Parameterized Macrocell Quick Reference Guide

Application Note

MACROCELL_ACC

Parameterized accumulator

Port

Name	Type	Required	Width	Description
clock	input	yes	1	clock
data	input	yes	width_data	input data
result	output	yes	width_result	result
add_sub	input	no	1	0 for add, 1 for sub
clken	input	no	1	clock enable, high active, if not used, connect to 1
cin	input	no	1	carry input, if not used, connect to 0
aclr	input	no	1	asynchronous reset, low active, if not used, connect to 1
sload	input	no	1	synchronous load, high active, if not used, connect to 0
cout	output	no	1	carry output
overflow	output	no	1	over flow

Parameter

Name	Value	Required	Description
width_data	0<width_data<=width_result<=64	yes	the width of data
width_result	0<width_data<=width_result<=64	yes	the width of result
is_signed	1 or 0	yes	1 for signed, 0 for unsigned
no_cin	1 or 0	yes	1 for no cin port
no_add_sub	1 or 0	yes	1 for no add_sub control
no_sload	1 or 0	yes	1 for no sload control
no_cout	1 or 0	yes	1 for no cout output
no_overflow	1 or 0	yes	1 for no overflow output

MACROCELL_DIV

Parameterized divider

Port

Name	Type	Required	Width	Description
clock	input	no	1	clock
clken	input	no	1	clock enable, if not used, connect to 1
alcr	input	no	1	asynchronous reset, if not used, connect to 1
start	input	no	1	one pulse signal, for serial divider start
numer	input	yes	width_num	dividend
denom	input	yes	width_de	divisor
quotient	output	yes	width_num	quotient
remain	output	yes	width_de	remainder

Parameter

Name	Value	Required	Description
width_num	width_num>2	Yes	The width of numer, Max is 32
width_de	0<width_de<=width_num	Yes	The width of denom, Max is 32
is_serial	1 or 0	Yes	1 for serial divider implement the result is send out delay “width_num” clocks after “start” signal. 0 for pure combinational implement

MACROCELL_ADD_SUB

Parameterized adder/subtractor

Port

Name	Type	Required	Width	Description
dataa	input	yes	width_data	data a input
datab	input	yes	width_data	data b input
cin	input	no	1	carry or borrow in
add_sub	input	no	1	1 for sub, 0 for add

clock	input	no	1	clock
aclr	input	no	1	asynchronous reset, low active
clken	input	no	1	clock enable, high active
result	input	yes	width_data	result = dataa + datab + cin result = dataa – datab – cin result = (add_sub)?(dataa – datab):(dataa + datab);
cout	output	no	1	carry out or borrow out
overflow	output	no	1	overflow

Parameter

Name	Value	Required	Description
width_data	64>= width_data >0	yes	1~64
is_signed	0 or 1	yes	1 for signed number
num_pipeline	>=0 <=width_data	yes	pipeline number, 0 for no pipeline
cell_func	0, 1, 2	yes	0 add， 1 sub 2 using add_sub control

MACROCELL_DEC

Parameterized decoder

Port

Name	Type	Required	Width	Description
clock	input	no	1	clock
clken	input	no	1	clock enable, if not used, connect to 1
aclr	input	no	1	asynchronous reset, if not used, connect to 1
data	input	yes	width_data	data input
eq	output	yes	2width_data	decoder wires

Parameter

Name	Value	Required	Description
width_data	1<=width_data<=8	Yes	The width of data
num_pipeline	0,1,2	Yes	pipeline stage number

MACROCELL_CMP

Parameterized comparator

Port

Name	Type	Required	Width	Description
dataa	input	yes	width_data	data A
datab	input	yes	width_data	data B
clock	input	no	1	clock
clken	input	no	1	clock enable
aclr	input	no	1	asynchronous reset
alb	output	no	1	alb =1 if A<B
ageb	output	no	1	ageb =1 if A>=B
aeb	output	no	1	aeb=1 if A==B
aneb	output	no	1	aneb=1 if A!=B
agb	output	no	1	agb=1 if A>B
aleb	output	no	1	aleb=1 if A<=B

Parameter

Name	Value	Required	Description
width_data	1<=width_data<=64	yes	The width of dataa and datab. Max is 64
is_signed	0,1	yes	1 for signed number compare 0 for unsigned number compare
num_pipeline	0,1,2	yes	pipeline stage
cell_func	[5:0]	yes	cell_func[0]=1 for agb output enable cell_func[1]=1 for alb output enable cell_func[2]=1 for aeb output enable cell_func[3]=1 for aleb output enable cell_func[4]=1 for aneb output enable cell_func[5]=1 for ageb output enable

MACROCELL_MUL

Parameterized multiplier

Port

Name	Type	Required	Description
A[N-1:0]	Input	yes	A operant input bus, N bits wide(parallel multipliers only)
B[N-1:0]	Input	yes	B operant input bus, N bits wide
clk	Input	no	Rising-edge clock input
ce	Input	no	Clock Enable
clr	Input	no	Asynchronous Clear
P	Output	yes	Product Output

Parameter

Name	Value	Description	Default Value
PortA_Width	2 - 32	Input DataA Width	18
PortA_Type	Signed/unsigned	Input DataA type	Signed
PortB_Width	2 - 32	Input DataB Width	18
PortB_Type	Signed/unsigned	Input DataB type	Signed
Constant_value	1 – (2^32-1),signed/unsigned(Note 1)	PortA as constant	NONE
Constant_set	0,1(Note 2)	Indicate using the constant multiplier	0
Pipe_stage	0 – 4(For different data widths will have different ranges)	Indicate using the pipe line multiplier	0

Note 1: The constant_value share the parameters from the PortA. If set the consant_set, the constant_value width and type are defined by the PortA_Width and PortA_Type.

Note 2: Constant_set = 1, means using the constant multiplier; 0 means using the parallel multiplier.

Instantiation templates

```
module myAccumulator(
    data,          //data input
    cin,           //carry in
    add_sub,       //add/sub control:1 for sub;0 for add
    clock,         //clock
    sload,         //synchronous load, high active
    clken,         //clock enable, high active
```

```
    aclr,                   //asynchronous reset, low active
    result,                 //result output
    cout,                   //carry out
    overflow                //over flow flag
    );

parameter dwidth   = 8;   //width for data
parameter rwidth   = 8;   //width for resul

input clock;
input [dwidth-1:0] data;
input cin;
input add_sub;
input sload;
input clken;
input aclr;

output [rwidth-1:0] result;
output cout;
output overflow;


macrocell_acc u_acc(
    .cin        (cin),
    .data       (data),
    .add_sub    (add_sub),
    .clock      (clock),
    .sload      (sload),
    .clken      (clken),
    .aclr       (aclr),
    .result     (result),
    .cout       (cout),
    .overflow   (overflow)
    );

    defparam u_acc.width_data   = dwidth;       // width for data
    defparam u_acc.width_result = rwidth;   // width for resul
    defparam u_acc.is_signed    = 1;            // 1 for signed number,0 for unsigned number
    defparam u_acc.no_cin       = 0;            // 1 for no carry in
    defparam u_acc.no_add_sub   = 0;            // 1 for no add_sub control
    defparam u_acc.no_sload         = 0;            // 1 for no sload signal control
    defparam u_acc.no_cout      = 0;            // 1 for no carry output
    defparam u_acc.no_overflow = 0;         // 1 for no over flow output
```

```
endmodule

module myDivider(
      clock,                  //clock
      aclr,                   //asynchronous reset, low active
      clken,                  //clock enable, high active
      start,                  //one pulse signal, for serial divider start
      numer,                  //dividend
      denom,                  //divisor
      quotient,               //quotient
      remain                  //remainder
      );

parameter numwidth      = 16; //width for numer
parameter dewidth  = 8;       //width for denom

input clock;
input aclr;
input clken;
input start;
input [numwidth-1:0] numer;
input [dewidth-1:0] denom;

output [numwidth-1:0] quotient;
output [dewidth-1:0] remain;


macrocell_div u_div(
      .clock          (clock),
      .aclr       (aclr),
      .clken          (clken),
      .start      (start),
      .numer          (numer),
      .denom          (denom),
      .quotient  (quotient),
      .remain         (remain),
      );

      defparam u_div.width_num    = numwidth; //width for numer
      defparam u_div.width_de     = dewidth;    //width for denom
      defparam u_div.is_serial    = 1;          //1 for serial divider implement
                                  //the result is send out "width_num" clocks after "start" signal.
                                  //0 for pure combinational implement
```

```
endmodule

module myAdd(
        dataa,                  //data a input
        datab,                  //data b input
        cin,                    //carry in or borrow in
        add_sub,                //add/sub control:1 for sub;0 for add
        clock,                  //clock
        clken,                  //clock enable, high active
        aclr,                   //asynchronous reset, low active
        result,                 //result output,
                                  //result=dataa+datab+cin
                                  //result=dataa-datab-cin
                                  //result=(add_sub)?(dataa-datab):(dataa+datab);
        cout,                   //carry out
        overflow                //over flow flag
        );

parameter dwidth    = 9;//width for dataa and datab

input [dwidth-1:0] dataa;
input [dwidth-1:0] datab;
input cin;
input add_sub;
input clock;
input clken;
input aclr;

output [dwidth-1:0] result;
output cout;
output overflow;


macrocell_add_sub u_addsub(
        .dataa              (dataa),
        .datab              (datab),
        .cin          (cin),
        .add_sub    (add_sub),
        .clock              (clock),
        .aclr         (aclr),
        .clken              (clken),
        .result             (result),
```

```
    .cout       (cout),
    .overflow (overflow)
    );

    defparam u_addsub.width_data     = dwidth; //width for dataa and datab
    defparam u_addsub.is_signed      = 1;          //1 for signed number,0 for unsigned number
    defparam u_addsub.num_pipeline = 6;       //pipeline number,0 for no pipeline
    defparam u_addsub.cell_func      = 0   ;      //0 for add;1 for sub;2 for using add_sub
control
                                                    //cell_func=0,result=dataa+datab+cin
                                                    //cell_func=1,result=dataa-datab-cin

//cell_func=2,result=(add_sub)?(dataa-datab):(dataa+datab);

Endmodule


module myDecoder(
    data,                   //data input
    clock,                  //clock
    aclr,                   //asynchronous reset, low active
    clken,                  //clock enable, high active
    eq                        //decoder wires,eq[data]=1
    );

parameter dwidth    = 8; //width for data

input [dwidth-1:0] data;
input clock;
input aclr;
input clken;

output [255:0] eq;


macrocell_dec u_dec(
    .data       (data),
    .clock          (clock),
    .aclr       (aclr),
    .clken          (clken),
    .eq             (eq)
    );

    defparam u_dec.width_data    = dwidth; //width for data
```

```
    defparam u_dec.num_pipeline = 1;        //pipeline number,0 for no pipeline

endmodule

module myComparator(
    dataa,                  //data a input
    datab,                  //data b input
    clock,                  //clock
    aclr,                   //asynchronous reset, low active
    clken,                  //clock enable, high active
    alb,                    //alb=1 if dataa < datab
    aeb,                    //aeb=1 if dataa == datab
    agb,                    //agb=1 if dataa > datab
    aleb,                   //aleb=1 if dataa <= datab
    aneb,                   //aneb=1 if dataa != datab
    ageb                    //ageb=1 if dataa >= datab
    );

parameter dwidth    = 10; //width for dataa and datab

input [dwidth-1:0] dataa
input [dwidth-1:0] datab;
input clock;
input aclr;
input clken;

output alb;
output aeb;
output agb;
output aleb;
output aneb;
output ageb;

macrocell_cmp u_cmp(
    .dataa          (dataa),
    .datab          (datab),
    .clock          (clock),
    .aclr       (aclr),
    .clken          (clken),
    .alb        (alb),
    .aeb        (aeb),
    .agb        (agb)
    .aleb       (aleb),
    .aneb           (aneb),
```

```
    .ageb           (ageb)
    );

    defparam u_cmp.width_data       = dwidth;       //width for dataa and datab
    defparam u_cmp.is_signed        = 0;            //1 for signed number,0 for unsigned number
    defparam u_cmp.num_pipeline     = 0;            //pipeline number,0 for no pipeline
    defparam u_cmp.cell_func        = 6'b111111; //cell_func[0]=1 for agb output enable
                                                 //cell_func[1]=1 for alb output enable
                                                 //cell_func[2]=1 for aeb output enable
                                                 //cell_func[3]=1 for aleb output enable
                                                 //cell_func[4]=1 for aneb output enable
                                                 //cell_func[5]=1 for ageb output enable


endmodule
```

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