Stratix MAC WYSIWYG Description

Version 1.7
January 4, 2007

By
Altera Corporation

Table of Contents:

1 MAC WYSIWYG primitive

This document describes the WYSIWYG primitive for the Stratix™ MAC. More information about the MAC in Stratix can be found at http://www.altera.com/products/devices/stratix/

1.1 Stratix MAC Multiplier

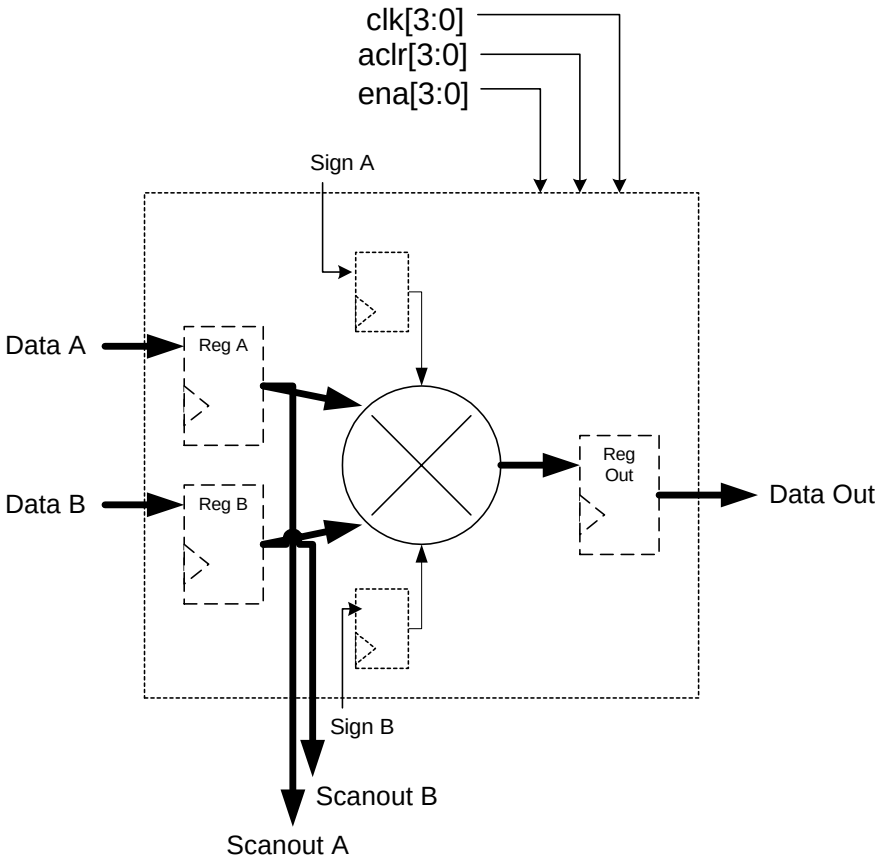

Figure 1: Stratix MAC Multiplier

1.1.1 MAC Multiplier Cell Primitives

```
stratix_mac_mult <mac_mult_name>
(
        .dataa(<data_a source bus>),
        .datab(<data_b source bus>),
        .signa(<signed/unsigned a source>),
        .signb(<signed/unsigned b source>),
        .clk(<clock source bus>),
        .aclr(<asynchronous clear source bus>),
        .ena(<clock enable source bus>),

        .dataout(<data output bus>),
```

```
        .scanouta(<scan output bus for data a>),
        .scanoutb(<scan output bus for data b>),

        .observabledataa_regout(<observable bus for data a>),
        .observabledatab_regout(<observable bus for data b>),

        .observablesigna_regout(<observable port for sign a>),
        .observablesignb_regout(<observable port for sign b>),
    );
    defparam <mac_mult_name>.dataa_width = <dataa width>;
    defparam <mac_mult_name>.datab_width = <datab width>;
    defparam <mac_mult_name>.dataa_clock = <clk for data_a>;
    defparam <mac_mult_name>.datab_clock = <clk for data_b>;
    defparam <mac_mult_name>.signa_clock = <clk for sign_a>;
    defparam <mac_mult_name>.signb_clock = <clk for sign_b>;
    defparam <mac_mult_name>.output_clock = <clk for output>;
    defparam <mac_mult_name>.dataa_clear = <aclr for data_a>;
    defparam <mac_mult_name>.datab_clear = <aclr for data_b>;
    defparam <mac_mult_name>.signa_clear = <aclr for sign_a>;
    defparam <mac_mult_name>.signb_clear = <aclr for sign_b>;
    defparam <mac_mult_name>.output_clear = <aclr for output>;

    // simulation-only parameters
    defparam <mac_mult_name>.dataout_width = <width of dataout port>;
```

1.1.2 MAC Multiplier Cell Input Ports

<*mac_mult_name*>**:** is the unique identifier for the MAC multiplier. This is any identifier name which is legal for the given description language (e.g. Verilog, VHDL, AHDL, etc.). This field is required.

.dataa(<*data_a source bus*>**), .datab(**<*data_b source bus*>**):** are the two input buses for the multiplier. The inputs can be registered by specifying the dataa/b_clock option. Each input can be of a different width, however internally this is implemented by tying unused bits to GND. Note that the multiplier input buses may be swapped in some cases to improve speed and fitting. Note also that the inputs can be driven by the scanouta/b bus of the previous multiplier through dedicated internal routing. However, regular routing can also be used in certain cases as is outlined in the description of the scanouts. This port is required.

.signa(<*signed/unsigned a source*>***),* .signb(**<*signed/unsigned b source*>***)*:** designates the two sign signals for each input of the multiplier. High implies the input is a signed integer (i.e. MSB bit is used as sign bit), low implies the input is an unsigned integer. Each input can have a different signed/unsigned value since the multiplier will take this into account. This can be set to a constant value by tying the input to VCC or GND. The signal can also be registered by specifying the signa/b_clock option. This port is not required and defaults to VCC if left unconnected.

.clk(<*clock source bus*>**):** designates the bus of four clock inputs that drive the multiplier. Every register bank in the multiplier can independently choose which one of these four clocks can drive the register. However, every clock can only be

enabled by the corresponding clock enable signal of the same index – so there can only be a maximum of four clk/ena pairs. This port must be connected if it is specified to be used by one of the parameters, otherwise it must not be connected.

.aclr(<*aclr source bus*>**):** designates the bus of four asynchronous clear inputs that drive the multiplier. Every register bank in the multiplier can independently choose which one of these four signals can drive the register. Unlike the clock enable signal, the aclr signal is not tied to any particular clock (e.g. can mix aclr[0] with clk[3]). This port must be connected if it is specified to be used by one of the parameters, otherwise it must not be connected.

.ena(<*clock enable source bus*>**):** designates the bus of four clock enable inputs that drive the multiplier. Each one of the four clock signals is enabled by the corresponding ena signal. This means that there can be a maximum of four pairs of clk/ce signals and not the sixteen combinations possible if they were completely independent. The reason for this is because the clock enable signal enables and disables the clock signal at the entry to the MAC block, and not at the register. This port must be connected if it is specified to be used by one of the parameters, otherwise it must not be connected.

1.1.3 MAC Multiplier Cell Output Ports

.dataout(<*data output bus*>**)**: the output bus of the multiplier (could be registered if using output registers). Note that this bus can only feed the data inputs of a stratix_mac_out atom through dedicated internal routing – it is not visible external to the MAC. This output must be connected – it cannot be left unconnected. The outputs can be registered by specifying the output_clock option.

.scanouta(<*scan output a bus*>**), .scanoutb(**<*scan output b bus*>**):** the scan outputs from the corresponding inputs (basically the output of the input registers if the inputs are registered, or just the inputs themselves if they are not). This bus can use internal routing to drive the dataa/b bus of the next adjacent multiplier, or it can drive external routing directly. Note that only the last multiplier in a MAC can drive external routing, which places some restrictions on the MAC when used as will be described later. Note also that this output is useless if configuring a 36-bit multiplier. This output can be left unconnected if not used.

.observabledataa_regout(<*observable bus*>***),* .observabledatab_regout(**<*observable bus*>***)*:** assigns names for the observable bus for the data inputs. These cannot be connected to other routing and are used for simulation and timing purposes only. By connecting wires to the observable bus, the user is able to rename the observable bus to the name of the connected wires.

.observablesigna_regout(<*observable port*>***),* .observablesignb_regout(**<*observable port*>**):** assigns a name for the observable port for the two sign ports. The observable port will only be created if its corresponding port is registered. These cannot be connected to other routing and are used for simulation and timing purposes only. By connecting wires to the observable ports, the user is able to rename the observable ports to the name of the connected wires. **This feature has not been implemented yet for these ports.**

1.1.4 MAC Multiplier Cell Modes

<*dataa width*>, <*datab width*> is an integer in the range *{1, 2, …, 18}*. It specifies the width of each operand of the multiplier, as well as the width of the corresponding scanout bus. The combination of the dataa width and the datab width implies the output width: the output width is the sum of both input widths (e.g. for dataa_width = 13 and datab_width = 15, dataout_width = 28). Internally there are only 9-bit and 18-bit multipliers, so multipliers of smaller width are implemented by tying the unused low-order bits to 0 (not the high-order bits since the MSB bit is still needed when doing signed multiplication). To implement a 36-bit multiplier, four 18-bit multipliers are used as will be described later. *This field is required.*

<*clk for data_a*>, <*clk for data_b*>, <*clk for sign_a*>, <*clk for sign_b*>, <*clk for output*> is one of *{none, 0, 1, 2, 3}*. Specifies which clock signal, if any, drives the given register, where the number indicates which clk signal (from clk[0] to clk[3]) should be used. If "none" is specified, then the corresponding signal is not registered. Otherwise, the corresponding signal will be registered and will be driven by the indicated clock, which must be connected. Additionally, this parameter also specifies which clock enable signal will be used. This forces the use of ena[2] if clk[2] is used for example. If the clock enable signal is not used, it must be tied to VCC. This field is not required and defaults to "none" if unspecified.

<*aclr for data_a*>, <*aclr for data_b*>, <*aclr for sign_a*>, <*aclr for sign_b*>, <*aclr for output*> is one of *{none, 0, 1, 2, 3}*. Specifies which asynchronous clear signal, if any, drives the given register, where the number indicates which aclr signal (from aclr[0] to aclr[3]) should be used. If the option is not "none", then the register's clk parameter cannot be "none", and the corresponding signal's register will be cleared by the given aclr signal. However, if the option is "none", then the register's clk parameter must also be "none", and the corresponding signal is not registered. To disable the aclr, this signal should be tied to GND. This field is required if a clk is specified, and must be "none" if a clk is not specified.

<*width of dataout port*> specifies the width of the dataout port. This is redundant information since the width is derived from the input port widths, but is needed for sim models that cannot do the calculation. This is only a simulation parameter and is ignored by the compiler.

1.1.5 MAC Multiplier Cell Polarities and Default Values

All signals are active high. All inputs have programmable inverts, however if the data input is driven by a scanout signal through dedicated routing, no programmable inverts can be used.

1.2 Stratix MAC Output

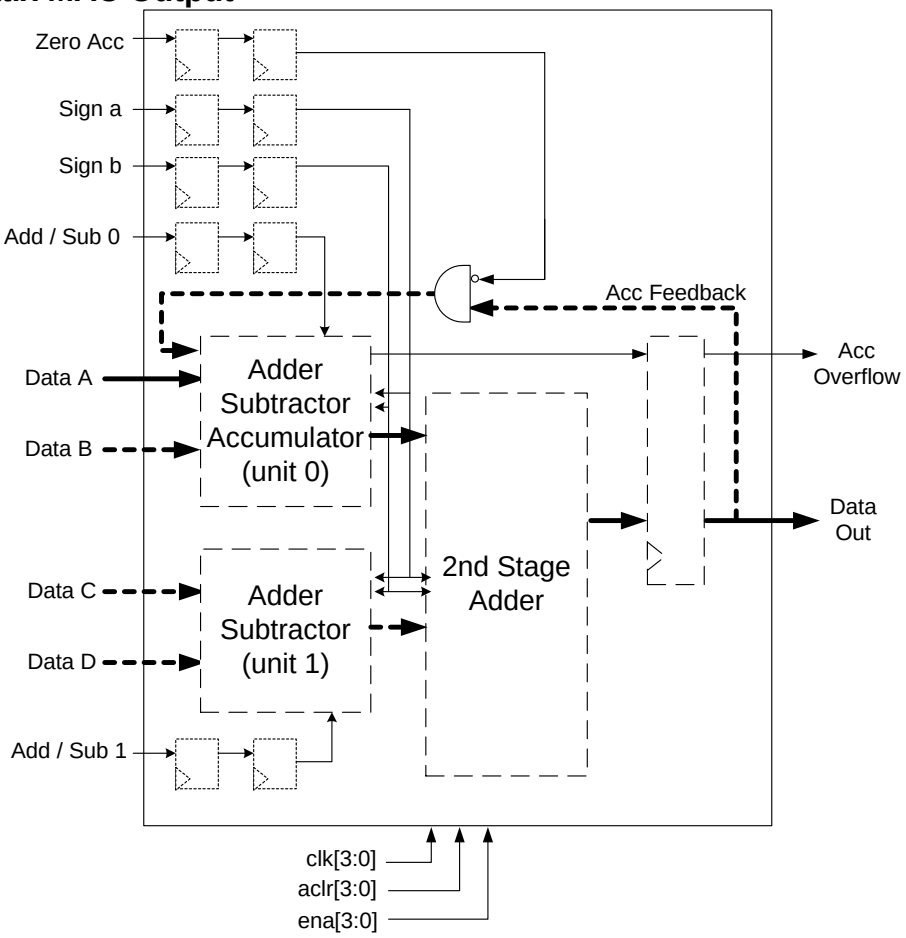

Figure 2: Stratix MAC Output

1.2.1 MAC Output Cell Primitives

```
stratix_mac_out <mac_out_name>
(
        .dataa(<data_a source bus>),
        .datab(<data_b source bus>),
        .datac(<data_b source bus>),
        .datad(<data_b source bus>),
        .zeroacc(<zero accumulator source>),
        .addnsub0(<add or subtract 0 source>),
        .addnsub1(<add or subtract 1 source>),
        .signa(<signed/unsigned a source>),
        .signb(<signed/unsigned b source>),
        .clk(<clock source bus>),
        .aclr(<asynchronous clear source bus>),
        .ena(<clock enable source bus>),

        .dataout(<data output bus>),
        .accoverflow(<accumulator overflow signal>),

        .observablezeroacc_regout(<observable port for zeroacc>),
```

```
        .observableaddnsub0_regout(<observable port for addnsub0>),
        .observableaddnsub1_regout(<observable port for addnsub1>),
        .observablesigna_regout(<observable port for signa>),
        .observablesignb_regout(<observable port for signb>),

        .observablezeroacc_pipeline_regout
                    (<observable pipeline port for zeroacc>),
        .observableaddnsub0_pipeline_regout
                    (<observable pipeline port for addnsub0>),
        .observableaddnsub1_pipeline_regout
                    (<observable pipeline port for addnsub1>),
        .observablesigna_pipeline_regout
                    (<observable pipeline port for signa>),
        .observablesignb_pipeline_regout
                    (<observable pipeline port for signb>),

);
defparam <mac_out_name>.operation_mode = <operation mode>;
defparam <mac_out_name>.dataa_width = <dataa width>;
defparam <mac_out_name>.datab_width = <datab width>;
defparam <mac_out_name>.datac_width = <datac width>;
defparam <mac_out_name>.datad_width = <datad width>;
defparam <mac_out_name>.addnsub0_clock = <clk for addnsub0>;
defparam <mac_out_name>.addnsub1_clock = <clk for addnsub1>;
defparam <mac_out_name>.zeroacc_clock = <clk for zeroacc>;
defparam <mac_out_name>.signa_clock = <clk for sign_a>;
defparam <mac_out_name>.signb_clock = <clk for sign_b>;
defparam <mac_out_name>.output_clock = <clk for output>;
defparam <mac_out_name>.addnsub0_clear = <aclr for addnsub0>;
defparam <mac_out_name>.addnsub1_clear = <aclr for addnsub1>;
defparam <mac_out_name>.zeroacc_clear = <aclr for zeroacc>;
defparam <mac_out_name>.signa_clear = <aclr for sign_a>;
defparam <mac_out_name>.signb_clear = <aclr for sign_b>;
defparam <mac_out_name>.output_clear = <aclr for output>;
defparam <mac_out_name>.addnsub0_pipeline_clock = <clk for
addnsub0 pipeline>;
defparam <mac_out_name>.addnsub1_pipeline_clock = <clk for
addnsub1 pipeline>;
defparam <mac_out_name>.zeroacc_pipeline_clock = <clk for zeroacc
pipeline>;
defparam <mac_out_name>.signa_pipeline_clock = <clk for sign_a
pipeline>;
defparam <mac_out_name>.signb_pipeline_clock = <clk for sign_b
pipeline>;
defparam <mac_out_name>.addnsub0_pipeline_clear = <aclr for
addnsub0 pipeline>;
defparam <mac_out_name>.addnsub1_pipeline_clear = <aclr for
addnsub1 pipeline>;
defparam <mac_out_name>.zeroacc_pipeline_clear = <aclr for
zeroacc pipeline>;
defparam <mac_out_name>.signa_pipeline_clear = <aclr for sign_a
pipeline>;
```

```
defparam <mac_out_name>.signb_pipeline_clear = <aclr for sign_b
pipeline>;
defparam <mac_out_name>.data_out_programmable_invert = <
programmable invert for dataout port>;
defparam <mac_out_name>.overflow_programmable_invert = <
programmable invert for overflow port>;

// simulation-only parameters
defparam <mac_mult_name>.dataout_width = <width of dataout port>;
```

1.2.2 MAC Output Cell Input Ports

<*mac_out_name*>**:** is the unique identifier for the MAC output. This is any identifier name which is legal for the given description language (e.g. Verilog, VHDL, AHDL, etc.). This field is required.

.**dataa(**<*data_a source bus*>**), … .datad(**<*data_d source bus*>**):** are the four input buses for the output block. Depending on the mode, these inputs can take on different meanings. Note that this input can only be driven by the output of a stratix_mac_mult atom through dedicated internal routing – it is not visible external to the MAC. If the input is used in the specified operation mode, then it is required. Otherwise, it must be left unconnected.

.signa(<*signed/unsigned a source*>***),* .signb(**<*signed/unsigned b source*>***)*:** designates the two sign signals that drive the multiplier. NOTE: they do not correspond to the dataa and datab ports directly – instead they specify the sign of all inputs to the MAC_OUT. If either signal is high, then this implies that all inputs are signed integers. If both signals are low, then all the inputs are unsigned integers. This can be set to a constant value by tying the input to VCC or GND. This signal can be registered by specifying the signa/b_clock option. This signal can also be pipelined (i.e. double registered) by specifying the signa/b_pipeline_clock option. This signal is not required, and defaults to VCC if left unconnected.

.zeroacc(<*zero accumulator source*>**):** the signal which resets the accumulator to zero. Only available in accumulator mode. Note that this is different from clearing the accumulator registers (i.e. output registers) since only the feedback input into the accumulator gets cleared (so that no clock cycles are lost while clearing the accumulator). A value of VCC clears the accumulator feedback. This signal can be registered by specifying the zeroacc_clock option. This signal can also be pipelined (i.e. double registered) by specifying the zeroacc_pipeline_clock option. This signal is not required and defaults to GND if left unconnected.

.addnsub0(<*add or subtract 0 source*>**), .addnsub1(**<*add or subtract 1 source*>**):** specifies whether the first stage adders should be configured as subtractors or adders. Addnsub0 controls the top adder, and addnsub1 controls the bottom adder. High means that an addition should be performed, and low means that a subtraction should be performed. This can be set to a constant value by tying the input to VCC or GND. This signal can be registered by specifying the addnsub0/1_clock option. This signal can also be pipelined (i.e. double registered) by specifying the addnsub0/1_pipeline_clock option. This signal is not required and defaults to VCC if left unconnected.

.clk(<*clock source bus*>**):** designates the bus of four clock inputs that drive the MAC_OUT cell. Every register bank in the MAC_OUT cell can independently choose which one of these four clocks can drive the register. However, every clock can only be enabled by the corresponding clock enable signal of the same index – so there can only be a maximum of four clk/ena pairs. This signal must be connected if it is specified as used by a parameter.

.aclr(<*aclr source bus*>**):** designates the bus of four asynchronous clear inputs that drive the MAC_OUT cell. Every register bank in the MAC_OUT cell can independently choose which one of these four signals can drive the register. Unlike the clock enable signal, the aclr signal is not tied to any particular clock (e.g. can mix aclr[0] with clk[3]). This signal must be connected if it is specified as used by a parameter.

.ena(<*clock enable source bus*>**):** designates the bus of four clock enable inputs that drive the MAC_OUT cell. Each one of the four clk signals is enabled by the corresponding ena signal. This means that there can be a maximum of four pairs of clk/ce signals and not the sixteen combinations possible if they were completely independent. The reason for this is because the clock enable signal enables and disables the clock signal at the entry to the MAC block, and not at the register. This signal must be connected if it is specified as used by a parameter.

1.2.3 MAC Output Cell Output Ports

.dataout(<*data output bus*>**)**: the output bus of the MAC output cell (could be registered if output_clock option is specified).

.accoverflow(<*accumulator overflow signal*>**)**: indicates when the accumulator has overflowed (or underflowed if dealing with very negative numbers). This signal passes through the output register bank and therefore is always registered (since the accumulator must always be output registered) and is controlled by the same output register controls. Note that this signal does not latch – i.e. it only indicates whether the last accumulation operation overflowed, and gets cleared for the next cycle. This signal can be latched by adding external logic to implement a latch.

.observablezeroacc_regout(<*observable port for zeroacc*>**),**

.observableaddnsub0_regout(<*observable port for addnsub0*>**),**

.observableaddnsub1_regout(<*observable port for addnsub1*>**),**

.observablesigna_regout(<*observable port for signa*>**),**

.observablesignb_regout(<*observable port for signb*>**):** assigns a name for the observable port for the input signal. The observable port will only be created if its corresponding port is registered. These cannot be connected to other routing and are used for simulation and timing purposes only. By connecting wires to the observable ports, the user is able to rename the observable ports to the name of the connected wires. **This feature has not been implemented yet for these ports.**

.observablezeroacc_pipeline_regout(<*observable pipeline port for zeroacc*>**),**

.observableaddnsub0_pipeline_regout(<*observable port for pipeline addnsub0*>**),**

.observableaddnsub1_pipeline_regout(<*observable port for pipeline addnsub1*>**),**

.observablesigna_pipeline_regout(<*observable pipeline port for signa*>**),**

.observablesignb_pipeline_regout(<*observable pipeline port for signb*>**):** assigns a name for the observable pipeline port for the input signal. The observable port will only be created if its corresponding port is registered in the pipeline. These cannot be connected to other routing and are used for simulation and timing purposes only. By connecting wires to the observable pipeline ports, the user is able to rename the observable pipeline ports to the name of the connected wires. **This feature has not been implemented yet for these ports.**

1.2.4 MAC Output Cell Modes

<*clk for addnsub0*>, <*clk for addnsub1*>, <*clk for zeroacc*>, <*clk for sign_a*>, <*clk for sign_b*>, <*clk for output*> is one of *{none, 0, 1, 2, 3}*. Specifies which clock signal, if any, drives the given register, where the number indicates which clk signal (from clk[0] to clk[3]) should be used. If "none" is specified, then the corresponding signal is not registered. Otherwise, the corresponding signal will be registered and will be driven by the indicated clock, which must be connected. Additionally, this parameter also specifies which clock enable signal will be used. This forces the use of ena[2] if clk[2] is used for example. If the clock enable signal is not used, it must be tied to VCC.

<*aclr for addnsub0*>, <*aclr for addnsub1*>, <*aclr for zeroacc*>, <*aclr for sign_a*>, <*aclr for sign_b*>, <*aclr for output*> is one of *{none, 0, 1, 2, 3}*. Specifies which asynchronous clear signal, if any, drives the given register, where the number indicates which aclr signal (from aclr[0] to aclr[3]) should be used. If the option is not "none", then the register's clk parameter cannot be "none", and the corresponding signal's register will be cleared by the given aclr signal. However, if the option is "none", then the register's clk parameter must also be "none", and the corresponding signal is not registered. To disable the aclr, this signal should be tied to VCC.

<*clk for addnsub0 pipeline*>, <*clk for addnsub1 pipeline*>, <*clk for zeroacc pipeline*>, <*clk for sign_a pipeline*>, <*clk for sign_b pipeline*> is one of *{none, 0, 1, 2, 3}*. These behave identically to the clk parameters for the other registers. However, these signals control the pipeline register for the given signal (i.e. second stage register) instead.

<*aclr for addnsub0 pipeline*>, <*aclr for addnsub1 pipeline*>, <*aclr for zeroacc pipeline*>, <*aclr for sign_a pipeline*>, <*aclr for sign_b pipeline*> is one of *{none, 0, 1, 2, 3}*. These behave identically to the aclr parameters for the other registers. However, these signals control the pipeline register for the given signal (i.e. second stage register) instead.

<*programmable invert for dataout port*>, <*programmable invert for overflow port*> is a bit vector that specifies whether each output bit in the bus is inverted or not: a 1 indicates the output is inverted, while a 0 indicates it is not. For 'dataout', the bit vector is 72-bits long, while for 'overflow' it is 1 bit long. For example, a bit vector of "72'b000…011" means that dataout[0] and dataout[1] are inverted while dataout[2] and above are not inverted.

<*dataa width*>, <*datab width*>, <*datac width*>, <*datad width*> is an integer in the range {2, 3, …, 36}. It specifies the width of the multipliers that feed this output cell. Different widths are allowed, however internally this is represented by tying the

unused low-order bits to GND. This parameter also implies the output width as described in Table 1 below. In all modes except the 36_bit_multiply mode, all the widths must be the same. This is due to the way the smaller width numbers are represented (by tying off the lower unused bits). In the 36_bit_multiply mode, width_a + width_b must be equal to width_c + width_d. *This field is required.*

Table 1: Output Widths for MAC Output Cell

Operation Mode	Output Width
Output Only	width_a
Accumulator	width_a + 16
One Level Adder	width_a + 1
Two Level Adder	width_a + 2
36_Bit_Multiply	width_a + width_b

<*width of dataout port*> specifies the width of the dataout port. This is redundant information since the width is derived from the input port widths, but is needed for sim models that cannot do the calculation. This is only a simulation parameter and is ignored by the compiler.

<*operation mode*> is one of *{output_only, accumulator, one_level_adder, two_level_adder, 36_bit_multiply}*. It specifies how the output block behaves and which optional components within the block are used. In all modes, the output registers can be used or bypassed by setting the register_output option. *This field is required.*

- In "output_only" mode, only dataa can be used and is fed directly to the outputs. The function implemented in this mode is as follows:
  - out = a
- In "accumulator" mode, only dataa is used and feeds an internal representation of an accumulator. In this mode, the output registers are required since they are used to hold the accumulated data. Additionally, the zeroacc signal can be used to clear the accumulator feedback. Note that the addnsub0 signal can still be used to perform negative accumulation (i.e. subtract the multiplier result from the accumulated value instead of adding it) if desired. The accoverflow signal indicates whether the accumulator has overflowed (or underflowed) in this mode. This mode is only available if the multiplier is in 18-bit mode (i.e. the input width is 36-bits). The function implemented in this mode is as follows:
  - out = out +/- a
- In "one_level_adder" mode, only dataa and datab are used. The control signal addnsub0 controls whether dataa and datab will be added or subtracted (addnsub1 is not used in this mode). This control signal can be set to a constant by tying them to VCC or GND as desired. The function implemented in this mode is as follows:
  - out = a +/- b
- In "two_level_adder" mode, all four data inputs are used. The control signal addnsub0 controls whether dataa and datab will be added or subtracted (i.e. a -

b or a + b), while addnsub1 controls whether datac and datad will be added or subtracted (i.e. c - d or c + d). These control signals can be set to constants by tying them to VCC or GND as desired. The function implemented in this mode is as follows:
    - out = (a +/- b) + (c +/- d)
- In "36_bit_multiply" mode, all 4 data inputs must be used. In this mode, the four 18-bit multipliers are used to implement a 36-bit multiplier through internal adders and shifters. The multipliers have to be fed the appropriate bits from the original 36-bit signals as will be described later.

1.2.5 MAC Output Cell Polarities and Default Values

All control signals are active high. All non-data inputs have programmable inverts, however the data inputs do not have programmable inverts since they are driven through dedicated internal routing.

1.3 Modes of Operation

This section describes each of the five possible modes of operation for a MAC slice. In each mode, certain signals are allowed, and certain restrictions are required.
The following are conditions that apply in all modes of operation:

1. In all modes, the "Sign A" and "Sign B" signals on the multiplier can be used, but are not required. They default to VCC (i.e. signed integer) when left unconnected. Since there are only two sign signals for the entire MAC, there can only be two sign signals that drive all the cells in a MAC slice (e.g. signa must be the same signal for the MAC_MULT and the MAC_OUT cell it drives). Similarly, the register control parameters must be identical for each of the sign signals.
2. In all modes, all dynamic control signals (i.e. Sign A/B, AddnSub 0/1, Zero Acc) can be independently registered and/or pipelined (i.e. double registered) if the signal is connected. Additionally, the data inputs, MAC_MULT outputs, and MAC_OUT outputs can be independently registered.
3. In most modes, any register can be used or bypassed, though a register can only be used if its corresponding signal is connected. The only exception is the multiply-accumulator mode where the output registers of the MAC_OUT are required. If a register is used, then its specified CLK ports must be connected, and its specified ENA and ACLR ports must be connected or set to VCC/GND.
4. In all modes, the Data A, Data B and Data Out signals on the MAC_MULT are required and must be connected. Additionally, the Data A and Data Out signals on the MAC_OUT are required and must be connected.
5. In all modes, the Data A and Data B ports of the multiplier can be swapped since the result is functionally the same. However, this also means that any corresponding control signals must also be swapped. In cases where several multipliers feed adders, this means that they all must be checked together (e.g. since all multipliers share the Sign A and Sign B signals, these signals must be the same or constant in order to be able to swap the input ports).

6. If the Scanout signals of a multiplier are used and they drive another multiplier's Data inputs, then both multipliers must be of the same internal base width (i.e. 9-bits or 18-bits). Also, they both must be in the same operation mode.
7. Each Scanout signal can be used or bypassed independently (i.e. Scanout A can be used, while Scanout B is not used).
8. The Scanout signal of the bottom multiplier in a MAC can either drive the next MAC through dedicate internal routing, or it can drive regular routing (and possibly the next MAC through regular routing). However, if it drives regular routing, then some multipliers may become unusable, depending on the mode.

1.3.1 Multiplier Only Mode

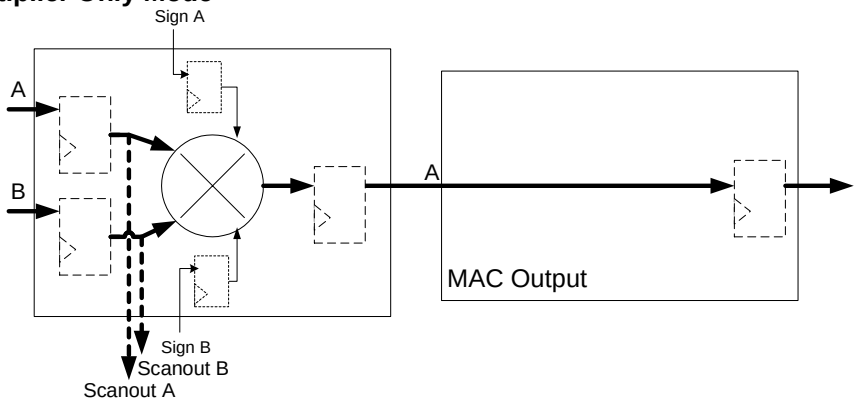

Figure 3: Multiplier Only Mode

In this mode, an independent multiplier is implemented. The following are conditions associated with this mode:

1. Up to four 18-bit multipliers or eight 9-bit multipliers can be implemented per MAC.
2. MAC_MULT conditions:
    a. The width of each input can be 1 to 18 bits.
    b. The width of the output is equal to width_a + width_b
    c. The Scanout signals can be used
3. MAC_OUT conditions:
    a. The width of the input must be the same as the output width of the multiplier feeding it (i.e. in the range 2 to 36 bits).
    b. The width of the output is equal to the width of the input (i.e. width_a)
    c. Out of all the input ports, only Data A must be connected – all other input ports cannot be used.
    d. The dynamic control signals AddnSub 0/1 and ZeroAcc are not used and cannot be connected.
4. Scan Chain conditions:
    a. The bottom multiplier(s) becomes unusable if the scanout signals are driven to regular routing since there are not enough outputs in the MAC for both the scanout signals and the bottom multiplier's results. In 18-bit mode, the bottom multiplier in the MAC becomes unusable and only 3 multipliers can be used in the MAC. In 9-bit mode, two multipliers can have their scanout signals drive regular routing (the ones in LAB rows 4

and 5). Therefore, depending on whether the multiplier above it is set to use regular routing, the two multipliers in rows 6 and 7 may become unusable. Only one of them may become unusable, or both of them may become unusable since choosing the scanout path is independent for both.

1.3.2 Multiply-Accumulator Mode

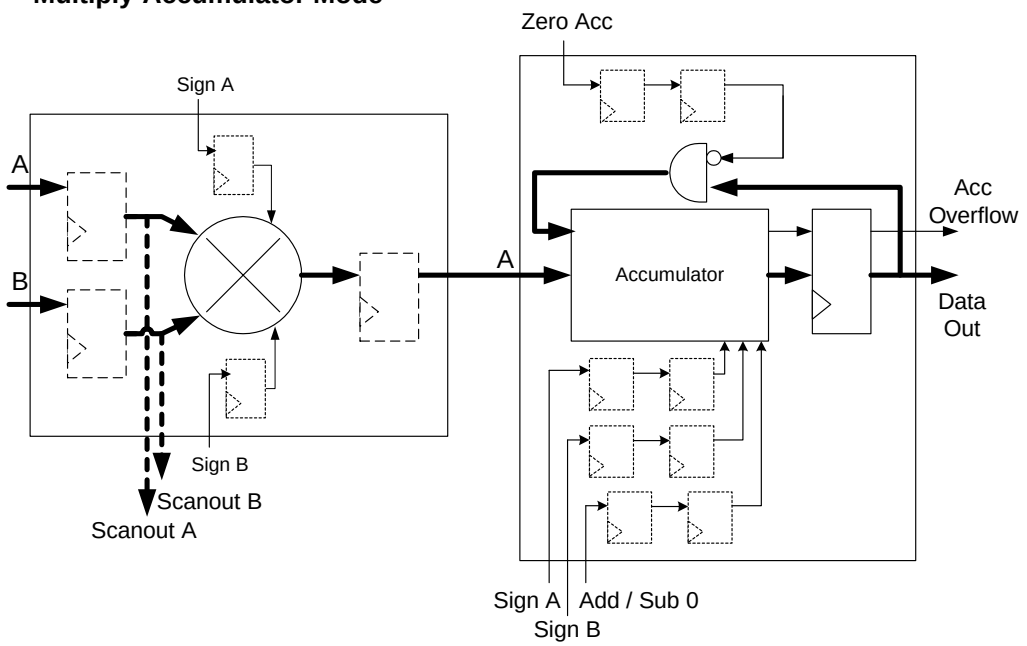

Figure 4: Multiply-Accumulator Mode

In this mode, a multiplier feeding an accumulator is implemented. The following are conditions associated with this mode:

1. Up to two 18-bit multiply-accumulators can be implemented per MAC. Accumulators with 9-bit based multipliers are not allowed, however smaller multipliers can still be implemented by tying to GND the unused low order bits of an 18-bit multiplier.
2. The top multiplier and the 3rd multiplier of the MAC are unusable when in accumulation mode, since only one multiplier can feed each of the two accumulators.
3. MAC_MULT conditions:
    a. The width of each input can be 1 to 18 bits.
    b. The width of the output is equal to width_a + width_b
    c. The Scanout signals can be used
4. MAC_OUT conditions:
    a. The width of the input must be the same as the output width of the multiplier feeding it (i.e. in the range 2 to 36 bits).
    b. The width of the output is equal to width_a + 16 (i.e. 16-bits extra to do the accumulation to a maximum of 52-bits)
    c. The output registers of the MAC_OUT are required in this mode.
    d. Out of all the input ports, only Data A must be connected – all other input ports cannot be used.

e. The dynamic control signal AddnSub 0 can be used if desired to perform negative accumulation (i.e. subtract multiplier result from total instead of adding), and should be tied to GND if only addition is desired. The dynamic control signal AddnSub 1 is not used and cannot be connected.
f. The dynamic control signal "Zero Acc" can be used, and defaults to GND if left unconnected. This signal, if asserted, clears the feedback input from the output registers feeding into the Accumulator. This signal can be registered and pipelined.
g. The output signal "Acc Overflow" indicates when the accumulator has overflowed, or underflowed (i.e. result is less than lowest negative number allowed). This signal is automatically cleared in the next clock cycle and is registered by the same output register bank used by the accumulator.

5. Scan Chain conditions:
   a. The bottom multiply-accumulator becomes unusable if the scanout signals are driven to regular routing instead of using the dedicated inter-MAC scanout routing. Therefore, only one multiply-accumulator can be implemented in a MAC which uses its scanout signal and drives regular routing.

1.3.3 Multiply-One-Level-Adder Mode

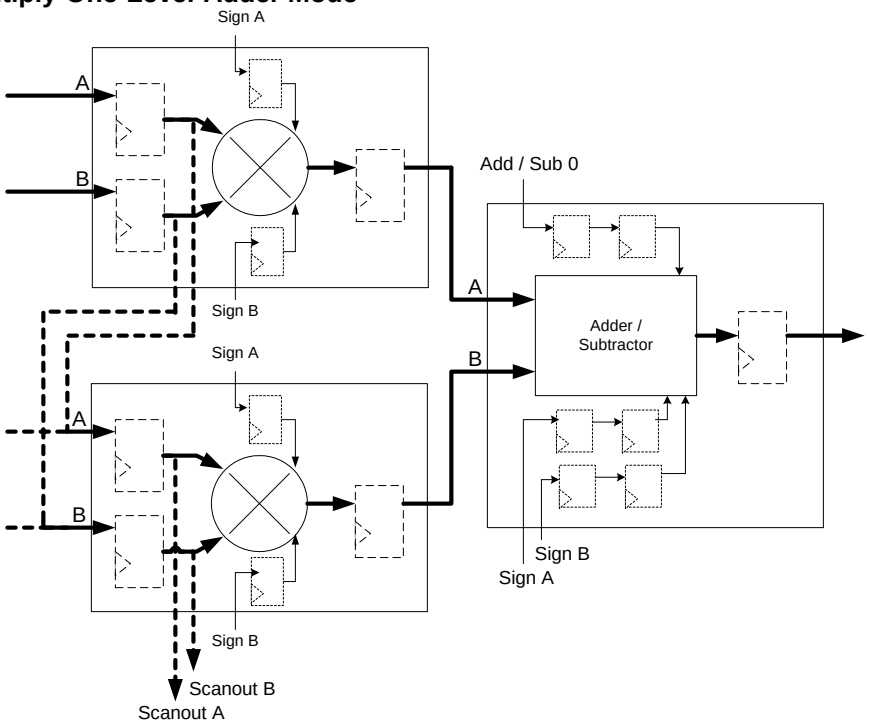

Figure 5: Multiply-One-Level-Adder Mode

In this mode, two multipliers feeding an adder/subtractor is implemented. The following are conditions associated with this mode:

1. Up to two 18-bit multiply-one-level-adders or four 9-bit multiply-one-level-adders can be implemented per MAC.
2. MAC_MULT conditions:

   a. The width of each input can be 1 to 18 bits.
   b. The width of the output is equal to width_a + width_b. The output width of both multipliers must be the same (though all four input widths can be different).
   c. Either multiplier can use their Scanout signals. If the top multiplier uses its Scanout signal then it must feed the bottom multiplier's data inputs – it cannot drive anything else. If the bottom multiplier's Scanout signals are used, then it can drive another multiplier of the same base width as described earlier, or it can drive regular routing (and thus any other cell). However, if two multiply-one-level-adders are in the same MAC then the top one can only drive the bottom one, since only the bottom multiplier in a MAC can drive regular routing.
3. MAC_OUT conditions:
   a. The width of each input must be the same as the output width of the multiplier feeding it (i.e. in the range 2 to 36 bits). Additionally, both inputs must have the same width.
   b. The width of the output is equal to width_a + 1
   c. Out of all the input ports, Data A and Data B must be connected – all other input ports cannot be used.
   d. The dynamic control signals AddnSub 1 and ZeroAcc are not used and cannot be connected.
   e. The dynamic control signal AddnSub 0 can be used and defaults to GND if left unconnected. This signal controls whether addition or subtraction is performed.
   f. The inputs to the adder can be swapped only if the AddnSub 0 signal is tied to GND (since otherwise a subtraction of the swapped parameters would yield different results). However, care must be taken to verify that the multipliers driving the adder are legal and are adjusted accordingly.
4. Scan Chain conditions:
   a. In this mode no multipliers are lost if the scanout signal drives regular routing. Unlike the previous modes, in this modes there are enough MAC outputs to accommodate both the scanout signals and the adder outputs.

1.3.4 Multiply-Two-Level-Adder Mode

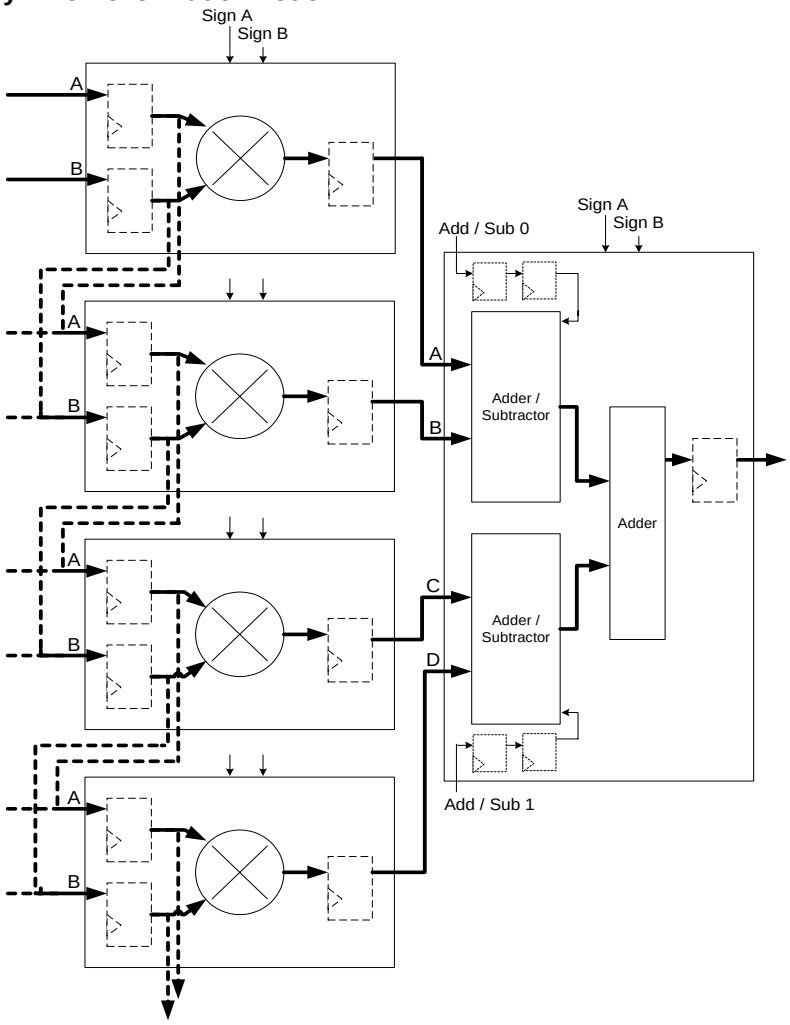

Figure 6: Multiply-Two-Level-Adder Mode

In this mode, four multipliers feeding a two-level adder is implemented. The following are conditions associated with this mode:

1. Up to one 18-bit or two 9-bit multiply-two-level-adders can be implemented per MAC.
2. MAC_MULT conditions:
    a. The width of each input can be 1 to 18 bits.
    b. The width of the output is equal to width_a + width_b. The output width of all multipliers must be the same (though their input widths can be different).
    c. Either multiplier can use their Scanout signals. As in the One-Level-Adder case, if any multiplier other than the last multiplier uses its Scanout signal, it can only drive its next adjacent multiplier. Only the last multiplier can drive regular routing or another multiplier of the same base width.
3. MAC_OUT conditions:

a. The width of each input must be the same as the output width of the multiplier feeding it (i.e. in the range 2 to 36 bits). Additionally, all inputs must have the same width.
b. The width of the output is equal to width_a + 2
c. All Data inputs must be connected. However, any multiplier can have all of its inputs tied to GND and thus will not affect the addition of the other multipliers – this can be used to implement 3 multipliers feeding an adder (the fourth multiplier will have no effect on the output).
d. The dynamic control signal ZeroAcc is not used and cannot be connected.
e. The dynamic control signals AddnSub 0 and AddnSub 1 can be used and default to GND if left unconnected. These signals control whether addition or subtraction is performed by each of the first level adder units – AddnSub 0 controls the unit that is fed by Data A and Data B, while AddnSub 1 controls the unit that is fed by Data C and Data D. The second stage adder can only perform addition.
f. The Data A and Data B inputs can be swapped only if AddnSub 0 is tied to GND. Similarly, Data C and Data D can be swapped only if AddnSub 1 is tied to GND. Additionally, the pair of inputs Data A and Data B can be swapped with the pair of inputs Data C and Data D (i.e. A with C and B with D). However, care must be taken to verify that the multipliers driving the adders are legal and are adjusted accordingly.

5. Scan Chain conditions:
   a. In this mode no multipliers are lost if the scanout signal drives regular routing. Unlike the previous modes, in this modes there are enough MAC outputs to accommodate both the scanout signals and the adder outputs.

1.3.5 36-bit Multiplier Mode

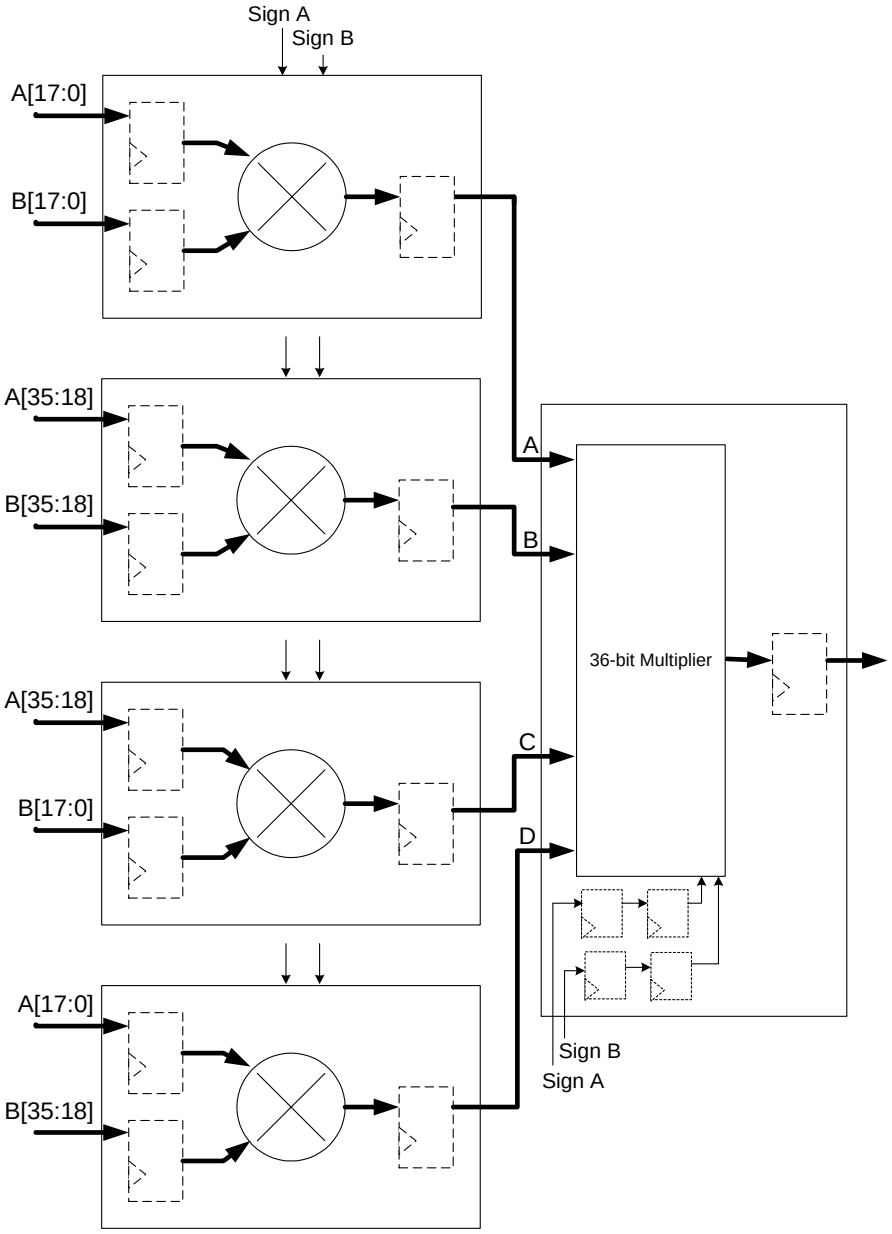

Figure 7: 36-bit Multiplier Mode

In this mode, an independent 36-bit multiplier is implemented. This is accomplished by using four 18-bit multipliers and special adder/shifter circuitry to perform a 36-bit multiplication. The four 18-bit multipliers are fed part of each input as outlined in the diagram. The following are conditions associated with this mode:

1. Only one 36-bit Multiplier using four 18-bit multipliers can be implemented per MAC.
2. MAC_MULT conditions:
    a. The width of each input can be 0 to 18 bits. However, this is completely dependant on the width of the original 36-bit inputs. As before, to implement smaller width multipliers, the unused low-order bits are tied to

GND. For example, if operand A is actually 27 bits wide, then all bits in A[35:18] and A[17:9] must be connected, while all bits in A[8:0] must be tied to GND. Additionally, if operand B is actually 15 bits wide, then all bits in B[35:21] must be connected, and all bits in B[20:18] and B[17:0] must be tied to GND (this means that the multipliers fed by B[17:0] will have a width of 0). This implies the MAC_OUT width would be 42-bits wide with OUT[71:30] representing the output result, and OUT[29:0] representing the unused low-order bits and would be left unconnected.

   b. The width of each output is equal to width_a + width_b
   c. The Scanout signals cannot be used

3. MAC_OUT conditions:
   a. The width of the input must be the same as the output width of the multiplier feeding it (i.e. in the range 1 to 36 bits).
   b. The width of the output is equal to width_a + width_b
   c. All Data input ports must be connected as outlined in the diagram.
   d. The dynamic control signals AddnSub 0/1 and ZeroAcc are not used and cannot be connected.
   e. The Data C and Data D inputs can be swapped with each other. However, care must be taken to verify that the multipliers driving the output block are legal and are adjusted accordingly.

6. Scan Chain conditions:
   a. In this mode the scanout signals cannot be used. If scan chains are desired, they should be implemented by the user external to the MAC block using regular logic.