applications page and we can see the testing C program provided by Platform:



Program C locates in Sources' subdirectory. You can click to check and do the necessary adjustment.

4. Download

Download steps are as experiment before.

After download is finished, if the downloaded is the chip memory testing program, on the PC you can see the letters: testing successful.

5. Set up a project C by yourself:

Set up a new project as the experiment before and close the original RAM testing program. Add program for the new project:

```
#include "xparameters.h"
#define UART_RX 0x84000000
#define UART_TX 0x84000004
#define UART_ST 0x84000008
#define UART_CT 0x8400000c
delay1()
{
    int q;
    for(q=0;q<16000;q++)
    { ;
    }
```

```
}
int main()
{
    int i,da,db;
    int *UART;
    char *j="hahaha\n1";
    UART=(int*)UART_ST;
    da=*(UART);
    i=1;
    do{
        UART=(int*)UART_TX;
        *(UART)=*(j);
        j++;
        i++;
        if((da&0x08)!=0)
            {
                UART=(int*)UART_CT;
                db=*(UART);
                db=db|0x01;
                *(UART)=db;
            }
            delay1();
        }while(*(j)!='1');
}
```

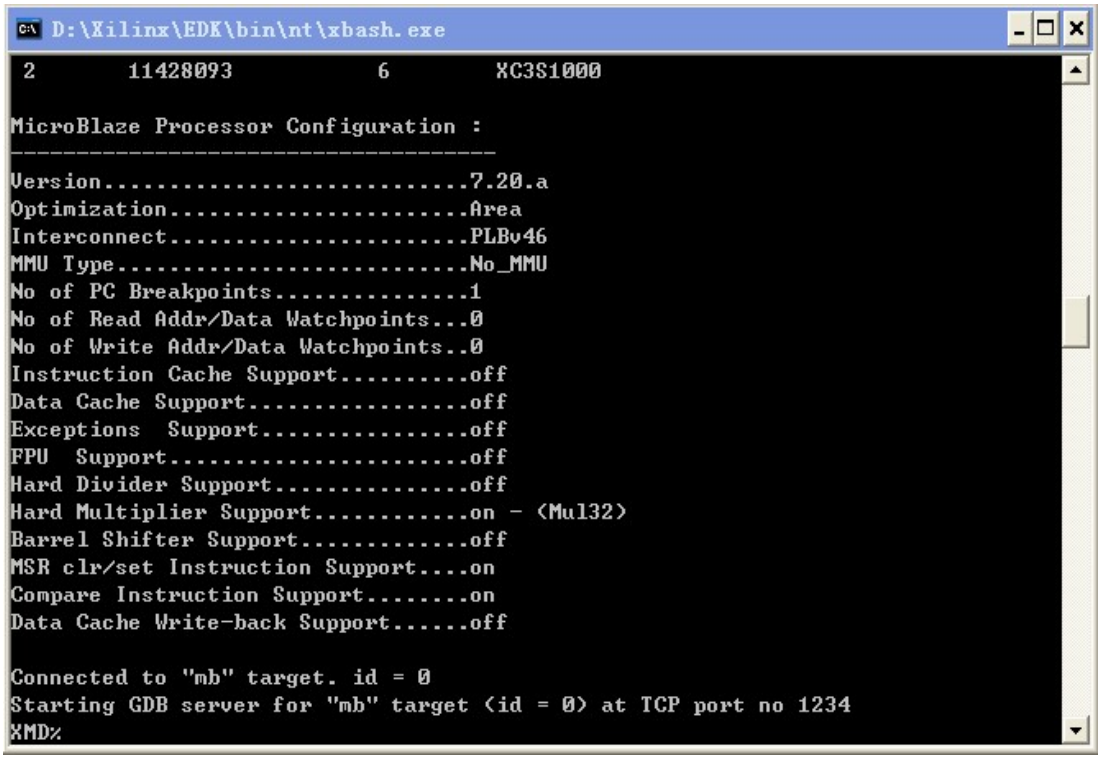
Generate library，Build project, update downloading bit file and download.

6．Online debugging:

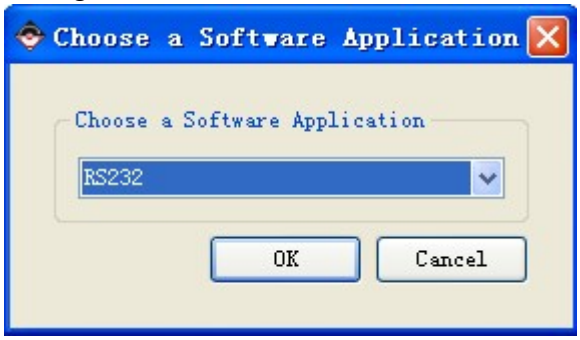
Use serial line to connect PC and development board and open serial debugging software.

You will use two icons  in the Xilinx Platform Studio tool bar when does the online debugging.

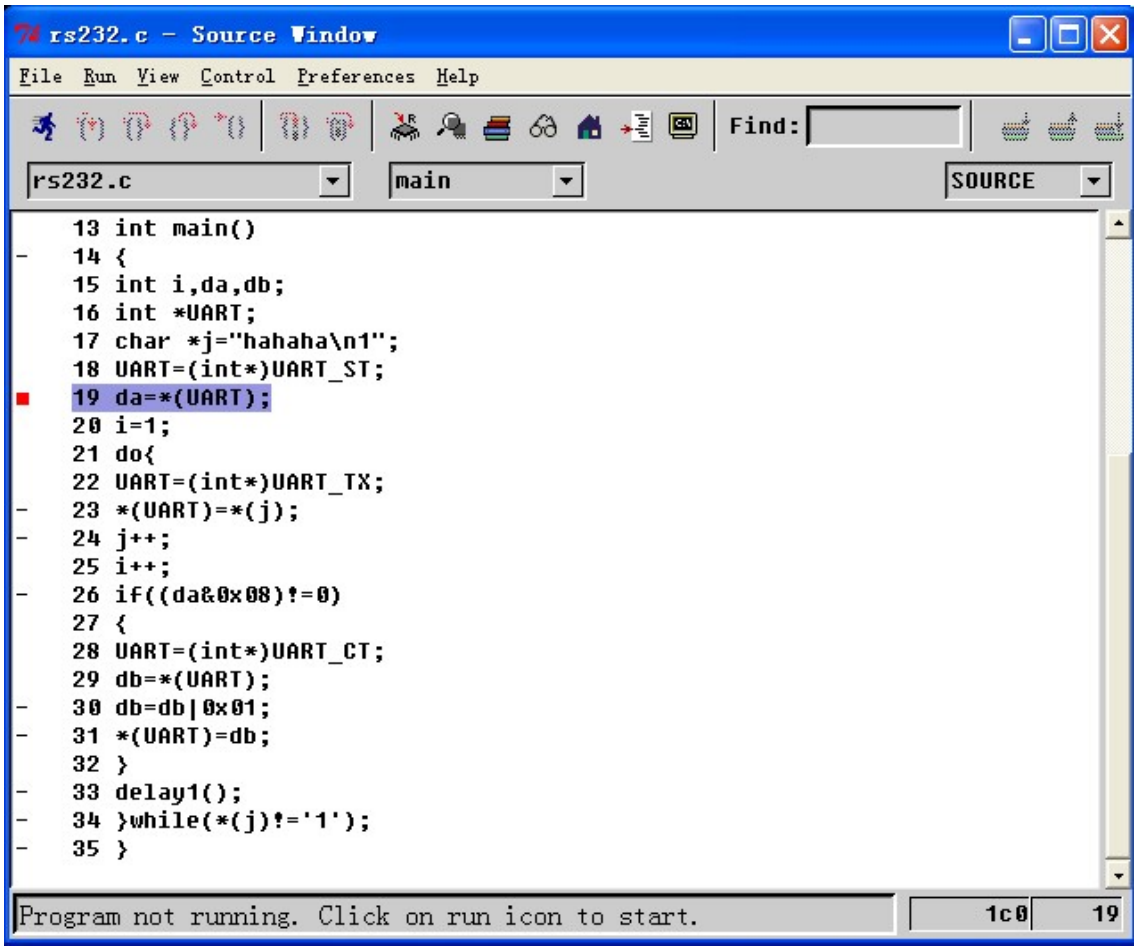
First, click  and comes the following dialog:



Then click the tool bar  . Choose the file name set by yourself in coming dialog:



OK



Click  and run the program. Continually click  until program running is finished. In the program processing, you can see the characters to be displayed in serial debugging software one by one.

5.2.5 Experiment result

Can output the specified characters from development board’s serial.