

Chapter 6

Chapter 6 The Usage of LPM Macro Module



IP





Here the macro modules and LPM functions of Quartus II include:

- Arithmetic component: adder, multiplier, accumulator, etc.
- Combinational circuit: multiplexer, comparer, LPM gate function, etc.
- ☑I/O component: PLL, etc.
- △ Memory: FIFO、RAM、ROM, etc.
- \square IP of single-chip computer, etc.



The Example of Invoking Macro Module of Counter

*This section gives the general usage methods of MegaWizard Plug-In Manager for the same type of macro modules by introducing the process of invoking and testing the LPM counter LPM_COUNTER.

The Invoking of the Text Code of the Counter LPM Module

Open the MegaWizard Plug-In Manager



Figure: Customizing new macro function block



Figure: Setting the LPM macro function block

Set up the project folder, for example, d:\LPM_MD, and choose Tools->MegaWizard Plug-In Manager.





LPM_COUNTER					nentation
1 Parameter Settings 2 EDA 3 S General General 2	Ummary Optional Inputs				
CNT4B	Cur How wide should the 'q' output bus be? What should the counter direction be? O Up only Down only Create an 'undown' input port to all	4	device family: bits	Cyclone IV E Match project/d	v lefault

Figure: The setting of 4-bit counter with addition and subtraction

Choose 4-bit counter and

choose "Create an updown input...", which makes the counter have the control functionality of add/subtract.





Figure: Setting the counter with the clock enable and carry output

Plain binary: common binary counter Modulus...: counter with the modulus of ... Clock Enable: clock enable control Carry-out: carry-out







Choose synchronous load and asynchronous clear

Figure: Adding 4-bit parallel data preset functions

The above-mentioned processes generate the Verilog file of the LPM counter, named "CNT4B.v", which can be invoked by the higher level Verilog program as the counter component.



6.1.2 Application of LPM Counter Code and Parameter Transmission Statement

% [Example]

- module CNT4B (aclr, clk_en, clock, data, sload, updown, cout, q);
- # input aclr, clk_en; //asynchronous clear,1 clear; clock enable, 1 enable, 0 disable
- input clock, sload;
 //clock input; synchronous preset load control, 1 load, 0 count
- input [3:0] data; input updown; //4-bit preset number, and addition and subtraction control, 1 addition, 0 subtraction
- output cout; output [3:0] q; //carry output and 4-bit count output
- #wire sub_wire0;wire [3:0] sub_wire1;// Defining internal connections
 - wire cout = sub_wire0; // The same assignment statement as *assign*
- # wire [3:0] q = sub_wire1[3:0]; // The same assignment statement as *assign*
- Ipm_counter lpm_counter_component(// Note that the unused ports in the instantiated statement must be connected to the specified level.
- sload(sload), .clk_en(clk_en), .aclr(aclr),
- data(data), .clock(clock), .updown(updown),
- .cout(sub_wire0), .q(sub_wire1), .aload(1'b0),
- .aset(1'b0), .cin(1'b1), .cnt_en(1'b1),
- % .eq(), .sclr(1'b0), .sset(1'b0));
- 🔀 defparam

Ħ

- # Ipm_counter_component.lpm_direction = "UNUSED",
- # lpm_counter_component.lpm_modulus = 12,
- # Ipm_counter_component.lpm_port_updown = "PORT_USED", // Use the addition and subtraction count
- # Ipm_counter_component.lpm_type = "LPM_COUNTER",
- lpm_counter_component.lpm_width = 4;
- 🔀 endmodule

- // Unused unidirectional counting parameters
- //counter with modulus of 12
- // Counter type
- // Counting bit width





defparam < macro module component instantiation name >.
 < macro module parameter name > = < parameter value >

[Example]

```
module REG24B (d, clk, q);
input [23:0] d; input clk;
output [23:0] q;
lpm_ff U1(.q (q[11:0]), .data (d[11:0]), .clock (clk));
defparam U1.lpm_width = 12;
lpm_ff U2(.q(q[23:12]), .data(d[23:12]), .clock(clk));
defparam U2.lpm_width = 12;
endmodule
```





For invoking the counter file "CNT4B.v", testing and implementing the counter, a program must be designed to instantiate it. Example of 6-3 is used to realize the functionality.

```
[ Example ]
```

```
module CNT4BIT (RST,ENA,CLK,DIN,SLD,UD,COUT,DOUT);
input RST, ENA, CLK, SLD,UD ;
input [3:0] DIN;
output COUT; output [3:0] DOUT ;
CNT4B U1(.sload (SLD), .clk_en (ENA), .aclr (RST), .cout (COUT),
.clock (CLK), .data (DIN), .updown (UD), .q (DOUT));
endmodule
```

6.1.3 Project Creation and Simulation Testing



Figure: The simulation waveform of CNT4BIT.v



6.2 Example of Building Attribute Control Multiplier

For implementing the multiplier, if use conventional method, the synthesized multiplier will occupy large logic resource and the speed may not be high. The useful method is to invoke the embedded hardware multiplier in the FPGA and this type of multiplier is commonly used in DSP technology. Thus, this type of multiplier is called DSP module.

The multiplier which uses R2 as the output port is constructed by the macro unit utilizing the way of pure combinational logic.

The multiplier which uses R1 as the output port is constructed by invoking the embedded multiplier in FPGA.





wire [15:0] R2, R1 /* synthesis multstyle = " logic " */

If the multiplier in the overall module is required to be constructed by using DSP module, the program can be written as follows:

module andd(A1,B1,A2,B2,R1,R2) /* synthesis multstyle = "dsp" */;



Flow Status	Successful - Tue May 23 21:32:59 2017
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	190 / 55,856 (< 1 %)
Total combinational functions	190 / 55,856 (< 1 %)
Dedicated logic registers	0 / 55,856 (0%)
Total registers	0
Total pins	64/325(20%)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	0 / 308 (0%)
Total PLLs	0/4(0%)

Figure: The compilation report of completely using logic macro units to construct the multiplier

Flow Status	Successful - Tue May 23 21:30:42 2017
Quartus II 64-Bit Version	13. 1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	0 / 55,856 (0 %)
Total combinational functions	0 / 55,856 (0 %)
Dedicated logic registers	0 / 55,856 (0 %)
Total registers	0
Total pins	64 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	2/308(<1%)
Total PLLs	0/4(0%)

Figure: The compilation report of invoking DSP module



The compiling report of example 6-4

Flow Status	Successful - Tue May 23 21:32:59 2017
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	190 / 55,856 (< 1 %)
Total combinational functions	190 / 55,856 (< 1 %)
Dedicated logic registers	0 / 55,856 (0 %)
Total registers	0
Total pins	64 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	0 / 308 (0 %)
Total PLLs	0/4(0%)





If the multiplier in the overall module is required to be constructed by using DSP module, the program can be written as follows:

module andd(A1, B1, A2, B2, R1, R2) /* synthesis multstyle = "dsp" */;

Flow Status	Successful - Tue May 23 21:30:42 2017
Quartus II 64-Bit Version	13. 1.0 Build 162 10/23/2013 SJ Full Versior
Revision Name	MULT8
Top-level Entity Name	MULT8
Family	Cyclone IV E
Device	EP4CE55F23C8
Timing Models	Final
Total logic elements	0 / 55,856 (0 %)
Total combinational functions	0 / 55,856 (0 %)
Dedicated logic registers	0 / 55,856 (0%)
Total registers	0
Total pins	64 / 325 (20 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	2/308(<1%)
Total PLLs	0/4(0%)



ategory:		More Analysis & Synthesis Settings		
General Files Libraries Operating Settings and Conditions	Analysis & Synthe Specify options for a affect VQM or EDIF	Specify the settings for the logic options in your project or entity in the Assignment Editor will override the option Existing option settings:	t. Assignments made to an individ n settings in this dialog box.	ual nod
Temperature	Optimization Techr	Name:	Setting:	
Compilation Process Settings Early Timing Estimate	Speed	Auto ROM Replacement	On	
Incremental Compilation	Balanced	Auto Resource Sharing	Off	
Physical Synthesis Optimizations	Area	Auto Shift Register Replacement	Auto	
EDA Tool Settings	0 mea	Block Design Naming	Auto	
Design Entry/Synthesis	Timing-Driven Sv	Carry Chain Length	70	
Simulation		Clock MUX Protection	On	
Formal Verification	Power-Up Don't	Create Debugging Nodes for IP Cores	Off	
Analysis & Synthesis Settings	Perform WYSIW	DSP Block Balancing	DSP blocks	
VHDL Input	PowerPlay power on	Disable Register Merging Across Hierarchies	Auto	
Verilog HDL Input		Extract VHDL State Machines	On	
Default Parameters	More Settings	Extract Verilog State Machines	On	

Figure: Selecting DSP Block Balancing as DSP blocks

6.3 Usage of Macro Block of Lab

In the design and development of involving memory applications such as RAM and ROM, invoking LPM module-type memory is the most convenient, most economical, most effective, and most efficient way to satisfy the design requirements. The following introduces the related technologies of using Quartus II to invoke LPM_RAM, including simulation test, generation of initialization configuration file, instantiation program expression, related attribute application and Verilog language description of memory.

6.3.1 Initialization File and Its Generation

- **K** In the design and development of the RAM and ROM applications, invoking LPM memory is the most convenient and cost-effective way for satisfying the design requirements.
- He initialization file of the memory is the data or code that can be configured in RAM or ROM. In the EDA design, the memory code file set or designed by the EDA tools is automatically invoked in the unified compilation.
- **Weight Strain Report Strai**



1. .mif format file

(1) The method of direct editing

File -> New -> Memory Initialization File



Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	80	86	8C	92	98	9E	A5	AA
08	B0	86	BC	C1	C6	CB	D0	D5
10	DA	DE	E2	E6	EA	ED	FO	F3
18	F5	F8	FA	FB	FD	FE	FE	FF
20	FF	FF	FE	FE	FD	FB	FA	F8
28	F5	F3	FO	ED	EA	E6	E2	DE
30	DA	D5	D0	CB	C6	C1	BC	86
38	B0	AA	A5	9E	98	92	8C	86
40	7F	79	73	6D	67	61	5A	55
48	4F	49	43	3E	39	34	2F	2A
50	25	21	1D	19	15	12	0F	0C
58	0.A.	07	05	04	02	01	01	00
60	00	00	01	01	02	04	05	07
68	0A	0C	OF	12	15	19	1D	21
70	25	2A	2F	34	39	3E	43	49
78	4F	55	5A	61	67	6D	73	79





1. .mif format

(2) The method of file editing

Example:

DEPTH=1	L28;	The numb	er of data in memory
WIDTH=8	3;	Width of c	output data
ADDRESS	_RAD	IX = HEX;	The data type of address
DATA_RA	ADIX :	= HEX;	The data type of memory
CONTENT	1		
BEGIN			
0000	:	0080;	Decuden editor con he would te decime
0001	:	0086;	Regular editor can be used to design
0002	:	008C;	MIF file. Address and data are both
2000			hexadecimal.
007E	×	0073;	
007F	:	0079;	Save as .mit file
END;			





1. .mif format

(3) specific mif file generator



Figure: Generation of .mif sinusoidal waveform file by using mif generator

🖡 DATA7X8.mif - 记事本
文件(22) 编辑(22) 格式(02) 查
DEPTH = 128;
WIDTH = 8;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT BEGIN
0000 : 0080;
8881 : 8886;
0002 : 008C;
0003 : 0092;
8084 : 8898;
0005 : 009E;
0006 : 00A5;
8987 : 88AA;
0008 : 0080;
007E : 0073;
007F : 0079;
END ;

Figure: Open .mif file





2. .hex format file

(1) method 1: New -> Hexadecimal (Intel-Format) File -> save as .hex format file

(2) method 2: The data is edited in HEX data editing window by using assembly program editor and .hex format file is generated by using assembly compiler.

3. .dat format file

.mif and .hex format file is related with the specific development software, as the invoking of them in Verilog text is necessily required to use the stipulated property expression of Quartus II.

However, the invoking of .dat format data file can be realized directly by using standard Verilog statements. The data format of .dat file is simplest and its form is given as follows:

00 E5 6D ... 34

6.3.2 Invoking LPM_RAM by Schematic Diagram Method

🕼 MegaWizard Plug-In Manager [page 2a]	
Which megafunction would you like to customize? Select a megafunction from the list below	Which device family will you be using? Cydone IV E Which type of output file do you want to create? AHDL VHDL
ALTUFM_PARALLEL	Verilog HDL What name do you want for the output file? D:/LPM_MD/RAM1P
FIFO partitioner LPM_SHIFTREG RAM initializer RAM: 1-PORT	Output files will be generated using the classic file structure Return to this page for another create operation Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in a library specified in the Libraries page of
RAM: 2-PORT ROM: 1-PORT ROM: 2-PORT	the Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu). Your current user library directories are:

Figure: Invoking single port LPM RAM



RAM: 1-PORT		About Documentation
1 Parameter 2 EDA 3 Summary Settings 3 Summary Summary Summary Widths/Blk Type/Clks Regs/Clken/Byte End	able/Adrs > Read During Write Option > Mem	Init
DAM1D	Currently selected device family:	Cyclone IV E
data[70] q[70] wren address[60] g inclock outclock Block type: AUTO	How wide should the 'q' output bus be? How many 8-bit words of memory? Note: You could enter arbitrary values for width and de What should the memory block type be?	Match project/default
	 M144K OLCs Set the maximum block depth to Auto • v 	Options
	What clocking method would you like to use? Single clock Dual clock: use separate 'input' and 'output' clocks	

Figure: Setting RAM parameters







Figure: Setting RAM to be controlled by input clock only



Figure: Setting to read the original data at the same time of writing data: Old Data







Figure: Setting the initialization file and allowing the in-system editing







Figure: The well-connected RAM module on a schematic diagram



6.3.3 Test LPM_RAM

in	CLK	
in	WREN	
1	⊳ A	00\01\02\03\04\05\06\07\08\09\00\01\02\03\04\05\06\07\08\09\0A\08\00\01\02\03\04\05\06\07\08\09\0A\08\00\01\02\03\04\05\06\07\08\09\2(
1	⊳ D	A0\D5\0A\3F\74\A9\DE\13\48\7D\82\E7\1C\51\86\8B\F0\25\5A\8F\C4\F9\2E\63\98\CD\02\37\6C\A1\D6\08\44
뱅	₽Q	(X80 X86 X8C X92 X98 X9E XA5 XAA X80 X86 X80 X86 X8C X92 X98 X9E XA5 XAA X80 X86 X8C XC1 X82 XE7 X1C X51 X86 X8B XF0 X25 X5A X8F X

Figure: The simulation waveform of RAM

6.3.4 Expression of Memory Initialization File Loading of Verilog Code Description

- In Section 6.3.2, the reader has already seen that invoking the initialization file from the edited memory can use the editor called by the LPM module to select the settings in a specific dialog box (as shown in Figure 6-17). But if you invoke the initialization file in the memory of the Verilog program of the pure code, you must use a specific instruction statement. Here are two methods.
- Here first method is to use the attribute statement given by Quartus II. These statements are used only in the Quartus II platform. On the right side of the memory definition statement of Example 2-3, there are:
- # /* synthesis ram init file="DATA7X8.mif" */ ;



***** The following definition expression is the Verilog-2001 version and its function is same:

(* ram_init_file = "DATA7X8.mif" *) reg[7:0]
mem[127:0]





The second method is to use the Verilog language directly, that is, using procedural statement *initial* and system function *\$readmemh*. Because the standard Verilog statement is used, its expression has general characteristics, so it is not limited to EDA software environment of Quartus II. As "initial" and "\$readmemh" are used, the format of initialization file must be ".dat".

In .dat file, the data starts from lower address bit. Therefore, in example 6-6, the memory "mem" is written as mem[0:127].

```
[ Example ]
module RAM78 (output[7:0]Q, input[7:0]D,input[6:0]A, input CLK,WREN);
reg[7:0] mem[0:127] ;
always @(posedge CLK ) if (WREN) mem[A] <= D;
assign Q = mem[A];
initial $readmemh("RAM78_DAT.dat", mem );
endmodule</pre>
```

6.3.5 Structure Control of Memory Design

The construction of memory in different Verilog expressions will obtain memories of different structures, such as a memory built by a logical macro unit or a memory built with an embedded RAM unit. The latter has the best resource utilization rate and the most concise and high-speed memory hardware structure for FPGA with large number of RAM units.

The RAMs described by Examples 6-6 and 6-7 have the same interface and function. Now let's compare their structure. The corresponding RTL diagram to Example 6-6 is shown in Figure 6-20; the corresponding RTL diagram to Example 6-7 is shown in Figure 6-21.





Figure: The RTL circuit module diagram of Example 6-6

```
[ Example ]
```

```
module RAM78(output reg[7:0] Q, input[7:0] D,input[6:0] A, input
CLK,WREN);
      reg[7:0] mem[127:0] /* synthesis ram_init_file="DATA7X8.mif" */;
```

```
always @(posedge CLK ) if (WREN) mem[A] <= D;
```

```
always @(posedge CLK ) Q = mem[A];
```

endmodule



Figure: The RTL circuit module diagram of Example 6-7



- ₩ Why the differences between the example 7-6 and 7-8 are so big? The reasons are given as follows:
- (1) The expression way of Verilog. The output Q of the memory in Example 7-6 adopts "assign" statements. There is no any register or memory components in this case. Therefore, this structure can not use the ready-made RAM in the FPGA.
- Example 7-8 uses two "always" process and the output Q has an added register. This expression satisfies the RAM structure in the FPGA.





(2) invoke the constraint configuration of the embedded RAM units. The correct and proper Verilog descriptions is the basis of invoking the RAM units of the FPGA, which however can not guarantee that the design will invoke the RAM units. This is because the synthesizer still does not know the design purpose of the users. For constructing the circuit with the use of RAM after synthesis, the constraint configurations are needed for the synthesizer of EDA tools.

Name:	Auto RAM Replacement	-
Setting:	On	-
Description	r	
and the second se	ALL ALL BUILDER ALL ALL ALL ALL ALL ALL ALL ALL ALL AL	which we have a set of the set of
on this op	with the altsynctam or the lpm_tam_dp ion may change the functionality of th n settings:	negafunction. Turning te design.
isting optio	with the altsyncram or the lpm_ram_dp ion may change the functionality of th n settings:	e design.
eisting optio	with the altsyncram or the lpm_ram_dp ion may change the functionality of th n settings: Nock Balancing	o megafunction. Turning ne design. Setting: On
isting optio Name: Auto RAM E	with the altsyncram or the lpm_ram_dp ion may change the functionality of th n settings: Nock Balancing Teplacement	o megafunction. Turning ne design. Setting: On On

Settings->Analysis & Synthesis Settings->More Settings->Auto RAM Replacement->On



6.4 Usage Examples of LPM_ROM

6.4.1 Design of Simple Sinusoidal Signal Generator

MegaWizard Plug-In Manager -> Memory Compiler -> ROM:1-PORT



Figure: Adding the initialization configuration file and allowing in-system access to ROM content.







Figure: The block diagram of sinusoidal signal generator

- Counter or address signal generator. Here according to the parameters of ROM above, we select 7bit output.
- Sinusoidal signal data memory ROM (7-bit address line, 8-bit data line), containing 128 8-bit waveform data (a sinusoidal period), that is, LPM_ROM: ROM78.
- Design of top-level schematic diagram.
- 8-bit D/A (set the experimental device to be DAC0832 for this example).





```
module SIN_GNT(RST,CLK,EN, Q,AR);
output [7:0] Q ; output [6:0] AR ;
input EN,CLK,RST ; wire [6:0] TMP; reg [6:0] Q1 ;
always @(posedge CLK or negedge RST )
if(!RST) Q1 <=7'B0000000;
else if (EN) Q1 <= Q1+1 ;
else Q1 <= Q1;
assign TMP=Q1; assign AR=TMP;
ROM78 IC1(.address(TMP), .inclock(CLK), .q (Q) );
endmodule
```





Figure: The circuit schematic diagram of sinusoidal signal generator



Figure: The circuit simulation waveform

6.4.2 Hardware Implementation and Testing of Sinusoidal Signal Generator

File Edit View	Project Pro	cessing Tools W	indow Help 💎			
Instance Manager	: 💽 🔊 🔳	📔 🔝 Ready to a	cquire	× 🥑	JTAG Chain	Configuration: JTAG ready
Instance	Status	LEs: 690	Memory: 61440	Small: 0/0	Hardware:	USB-Blaster [USB-0]
SNT1	Not running	690 cells	61440 bits	0 blocks	naraware.	(
					Device:	@1: EP3C55/EP4CE55 (0x020F50DD)
•		m		۲	>> SOF M	1anager: 👗 🕕 D:/LPM_MD/output_files/S
•		111		Þ	>> SOF N	Nanager: 🔛 🕕 D:/LPM_MD/output_files/S
< [3 23:41:16	111		•	SOF N	fanager: 🚠 🔘 D:/LPM_MD/output_files/S click to insert time
log: 2017/05/23 Type Alias	3 23:41:16 4 Name 150	II 151 152 153	154 155 156	157 158	>> SOF M	Ianager: 🚠 🕡 D:/LPM_MD/output_files/S click to insert time 161 162 163 164 165 166 1
Iog: 2017/05/23 Type Alias Image: State of the state of t	3 23:41:16 Name 150 - AR 041	151 152 153 1 05h X 06h X 0	154 155 156 17h X 08h X 09h X 0	↓ 157 158 Ah X 08h X 0C	SOF № 159 160 h X 00h X 0E	Ianager: Image Image D:/LPM_MD/output_files/S click to insert time click to insert time click to insert time 161 162 163 164 165 166 1 th X 0Fh X 10h X 11h X 12h X 14h

Figure: SignalTap II real-time test interface of sinusoidal signal generator data output

Instance Manage	er: 📉 🔊 🔳	🛯 🎦 o acquire 🛛	9 ×	JTAG Chain	Configuration: JTAG	ready				
Instance	Status	LEs: 690	Memo	Hardware:	USB-Blaster [USB-0]					Setup
SNT1	Not running	690 cells	6144	Device:	@1:EP3C55/EP4CE55 1anager: 🔔 🕕	i (0x020F50DD) D:/LPM_MD3/output	_files/SIN_GNT.sof			Scan Chair
log: 2017/05/2	24 20:41	40 204 2	20 21	56 102	click	to insert time bar	100 100	256	220 284	449 513
	⊞~AR									1
3	±- a	\sim	\cap	\checkmark	\sim	\sim	\sim	\sim	\wedge	\wedge
Data	 Setup 	III		1				_		×

Figure: The waveform display diagram of SignalTap II for sine signal generator

6.5 Application of In-System Memory Content Editor

(1) Open the editing window of in-system memory content editor

The in-system memory content editor of Quartus II reads or writes the data of the operating memory of FPGA via JTAG port, and the read or write process does not affect the operation of the FPGA.

Tool->In-System Memory Content Editor.

🛲 In-System	n Memory Content Ed	itor - D:/LPM_MD/S	SIN_GNT - SIN	I_GNT					
File Edit 1	View Processing Tool	s Window Help	P					Search altera	a.com
Instance Ma	anager: 🛐 📳 🔳	Ready to acquir	e		() ×	JTAG Chair	n Configuration: אָד	AG ready	Ø ×
Index	Instance ID	Status	Width	Depth	Ту	Hardware:	USB-Blaster [USB-	0]	Setup
0	RM78	Not running	8	128	RA	Device:	@1: EP3C55/EP4K	CE55 (0x020F50DD) 🔹	Scan Chain
		III			Þ	File: 볿	1		
Instance 0:	RM78								
000000	80 86 8C 92 98	9E A5 AA B0 B	5 BC C1 C6	CB DO D5 D	A DE E2 E	6 EA ED	F0 F3 F5 F8		
00001a	FA FB FD FE FE	FF FF FF FE FI	E FD FB FA	F8 F5 F3 F	O ED EA E	6 E2 DE	DA DS DO CB		
000034	C6 C1 BC B6 B0	AA A5 9E 98 92	2 8C 86 7F	79 73 6D 6	7 61 5A 5	5 4F 49	43 3E 39 34	ysm	gaZUOIC>94
00004e 000068	2F 2A 25 21 1D OA OC OF 12 15	19 15 12 OF 00 19 1D 21 25 22	COA 07 05 A 2F 34 39	04 02 01 0 3E 43 49 4	1 00 00 0 F 55 5A 6	0 01 01 1 67 6D	02 04 05 07 73 79	/*&!!&*/49>CI	OUZagmsy

Figure: In-System Memory Content Editor editing window, reading waveform data from the ROM of FPGA



(2) Read the data of the ROM

Edit	M Men	nory Co	onter ing	nt Ed	litor	- D:/ Windo	VLPM	I_MD Help)/SIN	N_GI	NT -	SIN	_GN	IT								_						1	Searc	ch alte	era.co	om	
instance M	1anager	: 📑	Ð		E	Re	ady 1	to acc	quire							Tİ.	0	×		JTAG	Chai	n Co	nfigu	ratio	n: J	TAG r	eady					(
Index		Insta	ance I	D		Statu	JS			Widt	th			De	pth			T		Hard	ware:	US	B-Bla	aster	USB	8-0]				•		Setup)
0		RM7	3			Notri	unnin	ig		8				12	B			R/		Devid	e:	@	1: EP	3C5	5/EP4	KE55	(0x02	0F50	DD)	•		Scan C	hain
	DM79				III													•		File:													
00000	80	86 8C	92	98	9E	A5	AA	BO	B6	BC	C1	C6	СВ	DO	D5	DA	DE	E2	E6	EA	ED	FO	F3	F5	F8								
0001a	FA	FB FD	FE	FE	FF	FF	FF	FE	FE	FD	FB	FA	F8	F5	F3	FO	ED	EA	E6	E2	DE	DA	D5	DO	CB								
	00	C1 BC	B6	B0	AA	A5	9E	98	92	80	86	7 F	79	73	6D	67	61	5A	55	4F	49	43	3E	39	34		•••	•••		.ys	mga	ZUOI	:C>
00034	Co								1 mm			A.F.	04	00	01	01	00	00	00	01	01	02	04	0E	07	14	0.1						

Figure: In-System Memory Content Editor editing window, reading waveform data from the ROM of FPGA



(3) Write data

Instance 0: I	RM78																									
000000	11	11	11	11	11	11	11	AA	BO	B6	BC	C1	C6	CB	DO	D5	DA	DE	E2	E6	EA	ED	FO	F3	F5	F8
00001a	FA	FB	FD	FE	FE	FF	FF	FF	FE	FE	FD	FB	FA	F8	F5	F3	FO	ED	EA	E6	E2	DE	DA	D5	DO	CB
000034	C6	C1	BC	B6	BO	AA	A5	9E	98	92	80	86	7F	79	73	6D	67	61	5A	55	4F	49	43	3E	39	34
00004e	2F	2A	25	21	1D	19	15	12	OF	00	0A	07	05	04	02	01	01	00	00	00	01	01	02	04	05	07
000068	OA	00	OF	12	15	19	1D	21	25	2A	2F	34	39	3E	43	49	4F	55	5A	61	67	6D	73	79		

Figure: Here, the edited data is loaded into the ROM in the FPGA



After modification, choose Write Data to In-System Memory command in the processing menu. Then the edited data can be download to the LPM_ROM through JTAG port.

Figure: The data waveform measured by SignalTap II





(4) The input and output data file

The data read from the system can be saved as MIF and HEX format file in the computer or "in-system" downloaded to the FPGA, through the command of Export Data to File or Import Data from File in the menu.



6.6 Invoke of Embedded PLL of LPM

The Cyclone/II/III/IV and Stratix/II/III/IV FPGA contain high performance PLL, which can be synchronized with the input clock signal. The input clock signal is also severed as the reference. The PLL can thus output one or several synchronized frequency scaling or frequency division on-chip clock.



6.6.1 Building Embedded PLL Component

Which megafunction would you like to customize? Select a megafunction from the list below	Which device family will you be using?	Cydone IV E
۹. 🗙	Which type of output file do you want to	o create?
ALTMEMPHY ALTOCT ALTPLL ALTPLL_RECONFIG ALTREMOTE_UPDATE	 VHDL Verilog HDL What name do you want for the output D:/LPM_MD/PLL20M 	file?

Figure: Choosing phase-locked loop ALTPLL





Figure: Selecting the input reference clock inclk0 to be 20MHz

MegaWizard Plug-In Manager->Create a new custom->I/O->ALTPLL





Figure: Selecting the control signal of the PLL

PLL enable: pllena asynchronous reset: areset lock output: locked



PLL20M	Able to implement the requested PLL Use this dock Clock Tap Settings	Requested Settings	Actual Settings
Operation Mode: Normal Clk Ratio Ph (dg) DC (%)	Enter output dock frequency:	0.002 MHz	• 0.002000
<u>c0 1/10000 0.00 50.00</u>	 Enter output clock parameters: Clock multiplication factor 		1
Cyclone IV E	Clock division factor		Copy 10000
	Clock phase shift	0.00 🖨 deg	• 0.00
	Clock duty cycle (%)	50.00	50.00

Figure: Selecting the output frequency of c0 to be 0.002MHz

Set up the frequency, phase and duty circle of the output clock



	PLL20M	Able to implement the requested PLL	IOCK		
inclk0	inclk0 frequency: 20.000 MHz CI.	Use this dock Clock Tap Settings	Requested S	Settings	Actual Settings
	Cik Ratio Ph (do) DC (%)	Enter output dock frequency:	195	MHz 💌	195.000000
	c0 1/10000 0.00 50.00 c1 39/4 0.00 50.00	 Enter output clock parameters: Clock multiplication factor 	1		39
		Clock division factor	1	<< Copy	4

🔹 deg 🔻

0.00

0.00

Figure: Outputing second clock signal c1

Clock phase shift

Cyclone IV E







Figure: A sinusoidal signal generator circuit using embedded phase-locked loop as the clock

6.7 The Usage of In-System Sources and Probes Editor

SignalTap II and In-System Memory Content Editor can bring great convenience for logic system design, test and debug. However, they also have some disadvantages. For example, SignalTap II (1) occupies a large number of memory units as the data buffer; (2) unidirectionally gather and display the information of hardware system in the operation and can not interact with the system bidirectionally. In-System Memory Content Editor can interact with the system bidirectionally, the target of which is however only confined to memory.

In-System Sources and Probes Editor can overcome the above-mentioned drawback. In particular, the test signal for the system hardware does not have to connect to the I/O port (i.e. all the test signal internally connects to the test system). All these tasks are realized by the communication through JTAG port of FPGA.



In-System Sources a	nd Probes	About D	ocumentation
Parameter 2 EDA 3 Summary Settings			
JTAG1	Currently selected device family	Cyclone III	
← probe[150]	 Do you want to specify an Instance Index? No, assign it automatically Yes, use this number 		
	The 'Instance ID' of this instance (optional): JGTT		4 characters
	How wide should the probe port be? [16	•	bits
	How wide should the source port be? 3	•	bits

Figure: Setting the parameters for In-System Sources and Probes module

MegaWizard Plug-In Manager->Create a new custom megafunction variation->JTAG-accessible->In System Sources and Probes

The testing port "probe" of the "JTAG1" module is set up to be 16 bits and the signal source is 3 bits.







Figure: Adding In-System Sources and Probes testing module in the circuit



File Edit V	ew Proce	ssing Tool	- vVi	noow neg	e e											
Instance Mana	ger: 📑		E.	🔇 Read	l <mark>y to acquir</mark> e					- 😡 ×	JTAG Ch	ain Confi	guration:	JTAG re	ady	
Probe read in	iterval			E	Event log						Hardwar	e: USB-	Blaster [US	B-0]		
Current inte	val: 0 came	les per seco	d	1	- Mavimum size	16		-			Devices	@1.	EP3C16/EP	4CE15 (0v020E	2000)
Currentinte	var. o samp	ies per seco	u	2 °							Device.		El 3010/El	100101	UNDEDI 1	20007
O Automat	ic				Save data	to event le	og				File:	D:/L	M_MD1/SI	N_GNT.	sof	
Manual	1		S	- Wr	rite source da	a: Contir	nuously 🔻									
			1	2455-5415	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2222.22	0.000	STOCKNER .	1		1					
Index	I	nstance ID		Status	Sour	ces: 3	Prot	es: 16	1	ame]					
Index 0	I JGTT	nstance ID	Not	Status running	Sour 3	ces: 3	Prot 16	oes: 16	TAG1:inst	ame altsource_pr						
Index	I JGTT	nstance ID	Not	Status running	Sour 3	ces: 3	Prob 16	oes: 16	таб1:inst4	ame altsource_pr						
Index	I JGTT	nstance ID	Not	Status running	Sour 3	ces: 3	Prot 16	es: 16	л ЛАG1:inst4	ame altsource_pr						
Index 0 1 0 1 0 1 0	JGTT	nstance ID	Not	Status running	Sou. 3	ces: 3	Prol	oes: 16	TAG1:inst	ame altsource_pr						
Index 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0	I JGTT Pe lia Na	ame [Not	Status running	3 -15	-14	Prot 16 -13	bes: 16	TAG1:inst	ame altsource_pr -11	-10	-9	-8		-7	
Index 0 JGTT Index Typ P15 \$	I JGTT Ne lia Na COUT	ame [Not	Status running -16	3 -15	ces: 3	Prot 16 -13	es: 16	ITAG1:inst	ame altsource_pr -11	-10	-9 T	-8		-7	
Index 1 0 JGTT Index Typ P15 P148]	I JGTT e liz Na ≩ COUT 2 ⊡ A	ame [] R[60]	Not ata 0	Status running -16 27h	3 -15 X 28h	-14 -14	Prot 16 -13 29h (2Ah	JTAG1:inst -12 X 28h	ame altsource_pr -11 -12	-10 -20h	-9	S Eh X	2Fh	-7 X	30h
Index 0 JGTT Index Typ P15 \$ P[148] \$ P[70] \$	I JGTT Me liz Na X COUT X ⊞ A	ame [] R[60] [70]	Not ata 0 37h iCh	Status running -16 -27h FAh	Sou 3 -15 <u>(28h</u> X F8h	-14 -14 X	Prot 16 -13 -13 -13 -13 -13 -13 -13 -13 -13 -13	2Ah F3h	-12 -12 X 28h X F0h	ame altsource_pr -11 X 2Ch X EDh	-10 -10 (2Dh (EAh	-9) :) :	-8 Eh X :6h X	2Fh E2h	-7 	30h DEh
Index	I JGTT Me liz Na X COUT X ⊕ Q X ⊕ Q	ame [] R[60] [70]	Not ata 0 37h iCh 1	Status running -16 -27h FAh	3 -15 X 28h X F8h	-14 -14 X	Prot 16 -13 29h (F5h (2Ah F3h	-12 -12 X 28h X F0h	ame altsource_pr -11 X 2Ch X EDh	-10 -2Dh (EAh	-? -:: 	-8 I Eh (.6h (2Fh E2h	-7 	30h DEh
Index	I JGTT JGTT A COUT A A COUT A A COUT A A COUT A A COUT A A COUT A A COUT A A COUT A A COUT A A A COUT A A A A A A A A A A A A A A A A A A A	ame [] 	Not ata 0 37h 1 1	Status running -16 27h FAh	3 -15 X 28h X F8h	-14 -14 X	Prot 16 -13 29h (F5h (2Ah F3h	-12 -12 X 28h X F0h	ame [altsource_pr -11 X 2Ch X EDh	-10 -2Dh (EAh	-9 	-8 Eh (2Fh E2h	-7 	30h DEh

Figure: The testing condition of In-System Sources and Probes Editor

Tool->In-System Sources and Probes Editor

Maximum Size of Event Log refers to the number of sampling period, which is usually $8 \sim 32$, (most of time is 32).

S2 \smallsetminus S1 \smallsetminus S0 are the signals controlling the output of the source, which corresponds to RST \searrow EN \searrow CLK respectively.

6.8 Principle and Application of DDS

Direct Digital Synthesizer (DDS) is a frequency synthesis technology, which has high frequency resolution, can achieve fast frequency switching, and can keep the continuous phase in the change. It is easy to realize the numerical control modulation of frequency, phase and amplitude. Therefore, the application of direct digital frequency synthesizer is particularly extensive in the design of frequency source of modern electronic systems and equipment, especially in the field of communication. This section introduces the working principle of DDS and its hardware implementation.



6.8.1 Principle of DDS



Figure: Basic DDS structure



DDS has the following four features.

(1) Theoretically, the frequency resolution of DDS can get the corresponding resolution accuracy when the bit number N of phase accumulator is large enough, which is difficult to achieve by traditional methods.

(2) DDS is an open loop system of fully digital structure without feedback link, so its speed is extremely fast, usually in the nanosecond order.

(3) The phase error of DDS mainly depends on the phase characteristics of the clock, and the phase error is small.

(4) The phase of DDS continuously changes, and the signal formed has good frequency spectrum, which cannot be realized by the traditional direct frequency synthesis method.

6.8.2 Example of DDS Signal Generator



Figure: The top-level schematic diagram of DDS signal generator circuit







Figure: The simulation waveform of Figure 6-41