Design Instantiation Guide for Angelo Device

Application Note

Introduction

This document describes how to directly instantiate tri-state I/O, PLL and EMB9K modules in user's design. It is only adequate for Angelo devices.

Using Wizard Manager of Primace software can easily implement PLL and EMB9K instantiation. If you do not want to use Wizard Manager, this document can be a good reference for you to manually implement the instantiations.

Tri-state I/O

Architecture Diagram

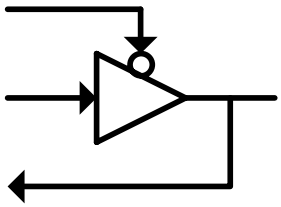

Figure 1 Tri-state I/O Diagram

Instantiation Template

```
inout pad;
assign data_in = pad;
TRIBUF i_tribuf(
                .A(data_out),
                .OEN(oen),
                .Y(pad)
                );
```

Note: Text in italic and bold is user-defined.

Port Definition

Table 1 Port Definition of TRIBUF			
Name	**Type**	**Width**	**Description**
A	Input	1	Output Signal
OEN	Input	1	Enable signal. Low active.
Y	Output	1	Connect to PAD.

Waveform

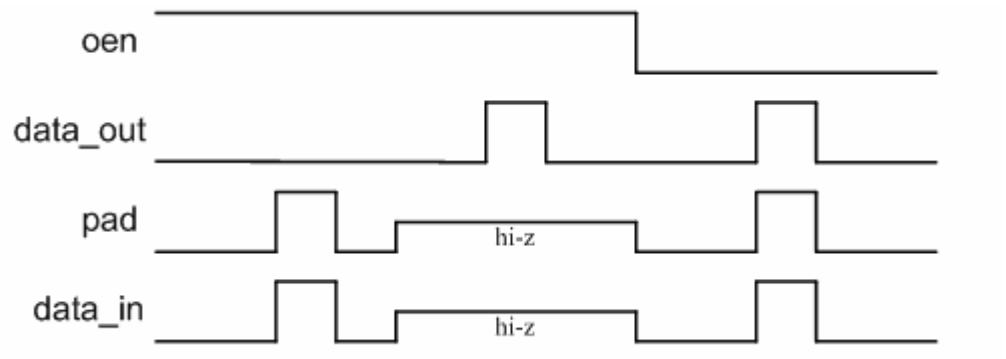

Figure 2 Tri-state I/O Waveform

PLL

Frequency Synthesis

Angelo PLL provides clock synthesis for PLL three output ports. Figure 3 shows the schematic diagram of PLL in frequency synthesis mode. The input clock is divided by a reference divider, N, and is then multiplied by the M feedback factor. The control loop drives the VCO to match clkin × (M / N). Then the output ports have the post-scale counter C0/C1/C2 to divide down the high-frequency VCO.

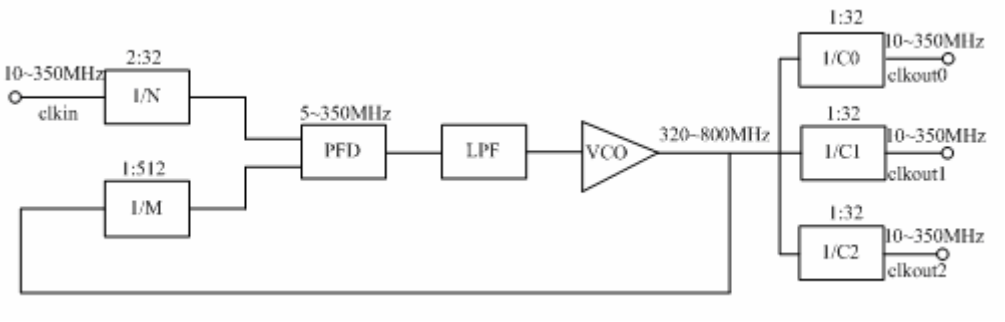

Figure 3 PLL in Frequency Synthesis Mode

The following formulas describe the relationship between output and input.

clkout0 = VCO / C0 = clkin × (M / N) / C0 ;
clkout1 = VCO / C1 = clkin × (M / N) / C1 ;
clkout2 = VCO / C2 = clkin × (M / N) / C2 ;

Table 2 shows the PLL specification in frequency synthesize mode.

Table 2 PLL Specification in Frequency Synthesize Mode

Parameter	Value
N range	2- 32
M range	1- 512
C0/C1/C2 range	1- 32
Input clock frequency range	10MHz - 350MHz
Output clock frequency range	10MHz - 350MHz
VCO output frequency range	320MHz - 800MHz
PFD input frequency range	5MHz - 350MHz

Clock Deskew

Angelo PLL can work at deskew mode which provides zero propagation delay between the source clock and output clock, low clock skew among output clock signals distributed throughout the device, and advanced clock domain control. The deskew feature also supports external loop for a board-level clock serving multiple devices. This is achieved by driving the clkout0 output off-chip to the board and then bringing the clock back in as a feedback clock fbclkin. Figure 4 shows the schematic of PLL in deskew mode. Short feedback means in-chip delay which less than 3ns, and long feedback means off-chip delay which more than 10ns.

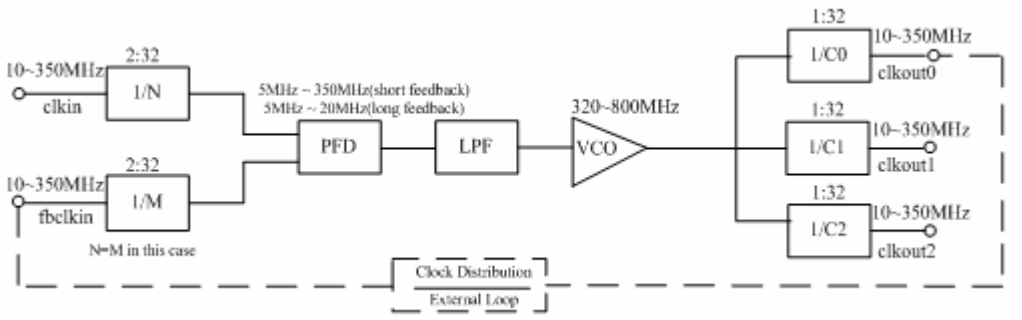

Figure 4 PLL in Deskew Mode

In deskew mode, M must equal to N, and the feedback loop is different from normal frequency synthesis mode. So the formulas are changed as follows.

clkout0 = clkin ;
clkout1 = clkin × C0 / C1 ;
clkout2 = clkin × C0 / C2 ;

Table 3 shows the PLL specification in deskew mode.

Table 3 PLL Specification in Deskew Mode	
Parameter	**Value**
N range	2- 32
M range	M = N
C0/C1/C2 range	1- 32
Input clock frequency range	10MHz - 350MHz
Output clock frequency range	10MHz - 350MHz
VCO output frequency range	320MHz - 800MHz
PFD input frequency range	5MHz - 350MHz (short feedback) 5MHz – 20MHz (long feedback)

The following figure describes an application for board deskew. This circuit ensures the clock driving off-chip device and the clock driving in-chip D flip-flop are consistent.

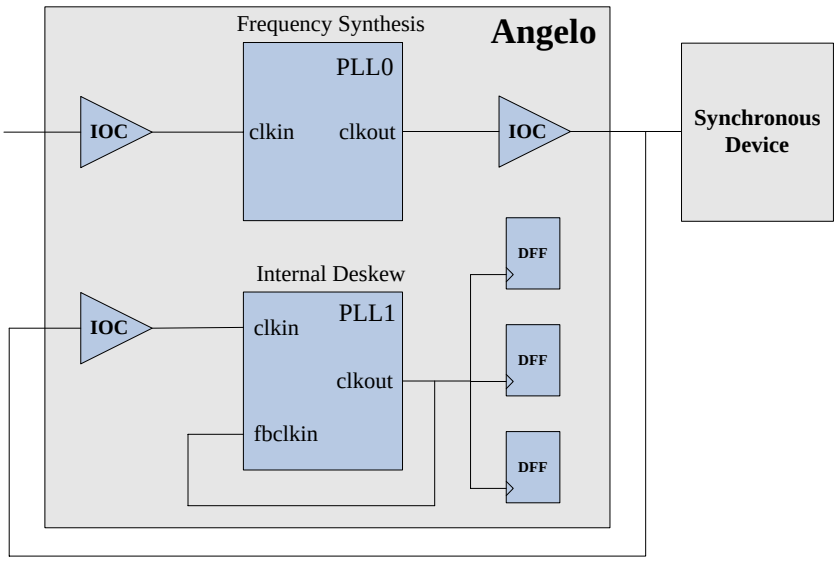

Figure 5 Application for Board Deskew

Phase Shifting

Angelo PLL has the phase shifting capability on output ports clkout1 and clkout2 (clkout0 cannot support this feature). The finest resolution equals 1/10 of the VCO period. Each clock output counter can choose a different phase of the VCO period from 10 taps. A phase-shift range for the entire period of the output clock is implemented by using a combination of the VCO phase output and the post-scale counter starting time. The phase tuning can be only activated when C1 and C2 are set to 1, 2, 4, 8, 16 and 32, so the phase-shift value will be set from 0 to 9, 0 to 19, 0 to 39, 0 to 79, 0 to 159 and 0 to 319 correspondingly.

A phase link feature is also provided for Angelo PLL phase shifting. If you disable this feature, the phase relationship between clkout1, clkout2 and clkout0 are not deterministic. To enable phase link, parameter pll_fsl must be set to 1 and C1, C2 will be forced to set to the same value with C0 inside PLL. As shown in Figure 5, in this situation, the phase relationship between clkout1, clkout2 and clkout0 are linked together.

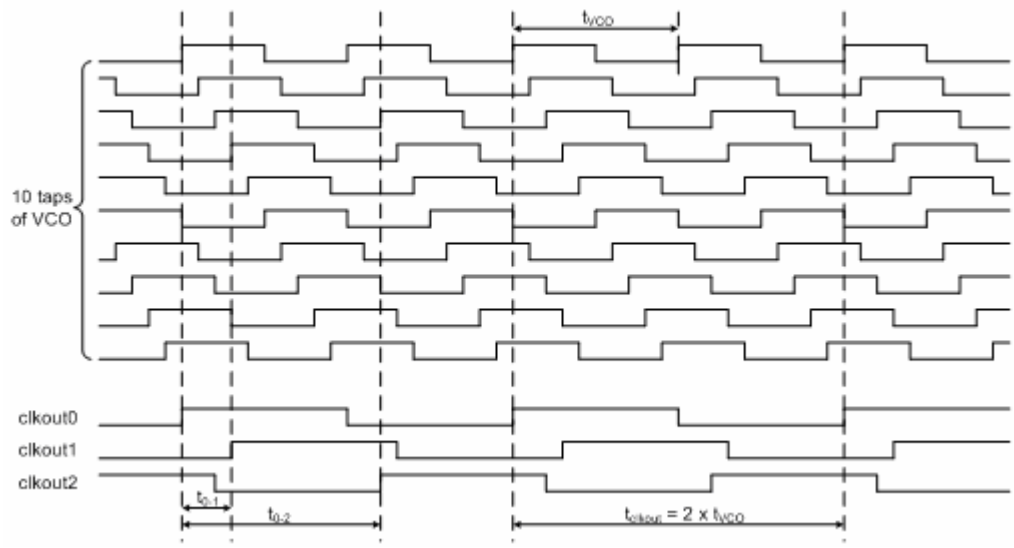

Figure 5 PLL Phase Shifting in Phase Link Mode

Note: C0=C1=C2=2, so phase-shift value is 0-19, value of clkout1 is 3 and value of clkout2 is 12.

Lock Status

The pll_lock output signal indicates that there is a stable clock output. This signal is optional; but it is useful in monitoring the PLL lock process.

Instantiation Template

```
flexera_pll0 i_pll(
                        .clkin(clkin),
                        .fbclkin(fbclkin),
                        .pll_reset(pll_reset),
                        .clkout0(clkout0),
                        .clkout1(clkout1),
                        .clkout2(clkout2),
                        .pll_lock(pll_lock)
                        );
defparam i_pll.pll_pwrdn = 0;
defparam i_pll.pll_deskew = 0;
defparam i_pll.pll_bypass = 0;
defparam i_pll.pll_o0en = 1;
defparam i_pll.pll_o1en = 1;
defparam i_pll.pll_o2en = 1;
defparam i_pll.pll_ph1en = 0;
defparam i_pll.pll_ph2en = 0;
```

defparam ***i_pll***.pll_refdiv = ***1***;
defparam ***i_pll***.pll_fbdiv = ***1***;
defparam ***i_pll***.pll_odiv0 = ***1***;
defparam ***i_pll***.pll_odiv1 = ***1***;
defparam ***i_pll***.pll_odiv2 = ***1***;
defparam ***i_pll***.pll_phsel1 = ***1***;
defparam ***i_pll***.pll_phsel2 = ***1***;
defparam ***i_pll***.pll_fsl = ***0***;

Note: Text in italic and bold is user-defined.

Port Definition

Table 4 Port Definition of flexera_pll0

Name	Type	Width	Description
clkin	Input	1	Reference clock input.
fbclkin	Input	1	Feedback clock input, used in deskew mode.
pll_reset	Input	1	PLL reset signal. High active. When reset, all outputs are low.
clkout0	Output	1	Main clock output.
clkout1	Output	1	First shifted phase clock output. Can run at different frequency in synthesize mode.
clkout2	Output	1	Second shifted phase clock output. Can run at different frequency in synthesize mode.
pll_lock	Output	1	Lock status output. High active.

Parameter Description

Table 5 Parameter Description of flexera_pll0

Parameter	Description
pll_pwrdn	Temporarily invalid.
pll_deskew	Deskew enable. Port fbclkin selected as feedback. 1: deskew mode; 0: normal mode.
pll_bypass	Bypass input clkin to output clkout0/ clkout1/ clkout2 when they are enabled. 1: bypass mode; 0: normal mode.
pll_o0en	Output enable for clkout0. In deskew mode, it should be 1 to enable feedback loop. 1: enable output clkout0; 0: disable output clkout0.
pll_o1en	Output enable for clkout1. 1: enable output clkout1; 0: disable output clkout1.
pll_o2en	Output enable for clkout2. 1: enable output clkout2; 0: disable output clkout2.
pll_ph1en	Phase shift enable for clkout1. 1: enable clkout1 phase shift; 0: disable clkout1 phase shift.
pll_ph2en	Phase shift enable for clkout2. 1: enable clkout2 phase shift; 0: disable clkout2 phase shift.
pll_refdiv	Input reference divider N control.

	Range: 1-32.
pll_fbdiv	Feedback clock divider M control, should be forced to be equal to pll_refdiv in deskew mode. Range: 1-512.
pll_odiv0	Output clkout0 divider C0 control. Range: 1-32.
pll_odiv1	Output clkout1 divider C1 control, should be forced to be equal to pll_odiv0 in phase shift link mode. Range: 1-32.
pll_odiv2	Output clkout2 divider C2 control, should be forced to be equal to pll_odiv0 in phase shift link mode. Range: 1-32.
pll_phsel1	Output clkout1 phase shift control, 9-bit width. [8:5] are used to select VCO output into clkout1 divider, only 10 out of 16 states used, and the value is from 0 to 9. [4:0] are used to select different phases from clkout1 dividers, only 1, 2, 4, 8, 16, 32 divider supported. For divide by 2, 4, 8, 16, 32, [0], [1:0], [2:0], [3:0] and [4:0] are used, respectively, to select different phases from the programmable divider. For divide by 1, [4:0] should be all 0.
pll_phsel2	Output clkout2 phase shift control, 9-bit width. [8:5] are used to select VCO output into clkout2 divider, only 10 out of 16 states used, and the value is from 0 to 9. [4:0] are used to select different phases from clkout2 dividers, only 1, 2, 4, 8, 16, 32 divider supported. For divide by 2, 4, 8, 16, 32, [0], [1:0], [2:0], [3:0] and [4:0] are used, respectively, to select different phases from the programmable divider. For divide by 1, [4:0] should be all 0.
pll_fsl	Phase shift linked between 3 outputs clkout0, clkout1, clkout2: phase relationship will be established between clkout0 and clkout1, or between clkout0 and clkout2. 1: phase shift link mode; 0: normal mode.

EMB9K

Users can instantiate three easy-used wrapper modules about EMB9K, including emb9k_sp (single port), emb9k_sdp (simple dual-port) and emb9k_tdp (true dual-port). All of them are separately described below in detail. For more information please refer to Angelo Family Datasheet.

Single Port EMB9K

Instantiation Template

```
emb9k_sp i_emb9k(
                .clk(clk),
                .a(a),
                .d(d),
                .ce(ce),
                .we(we),
                .clke_q(clke_q),
                .rstn_q(rstn_q),
```

```
                .q(q)
                );
defparam i_emb9k.operation_mode = "single_port";
defparam i_emb9k.address_width = 14;
defparam i_emb9k.data_width = 1;
defparam i_emb9k.we_width = 1;
defparam i_emb9k.output_mode = "register";
defparam i_emb9k.is_clk_q_inverted  = "false";
defparam i_emb9k.init_file = "init_file.dat";
```

Note: Text in italic and bold is user-defined.

Port Definition

Table 6 Port Definition of emb9k_sp			
Name	**Type**	**Width**	**Description**
clk	Input	1	Clock input.
a	Input	address_width	Address input.
d	Input	data_width	Data input.
ce	Input	1	Enable signal. High active. When ce is high and we is low, data read from the memory is available upon the next rising edge of clk. If ce is low, data output retains its value.
we	Input	we_width	Write Enable. High active. Data is written into the memory upon the rising edge of clk when both we and ce are high. The we[0] controls d[7:0] or d[8:0], we[1] controls d[15:8] or d[17:9], we[2] controls d[23:16] or d[26:18] and we[3] controls d[31:24] or d[35:27].
clke_q	Input	1	Clock Enable of output register. High active.
rstn_q	Input	1	Reset of output register. Low active.
q	Output	data_width	Data output.

Parameter Description

Table 7 Parameter Description of emb9k_sp	
Parameter	**Description**
operation_mode	Must be “single_port”.
address_width	Address width. Range: 1-14. Only when data_width is 1, address_width can be 14.
data_width	Data width. Range: 1, 2, 4, 8, 9, 16, 18, 32, 36.
we_width	Write Enable width. Range: 1, 2, 4.
output_mode	“bypass”: data outputs directly without register. “register”: data outputs with register.
is_clk_q_inverted	1: clk connects to output register inversely; 0: clk connects to output register directly.
init_file	Path and file name of initialize data file. For example, “E:\\proj\\init_file.dat”. Refer to the section *Initial File Format*.

Note: $2^{address\_width}$ × data_width cannot over 9Kb, except the situation that address_width is 14 and data_width is 1.

Simple Dual-Port EMB9K

Instantiation Template

```
emb9k_sdp i_emb9k(
                    .clk_w(clk_w),
                    .ce_w(ce_w),
                    .we_w(we_w),
                    .a_w(a_w),
                    .d_w(d_w),
                    .clk_r(clk_r),
                    .clke_r(clke_r),
                    .rstn_r(rstn_r),
                    .ce_r(ce_r),
                    .a_r(a_r),
                    .q_r(q_r)
                    );
defparam i_emb9k.operation_mode = "simple_dual_port";
defparam i_emb9k.portw_address_width = 14;
defparam i_emb9k.portw_data_width = 1;
defparam i_emb9k.portw_we_width = 1;
defparam i_emb9k.portr_address_width = 14;
defparam i_emb9k.portr_data_width = 1;
defparam i_emb9k.portr_output_mode = "register";
defparam i_emb9k.is_clk_r_inverted = "false";
defparam i_emb9k.init_file = "init_file.dat";
```

Note: Text in italic and bold is user-defined.

Port Definition

Table 8 Port Definition of emb9k_sdp

Name	Type	Width	Description
clk_w	Input	1	Clock of port Write.
ce_w	Input	1	Enable of port Write. High active.
we_w	Input	portw_we_width	Write Enable of port Write. High active. Data is written into the memory upon the rising edge of clk_w when both we_w and ce_w are high. The we_w[0] controls d_w[7:0] or d_w[8:0], we_w[1] controls d_w[15:8] or d_w[17:9], we_w[2] controls d_w[23:16] or d_w[26:18] and we_w[3] controls d_w[31:24] or d_w[35:27].
a_w	Input	portw_address_width	Address of port Write.
d_w	Input	portw_data_width	Data input of port Write.

clk_r	Input	1	Clock of port Read.
clke_r	Input	1	Clock Enable of output register of port Read. High active.
rstn_r	Input	1	Reset of output register of port Read. Low active.
ce_r	Input	1	Enable of port Read. High active. When ce_r is high, data read from the memory is available upon the next rising edge of clk_r. If ce_r is low, data output retains its value.
a_r	Input	portr_address_width	Address of port Read.
q_r	Output	portr_data_width	Data output of port Read.

Parameter Description

Table 9 Parameter Description of emb9k_sdp

Parameter	Description
operation_mode	Must be “simple_dual_port”.
portw_address_width	Address width of port Write. Range: 1-14. Only when portw_data_width is 1, portw_address_width can be 14.
portw_data_width	Data width of port Write. Range: 1, 2, 4, 8, 9, 16, 18, 32, 36.
portw_we_width	Write Enable width of port Write. Range: 1, 2, 4.
portr_address_width	Address width of port Read. Range: 1-14. Only when portr_data_width is 1, portr_address_width can be 14.
portr_data_width	Data width of port Read. Range: 1, 2, 4, 8, 9, 16, 18, 32, 36.
portr_output_mode	“bypass”: data q_r outputs directly without register. “register”: data q_r outputs with register.
is_clk_r_inverted	1: clk_r connects to output register inversely; 0: clk_r connects to output register directly.
init_file	Path and file name of initialize data file. For example, “E:\\proj\\init_file.dat”. Refer to the section *Initial File Format*.

Notes:

(1) $2^{portw\_address\_width}$ × portw_data_width and $2^{portr\_address\_width}$ × portr_data_width cannot over 9Kb, except the situation that portw_address_width is 14 and portw_data_width is 1, or portr_address_width is 14 and portr_data_width is 1.

(2) If portw_data_width is 1 (portw_address_width is not 14), 2, 4, 8, 16, or 32, portr_data_width also should be one of these 6 modes. If portw_data_width is 1(portw_address_width is 14), 9, 18, or 36, portr_data_width also should be one of these 4 modes. If portw_data_width is not any one of above 10 modes, portr_data_width should be the same as portw_data_width.

True Dual-Port EMB9K

Instantiation Template

```
emb9k_tdp i_emb9k(
                .clka(clka),
```

```
        .cea(cea),
        .wea(wea),
        .aa(aa),
        .da(da),
        .rstn_qa(rstn_qa),
        .clke_qa(clke_qa),
        .qa(qa),
        .clkb(clkb),
        .ceb(ceb),
        .web(web),
        .ab(ab),
        .db(db),
        .rstn_qb(rstn_qb),
        .clke_qb(clke_qb),
        .qb(qb)
        );
    defparam i_emb9k.operation_mode = "true_dual_port";
    defparam i_emb9k.porta_address_width = 14;
    defparam i_emb9k.porta_data_width = 1;
    defparam i_emb9k.porta_we_width = 1;
    defparam i_emb9k.porta_output_mode = "register";
    defparam i_emb9k.is_clk_qa_inverted = "false";
    defparam i_emb9k.portb_address_width = 14;
    defparam i_emb9k.portb_data_width = 1;
    defparam i_emb9k.portb_we_width = 1;
    defparam i_emb9k.portb_output_mode = "register";
    defparam i_emb9k.is_clk_qb_inverted = "false";
    defparam i_emb9k.init_file = "init_file.dat";
```

Note: Text in italic and bold is user-defined.

Port Definition

Table 10 Port Definition of emb9k_tdp

Name	Type	Width	Description
clka	Input	1	Clock of port A.
cea	Input	1	Enable of port A. High active. When cea is high and wea is low, data read from the memory is available upon the next rising edge of clka. If cea is low, data output retains its value.
wea	Input	porta_we_width	Write Enable of port A. High active. Data is written into the memory upon the rising edge of clka when both wea and cea are high. The wea[0] controls da[7:0] or da[8:0], and wea[1] controls da[15:8] or da[17:9].
aa	Input	porta_address_width	Address of port A.
da	Input	porta_data_width	Data input of port A.
rstn_qa	Input	1	Reset of port A output register. Low active.

clke_qa	Input	1	Clock Enable of port A output register. High active.
qa	Output	porta_data_width	Data output of port A.
clkb	Input	1	Clock of port B.
ceb	Input	1	Enable of port B. High active. When ceb is high and web is low, data read from the memory is available upon the next rising edge of clkb. If ceb is low, data output retains its value.
web	Input	portb_we_width	Write Enable of port B. High active. Data is written into the memory upon the rising edge of clkb when both web and ceb are high. The web[0] controls db[7:0] or db[8:0], and web[1] controls db[15:8] or db[17:9].
ab	Input	portb_address_width	Address of port B.
db	Input	portb_data_width	Data input of port B.
rstn_qb	Input	1	Reset of port B output register. Low active.
clke_qb	Input	1	Clock Enable of port B output register. High active.
qb	Output	portb_data_width	Data output of port B.

Parameter Description

Table 11 Parameter Description of emb9k_tdp

Parameter	Description
operation_mode	Must be “true_dual_port”.
porta_address_width	Address width of port A. Range: 1-14. Only when porta_data_width is 1, porta_address_width can be 14.
porta_data_width	Data width of port A. Range: 1, 2, 4, 8, 9, 16, 18.
porta_we_width	Write Enable width of port A. Range: 1-2.
porta_output_mode	“bypass”: data qa outputs directly without register. “register”: data qa outputs with register.
is_clk_qa_inverted	1: clka connects to port A output register inversely; 0: clka connects to port A output register directly.
portb_address_width	Address width of port B. Range: 1-14. Only when portb_data_width is 1, portb_address_width can be 14.
portb_data_width	Data width of port B. Range: 1, 2, 4, 8, 9, 16, 18.
portb_we_width	Write Enable width of port B. Range: 1-2.
portb_output_mode	“bypass”: data qb outputs directly without register. “register”: data qb outputs with register.
is_clk_qb_inverted	1: clkb connects to port B output register inversely; 0: clkb connects to port B output register directly.
init_file	Path and file name of initialize data file. For example, “E:\\proj\\init_file.dat”. Refer to the section *Initial File Format*..

Notes:

(1) $2^{porta\_address\_width}$ × porta_data_width and $2^{portb\_address\_width}$ × portb_data_width cannot over 9Kb, except the situation that porta_address_width is 14 and porta_data_width is 1, or portb_address_width is 14 and portb_data_width is 1.

(2) If porta_data_width is 1(porta_address_width is not 14), 2, 4, 8, or 16, portb_data_width also should be one of these 5 modes. If porta_data_width is 1(porta_address_width is 14), 9, or 18, portb_data_width also should be one of these 3 modes. If porta_data_width is not any one of above 8 modes, portb_data_width should be the same as porta_data_width.

Initial File Format

The following paragraph shows the format of the EMB9K initial file. Note the texts in italic and bold are only for helping comprehension here and cannot appear in initial file.

```
//width=8          #specify data width
80                 #data of address 0
1B                 #data of address 1
F3                 #data of address 2
7A                 #data of address 3
......
```

First line of this file indicates data width, in which the "//" specifies that this line is not data, and the width can be 1, 2, 4, 8, 9, 16, 18, 32 or 36. The following continuous lines are initial data in hexadecimal format start from address 0. If data is wider than specified data width, for instance, "//width=9" and data is "F8A" (12'b1111_1000_1010), data "18A" (9'b1_1000_1010) will be written and the higher 3'b111 will be ignored.

About Agate Logic

Agate Logic is the global pioneer and leader of the innovative Adaptable Programmable Gate Array (APGA) technologies. The company offers a full spectrum of programmable logic devices, software design tools, intellectual property (IP) and design services. Focusing on multiple applications such as telecommunication equipments, industrial control systems and consumer products, we use the Chinese leading foundry partner, SMIC, to manufacture our chips to offer solutions tailored for the market in China.

Technical Support Assistance

Tel: +86 10 82150100

E-mail: support@agatelogic.com

Website: www.agatelogic.com.cn