EFFICIENT HIGH PERFORMANCE DATA OPERATION ELEMENT FOR USE IN A RECONFIGURABLE LOGIC ENVIRONMENT

Related Application/Priority

[0001] The present application claims the priority of Provisional Application No. 60/288,298, filed May 2, 2001.

Background of the Invention

[0002] The present invention relates to reconfigurable logic chips, especially reconfigurable logic chips used for reconfigurable computing.

[0003] Field programmable gate arrays (FPGAs) are programmable chips that can implement different configurations. Typically a design is created using design tools and a FPGA is configured for a specific design. Although designs can be changed, typically the FPGA uses a single configuration due to the relatively long time required to change a configuration compared to the operation time of the chip.

[0004] Recently reconfigurable chips designed to quickly switch portions of an algorithm on to a reconfigurable chip have been created. These reconfigurable chips are designed to use the reconfigurable elements of the chip so as to provide resources for the implementation portions of an algorithm.

[0005] It is desired to have a data operation element or reconfigurable functional unit for use in a reconfigurable chip that implements an improved design to more effectively implement algorithms on a reconfigurable chip.

Summary of the Invention

[0006] The present invention concerns a reconfigurable chip, including multiple reconfigurable functional units (such as a data path unit) adapted to implement different functions. The reconfigurable function units preferably include multiplexers, at least one shifting unit and at least one arithmetic logic unit (ALU). The reconfigurable functional units are configured by reconfigurable functional unit instructions. The instructions control the configuration of the multiplexer and shifting unit and the ALU. The reconfigurable chip also includes interconnect adapted to connect together the reconfigurable functional units. In this way data can be passed between the reconfigurable functional units.

[0007] The reconfigurable functional unit instruction preferably includes a number of fields for the multiplexers, the shifter unit and the arithmetic logic unit. These fields configure these elements in the reconfigurable functional unit in a desired way.

[0008] In a preferred embodiment, there is an associated instruction memory for each reconfigurable functional unit has an associated instruction memory. The instruction memory stores multiple instructions for the reconfigurable functional unit. In a preferred embodiment, the state machine addresses the instruction memory to determine the next instruction to be loaded into the reconfigurable functional unit. In one preferred embodiment, the reconfigurable functional units provide feedback to the state machine indicating when a function is finished and thus when the next function can be loaded into the reconfigurable functional unit.

[0009] In one embodiment, the shifter unit is configurable with a number of different modes. These modes are preferably selectable by a field of the reconfigurable functional unit instruction.

[0010] In one embodiment, interconnect elements are adapted to selectively connect some of the reconfigurable functional units to transfer word length data. The transferred data preferably has a fixed data length of 32 bits or greater. The

fixed length data transfer allows interconnect system to be simplified at the loss of flexibility in the data transfer. The shifting unit in the reconfigurable functional unit allows the arithmetic logic unit to operate on different bits in the word length input data of the reconfigurable functional unit compensating for the fixed structure of the interconnect elements. Thus, if needed data is in a certain location within a word, the shifter can move that bit location to the proper position for manipulation by the arithmetic logic unit.

[0011] Another embodiment of the present invention comprises using a multiplexer with a delay unit input and an input that bypasses the delay unit. In this manner the reconfigurable functional unit can implement a variable delay increasing the flexibility of the system.

Brief Description of the Drawing Figures

[0012] Fig. 1 is an overview of the reconfigurable chip of one embodiment of the present invention;

[0013] Fig. 2 is a simplified diagram of a reconfigurable functional unit of one embodiment of the present invention;

[0014] Fig. 3 is a diagram of a reconfigurable functional unit of one embodiment of the present invention;

[0015] Fig. 4 is a diagram of a multiplier unit which can be used with the embodiment of the present invention;

[0016] Fig. 5 is a diagram of one slice of the reconfigurable functional unit shown in Fig. 1 illustrating the interconnection between the data path units;

[0017] Fig. 6 is a diagram illustrating the connections between the data path unit and the horizontal and vertical bus lines;

[0018] Fig. 7 is a diagram illustrating the interconnection of a data path unit in one tile to a data path unit in another tile;

[0019] Fig. 8 is a diagram illustrating the interconnection of the data path units and a local system memory of one embodiment of the present invention;

[0020] Fig. 9 is a diagram illustrating a state machine and functional block configuration memory producing the instruction of configuration information for the functional block data unit;

[0021] Fig. 10A is a diagram illustrating the interconnection of a state machine, configuration state memory and data path unit of the present invention, showing the instruction and instruction fields for the data path unit;

[0022] Fig. 10B is a diagram illustrating a data path unit using a decoder for at least part of the instruction;

[0023] Fig. 11 is a diagram illustrating the control system configuration memory at data path unit as one embodiment of the present invention;

[0024] Fig. 12 is a diagram of an interconnection logic unit for use in one embodiment of the present invention;

[0025] Figs. 13A and 13B are charts illustrating the portions of the instructions for the ALU;

[0026] Fig. 14 is a diagram illustrating the flags for the system of one embodiment of the present invention;

[0027] Fig. 15 is a diagram illustrating the shifting mode for shifter;

[0028] Fig. 16 is a diagram of the instruction of one embodiment of the shifter;

[0029] Fig. 17 is a diagram illustrating the operation of the shifter of Fig. 16;

[0030] Fig. 18 is a diagram of a logic system using a multiple master latches of one embodiment of the present invention;

[0031] Fig. 19 is a diagram illustrating the background and foreground plane latches of one embodiment of the present invention;

[0032] Fig. 20 is a diagram of one embodiment of a reconfigurable functional unit for a data path in one embodiment of the present invention;

[0033] Fig. 21 is a diagram of the input multiplexers for the system of Fig. 20;

[0034] Fig. 22 is a diagram of the shifting mode for the shifter of one embodiment of the present invention;

[0035] Fig. 23 is a diagram illustrating some shifting modes for the shifter of one embodiment of the present invention; and

[0036] Fig. 24 is a diagram illustrating the implementation of a turbo look up table of one embodiment of the present invention.

Detailed Description of the Invention

[0037] Fig. 1 illustrates a reconfigurable chip 20. The reconfigurable chip 20 includes a central processing unit (CPU) 22, preferably a reduced instruction set (RISC) CPU. Data from the external memory (not shown) is transferred using memory controller 24. Bus 26, called the roadrunner bus, is used to transfer data from the memory controller to the reconfigurable fabric 28. The reconfigurable fabric 28 is divided into a number of slices. Each slice is broken down into a number of tiles. Each tile includes a data path unit (reconfigurable functional unit), control units and local system memory units. The local system memory units interact with the data path units as described below. In a preferred embodiment, each tile also has a number of multiplier units.

[0038] Fig. 2 illustrates a simplified diagram of a reconfigurable functional unit of one embodiment of the present invention. The reconfigurable functional unit includes input multipliers 30 and 32. As will be described below, the input multipliers allow the data path unit to receive inputs from a number of different locations, including nearby data path units as well as data buses. The selected output of the input multipliers are sent on to registers 36 and 38. Additionally, the output of the multiplier 32 goes to shifter unit 34. As described below, the shifter unit 34 allows for the selection of different bits to be operated on by the ALU 40. Since the interconnections between the data path units use fixed word

length connections to simplify the interconnection system, the use of a shifter unit in the data path unit allows access to bits packed within the interior of a word.

[0039] As will be described below, the shifter unit 34 preferably has a number of modes which implement more than just logical and arithmetic shifts left and right. These different modes allow the system to operate in a more efficient manner. The arithmetic logic unit 40 described below, preferably uses a field of the instruction for the data path unit to implement a function. The output of the ALU 40 preferably goes to an output register 42. The output can also actually be sent to an optional bit shifter 44 to produce a shifted value.

[0040] In one embodiment, a bypassing ALU feedback output on line 46 is also used. This allows portions of the data path unit to operate while the output register 42 controls what outputs are sent from the data path unit. This is useful when the output register 42 is used to address a local system memory unit.

[0041] The bit shifter 44 is used to implement the linear feedback shift register as described in the patent application, "Modifications to Reconfigurable Functional Unit in a Reconfigurable Chip to Perform Linear Feedback Shift Register Function," by Peter Lam, Attorney Docket No. 032001-060.

[0042] Note that the multiplexers, shifter unit 34 and ALU 40 are preferably controlled by an instruction for the data path unit. This instruction is preferably divided into a number of different fields, including multiplexer instruction fields for the multiplexers, shifter unit fields for the shifter 34 and ALU instruction field for the ALU 40. In one embodiment, a decoder is used for at least part of the instruction.

[0043] Fig. 3 is a detailed diagram of one embodiment of the present invention. The input multiplexers 50 and 52 receive as inputs data from nearby units. In one example, data words from 16 units, including data path units and multiplier units, are used as inputs. Global vertical and horizontal interconnections are used. In one embodiment a connection for the linear feedback shift register feedback, a

logical zero constant input and an input for a local system memory unit. Another input is the carry input from the prior data path unit which is provided to the ALU 54 directly. The multiplexer 50 is connected to the shifter 56, including a number of different modes of operation. The shifter 56 is connected to another multiplexer 58 so that the output of the multiplexer 50 can either avoid or use shifter unit 56. The shifter unit 56 can also use the A input from the input multiplexer 52 for some of the modes. The output of the multiplexer 58 and the output of multiplexer 52 can be sent to registers 60 and 62, respectively. The registers 60 and 62 can also be loaded from off the chip. This logic 64 and 66 allows for the register values to act as a mask register for the system. The multiplexers 68 and 70 select the inputs to the ALU 54. Outputs to the ALU are sent to a number of different possible paths. Note that the data path output out of multiplexer 72 can be the value from the output register 74, or the value from the multiplexer 76 (which can be the ALU value or the local system memory re-data on line 78). The flag values from the ALU are sent to multiplexers 80 and 82 which select the desired flag value. This flag value can be stored in registers 88 and 90 and the value of the registers 88 or 90 is sent to the multiplexers 92 and 94 or the selected value from multiplier 80 or 82 is used. The CONF value is a field in the instruction that indicates which flag to select.

[0044] In one embodiment, the registers 60, 62 and 74 can be implemented by using multiple master slave latches, shown in Fig. 18, to allow the loading of background configuration data into the register. In one embodiment, the operation of these registers can be controlled by the field of the reconfigurable functional unit instruction.

[0045] Fig. 4 is a diagram of a multiplier unit. The multiplier unit is oriented somewhat similar to the reconfigurable functional unit shown in Fig. 3. However, the multiplier unit has a dedicated multiplier rather than an ALU.

[0046] As shown in Fig. 5, in one embodiment for every seven data path units or reconfigurable functional units in a tile, there are two multiplier units.

[0047] Fig. 6 illustrates the connections of adjacent data path units and multipliers into the data path unit inputs. Note looking at Fig. 5, the data path unit 100 can receive as an input, outputs from eight previous data path units (and multipliers) above and seven of the next data path units (and multipliers) below. The output of the data path unit 100 is also fed back to itself. The outputs of any of these units can be selected using the input multiplexers of the system for either the A or B inputs.

[0048] Fig. 6 is a diagram that illustrates the connection of the one tile reconfigurable functional units (data path units) to horizontal and vertical connection lines. By using multiplexers, the outputs, and inputs, of the data path units can be interconnected to both the vertical routing lines and the horizontal routing lines.

[0049] Fig. 7 illustrates an example of interconnecting a data path unit in one tile to a data path unit in another tile using vertical interconnected lines. Note that the system of the present invention for the interconnections preferably uses word-based interconnections. In one embodiment, the interconnection lines allow the connection of 32 bit wide data. The shifter unit in the data path unit allows for the alignment of the data once it is received into a data path unit from the interconnection system. Because the system sends data in a 32 bit word, the complexity of the interconnect system is reduced and simplified, reducing somewhat the flexibility of the interconnect.

[0050] Fig. 8 illustrates the connection between the data path units and the local system memory. In a preferred environment, alternate data path units are used to implement the Writes and Reads of the local system memory. For example, data path unit 102 provides Read Addresses to and receives read data from the local system memory 104. Data path unit 106 provides the Write Address and Write

data for the local system memory 104. Note that by using the pass gates such as pass gates 106, 108, 110 and 112, the data path units 102 and 106 can connect to other local system memories, such as local system memory 114 and the data path units 116 and 118 can connect to the local system memory 104. In another embodiment, a data path unit can both read from and write to a local system memory. One of the uses of a data path unit is to provide an address to a local system memory to obtain data from a local system memory, which can then be put upon the horizontal and vertical interconnection buses. The connections shown in Fig. 8 are the direct connections to read and write data in and out of the local system memory. In a preferred environment, the local system memory is globally read from and written to using the memory control system. This general memory control system is used for configuration of the system and for obtaining the data operated on by the data path units. Note that as described above, in a preferred embodiment, the data path units include structures that allow addresses and data to be provided to the local system memory while the data path unit does some other function.

[0051] Fig. 9 is a disclosure of a control fabric unit 132 for the reconfigurable functional unit 130. In this embodiment, the control fabric unit 132 produces a control or instruction line for the reconfigurable functional unit 130. In this embodiment, the control fabric unit 132 is preferably composed of a state machine unit 134 and a functional block configuration memory unit 136. The state machine 134 produces the addresses into the instruction memory 136. One implementation of the state machine 134 uses a reconfigurable programmable sum-of-products unit 136.

[0052] Fig. 10A illustrates a system with the state machine configuration unit 136, the configuration state memory 138' and the data path unit 130'. Note that the configuration from the configuration state memory 138' can be considered to be an instruction for the data path unit 130'. The instruction preferably includes

fields such as an ALU configuration field, shift register configuration field, and a multiplexer configuration field. In one embodiment, some of the flags from the data path unit 130' are sent to the state machine 136' in order to switch configurations for the data path unit after the data path unit is done operating on a set of data. The configuration state machine 138' can also be loaded from an external configuration from external memory or from the processor.

[0053] Fig. 10B is a diagram illustrating a data path unit using a decoder to decode at least part of an instruction.

[0054] Fig. 11 shows the control system, including the state machines for the different configuration state memories. The data path unit flags are sent to the control system as described above.

[0055] Fig. 12 is a diagram that illustrates one example of an arithmetic logic unit. This arithmetic logic unit includes an arithmetic unit 142, a parallel logic unit 140 and a flag unit 144. Also shown is a carry selection unit 146. The ALU instruction field from the instruction is sent to select the operations of the ALU. The arithmetic unit 142 uses a carry input. In a preferred embodiment, this carry value is either the carry from the previous data path unit or the control signal or a carry which is part of the instruction.

[0056] Figs. 13A and 13B illustrate a list of some of the Opcodes used in one embodiment of an ALU of the reconfigurable functional unit of the present invention. Details of these Opcodes are described in the Appendix I, incorporated herein by reference.

[0057] Fig. 14 is a diagram of the flag system for the present invention. The flag unit is inside the data path unit and used for producing the flags which go to the control unit as well as to the next data path unit. The selection of the flags which are used the control of a field of the reconfigurable functional instruction and are preferred by the present invention. A description of the some of the flags is given below.

[0058] ROXR is driven every cycle. It is selected by conf == 1.

The operation is:

case opcode[7] ==0	flag[1] == ^(B[31:0])
	flag[0] == ^(B[15:0])
case opcode[7] == 1	flag[1] == ^(B[31:16])
	flag[0] == ^(B[15:0])

Abbreviations:

CO - Carry Out (of the addition/subtraction operation)

OV - Overflow (of the addition/subtraction operation)

EQ - Equal (A== B)

GT - Greater Than

LT - Less than

SN - Sign (sign bit of result)

Previous Flag

Cin - Carry in pevious row

Ctrl - Carry in from control

Max - Ox7fff[ffff] (for 16/32 bits)

Min - 0x8000[0000] (for 16/32 bits)

[0059] Fig. 15 illustrates the shift mode and the operation of some of the modes of the shifter unit of one embodiment of the present invention. Since the shifter unit has a number of different modes, the flexibility of the system of the present invention is increased.

[0060] Figs. 16 and 17 illustrate one implementation of the shifter unit using multiple rows of multiplexers. Additional logic is also of use to produce a special output. Fig. 17 illustrates the operation of some of the implementations of the shift register.

[0061] This shifter used in the Datapath unit performs more than right/left shift operations. The shifter includes an array of multiplexers which are controlled via

mux select signals. In one 4x6 multiplexer array shifter embodiment, a 32-bit operand which is divided into four groups of 8 signals is coupled to a first row of four multiplexers. Other than the last row, the outputs of the multiplexers in a previous row are coupled to the inputs of the next row of multiplexers. Each multiplexer in the array is controlled independently. The control signals determine how the signals are routed in the array and hence the type of operation performed on the operand. In one embodiment, examples of operations include: 32-bit logical right/left shift, 32 bit arithmetic right/left shift, lower 16-bit sign extend to 32-bit, constant generation, duplicate lower 16 bit to upper 16 bit, duplicate upper 16 bit to lower 16-bit, swap lower and upper 16-bit, 16-bit arithmetic right shift, and byte swap.

[0062] Fig. 18 illustrates a multiple master latch system used in one embodiment of the system of the present invention. In this example, two master latches are used. One of the master latches is used for the background configuration of the system. The other master latch receives data from the pipeline in the data path unit or from the processor. The inputs to the latch 150 are provided through the multiplexer 152. The latch 154 is connected to the configuration bus to receive data from background configuration. The multiplexer 156 can be used to select the input to the slave latch 158. The use of a background configuration memory to the system allows the quick operation of the system in the present invention.

[0063] The storage element of Fig. 18 has multiple master latches which share a single slave latch via a multiplexer which provides a multi-function storage element. In addition, by sharing a slave latch a significant space savings is realized (approximately 25%). This is particularly true in a system utilizing numerous storage elements. The storage element design relies on the fact that configuration bits are infrequently loaded into storage elements. So instead of having a separate slave latch for each master latch coupled to a configuration bitstream signal,

according to the invention the master latch coupled to the configuration bitstream signal shares its slave latch with another master latch. Hence, two or more master latches share a single slave latch. A multiplexer is coupled between the master latches and the single slave latch for selecting which master latch is coupled to the slave latch.

[0064] In one embodiment, one master latch's input is coupled to a signal that frequently requires the storage element functionality and the other master latch's input is coupled to a signal that requires the storage element functionality on an infrequent basis. In the first master latch is coupled to the data path signal and the second master latch is coupled to the configuration bit signal. When the data path signal is passed to the slave latch, the storage element functions to divide the data path pipeline into stages. When the configuration bitstream signal is passed to the slave latch, the storage element functions to store the configuration bits. In another embodiment, one master latch is coupled to the data path signal and more than one master latch is coupled to a configuration bit signal and all of the master latch outputs are coupled to the multiplexer which is used to select and pass one of the signals from the master latches to the shared slave latch.

[0065] For Fig. 18:

- master latches are reset upon 'RESET' or 'INIT'
- slave latches are reset upon 'RESET' only
- mux A selects config path whenever configuration is activating, (further qualified by the particular slice being selected)
- mux B selects arc bus when arc is writing (further qualified by decoding corresponding arc_address. Please refer to ARC ext spec for address map)
- master latches are transparent only during clock low
- slave latches are transparent only during clock high

- master latch 0 is transparent when the latpipe 0 is enabled or arc write to that register is happening
- master latch 1 is transparent when config loading is activate and the corresponded config address is decoded
- slave latch is transparent when 1. config activate to this slice or
  2. arc write to this register or
  3. latpipe signal from control is high
- This setup is under the assumption that configuration and arc write will not happen at the same time. If it does happen, the configuration has higher priority.

[0066] Another embodiment of the present invention concerns the variable delay units of the present invention. Variable delay units are comprised of a multiplexer which receives first unit sent into a register and the second input which bypasses the register. In this way a variable delay can be implemented. In the reconfigurable functional unit of Fig. 3 the register 60, connected to the multiplexer 68 and the register 62 connected from multiplexer 70 and register 88 connected to the multiplexer 92, the register 90 connected to the multiplexer 94 and the register 74 connected to the multiplexer 72, can implement such a variable delay. A multiplexer can select a delayed or a bypass signal; the delay signal going through a delay element like a flip-flop.

[0067] The flexible adaptive delay element includes a storage device (e.g., flip-flop, latch) having its input coupled to an input signal and its output coupled to a first input of a multiplexer. The other input of the multiplexer is coupled to the input signal. As a result, the first input of the multiplexer is coupled to the input signal and a second input of the multiplexer is couple to the input signal delayed by the amount provided from the storage device. The select signal can then be used to select either the delayed signal or the undelayed signal.

[0068] Fig. 19 shows an alternate embodiment of a background foreground plane arrangement

[0069] The present invention incorporates by reference the prior patent applications, including "A HIGH PERFORMANCE DATA PATH UNIT FOR BEHAVIORAL DATA TRANSMISSION AND RECEPTION," Inventor Hsinshih Wang, Serial No. 09/307,072 filed May 7, 1999 (Attorney Docket No. 032001-014), "CONTROL FABRIC FOR ENABLING DATA PATH FLOW," Inventors Shaila Hanrahan, et al., Serial No. 09/401,194 filed September 23, 1999 (Attorney Docket No. 032001-016), as well as "CONFIGURATION STATE MEMORY FOR FUNCTIONAL BLOCKS ON A RECONFIGURABLE CHIP," Inventors Shaila Hanrahan and Christopher E. Phillips, Serial No. 09/401,312, and filed on September 23, 1999 (Attorney Docket No. 032001-035).

[0070] Vermont Embodiments.

[0071] Fig. 20 illustrates a ultimate embodiment of the reconfigurable functional unit or data path unit. In this embodiment an additional register and multiplexer are added to the B input path before the shifter. Additionally, the input multiplexer is slightly modified. The input multiplexer is shown with respect to Fig. 21.

[0072] Fig. 22 illustrates the shifter mode table for the new embodiment of Fig. 19.

[0073] Fig. 23 illustrates the implementation of the new modes of Fig. 22.

[0074] Fig. 24 illustrates a turbo look up table for use with the system of the present invention. The turbo look up table is useful in the addition of data stored in a logarithmic format. This is useful for many communication systems. In one prior embodiment in order to do the multiplication of data stored in logarithmic format, the data must be converted to the normal format by doing an exponential expansion of the data. The exponentially expanded data is added together and then the combined information is converted back to the logarithmic format. In the

preferred embodiment, the turbo look up table is used in the production of an estimate of the addition of a correction factor. This estimate uses the value of the greatest value of A and B as a first estimate of the value of addition of A plus B. The absolute value of this difference of A minus B is used as an input to a look up table to provide a correction factor to add to the greatest value of A or B. By adding this correction factor to the greatest value of A or B a relatively accurate estimation is produced. Note that the look up table need not have the same number of input bits of A. In a preferred embodiment, only a few bits of precision are used. If the magnitude of A minus B is relatively large, the combined value does not significantly differ from the greatest value of A or B. For example, the addition of 1,000,000 to 0.1 is approximately 1,000,000. The addition of 1,000,000 to 1,000,000 is effectively doubling the maximum value.

[0075] Appendixes II and III further illustrate the Vermont embodiment of the reconfigurable functional unit.

[0076] It will be appreciated by those of ordinary skill in the art that the invention can be implemented in other specific forms without departing from the spirit or character thereof. The presently disclosed embodiments are therefore considered in all respects to be illustrative and not restrictive. The scope of the invention is illustrated by the appended claims rather than the foregoing description, and all changes that come within the meaning and range of equivalents thereof are intended to be embraced herein.

1.9 Opcode Details

Name	ADD
Pseudocode	result = A+B
Description	A plus B
Flags Affected	CO carry out of the operation OV if the operation overflow EQ if a==b SN result[31]

Name	ADD16
Pseudocode	result = {(AH+BH),(AL+BL)}
Description	Parallel A plus B
Flags Affected	CO,OV,EQ,SN similar to ADD except the flag[0] is also valid

Name	ADDC
Pseudocode	result = A+B+Cin
Description	A plus B with carry in for 32 bit operation. Cin is the CO of the lower 16bits.
Flags Affected	CO OV EQ SN similar to ADD

Name	ADDCNT
Pseudocode	result = A+B+Ctrl
Description	A plus B with control carry in
Flags Affected	CO,OV,EQ,SN similar to ADD

Name	SUB
Pseudocode	result = A-B
Description	A minus B
Flags Affected	COcarry out of the operation (A+~B+1) OVif the operation overflow GTif A>B

LTif A<B
EQif A==B
SNsign of result

Name	SUB16
Pseudocode	result = {(AH-BH),(AL-BL)}
Description	Parallel A minus B
Flags Affected	CO,OV,GT,LT,EQ,SN
	similar to SUB

Name	SUBC
Pseudocode	result = A+~B+Cin
Description	A minus B with carry in for 32 bit operation
Flags Affected	CO,OV,GT,LT,EQ,SN.
	similar to SUB

Name	SUBCNT
Pseudocode	result = A+~B+Ctrl
Description	A minus B with control carry in
Flags Affected	CO,OV,GT,LT,EQ,SN
	similar to SUB

Name	SADD

Pseudocode	if (overflow)
	result = max
	else if (underflow)
	result = min
	else
	result = A+B

Description	A plus B with saturation
Flags Affected	COcarry out of A+B OVoverflow of A+B EQA==B SN sign of result

Name	SADD16
Pseudocode	Parallel 16 bit version of SADD
Description	A plus B with saturation
Flags Affected	CO,OV,EQ,SN
	similar to SADD, flag[0] is valid

Name	SADDCNT

Pseudocode	if (overflow)
	result = max
	else if (underflow)
	result = min
	else
	result = A+B+Ctrl

Description	A plus B with control carry in and saturation
Flags Affected	CO,OV,EQ,SN
	similar to SADD

Name	SSUB

Pseudocode	if (overflow)
	result = max
	else if (underflow)
	result = min
	else
	result = A-B

Description	A minus B with saturation
Flags Affected	CO carry out of A+~B+1
	OV overflow of A-B
	GT A>B
	LT A<B
	EQ A==B
	SN sign of result

Name	SSUB16

Pseudocode	Parallel 16 bit version of SSUB
Description	A minus B with saturation
Flags Affected	CO,OV,GT,LT,EQ,SN
	similar to ssub, flag[0] valid

Name	SSUBCNT

Pseudocode	if (overflow)
	result = max
	else if (undeflow)
	result = min
	else
	result = A+~B+Ctrl

Description	A minus B with control carry in and saturation
Flags Affected	CO,OV,GT,LT,EQ,SN
	similar to ssub

Name	INC
Pseudocode	result = B+1
Description	Increment B
Flags Affected	CO carry out of B+1
	OV overflow of B+1
	SN sign of B+1

Name	DEC
Pseudocode	result = B-1
Description	Decrement B
Flags Affected	CO,carry out of B+0xffffffff
	OV,overflow of B-1
	SN sign of B-1

Name	NEG
Pseudocode	result = ~B+1
Description	Negate B

Flag Affected	SNsign of ~B+1
Name	ABS

Pseudocode	if (B is negative)
	result = ~B+1
	else
	result = B

Description	Absolute value of B
Flags Affected	

Name	ABS16
Pseudocode	Parallel 16 bit version of ABS
Description	Absolute value of B, for 32 bit operation
Flags Affected	

Name	CSUB

Pseudocode	if (A-B >= 0)
	result = A-B
	else
	result = A

Description	Conditional subtraction
Flags Affected	CO, carry out of A+~B+1 OV, overflow of A-B GT,A>B LT,A<B EQ,A==B SNsign of result

Name	AND
Pseudocode	result = A&B
Description	Bitwise AND
Flags Affected	EQ A==B SN bit[31] of result

Name	OR
Pseudocode	result = A|B
Description	Bitwise OR
Flags Affected	EQ,SN same as AND

Name	NAND
Pseudocode	result = ~(A&B)
Description	Bitwise NAND
Flags Affected	EQ,SN same as AND

Name	NOR
Pseudocode	result = ~(A|B)
Description	Bitwise NOR
Flags Affected	EQ,SN
same as AND	

Name	XOR
Pseudocode	result = A^B
Description	Bitwise XOR
Flags Affected	EQ,SN same as AND

Name	XNOR
Pseudocode	result = ~(A^B)
Description	Bitwise XNOR
Flags Affected	EQ,SN
same as AND	

Name	PASSA
Pseudocode	result = A
Description	Pass A
Flags Affected	EQ,SN same as AND

Name	PASSB
Pseudocode	result = B
Description	Pass B
Flags Affected	EQ,SN same as AND

Name	NOTA
Pseudocode	result = ~A
Description	Invert A
Flags Affected	EQ,SN same as AND

Name	NOTB
Pseudocode	result = ~B
Description	Invert B
Flags Affected	EQ,SN same as AND

Name	MIN

Pseudocode	if (A<B)
	result = A
	else
	result = B

Description	Return smaller of A and B
Flags Affected	GT A>B LT A<B EQ A==B SN bit[31] of result CO carry out of A+~B+1 OV overflow of A-B

Name	MIN16
Pseudocode	Parallel 16 bit version of MIN

Description	Return smaller of A and B, for 32 bit operation
Flags Affected	same as min, valid for flag[0] as well

Name	MAX

Pseudocode	if (A>B)
	result = A
	else
	result = B

Description	Return larger of A and B
Flags Affected	GT A>B LT A<B EQ A==B SN bit[31] of result CO carry out of A+~B+1 OV overflow of A-B

Name	MAX16
Pseudocode	Parallel 16 bit version of MAX
Description	Return larger of A and B, for 32 bit operation
Flags Affected	same as min, valid for flag[0] as well

Name	PENC

Pseudocode	result = 0;
	for(i = 31;i >= 0;i –;){
	if (B(i) == 1) {
	result = i+1;
	break;
	}
	}

Description	Leading One Detection.
Flags Affected	none

Name	MUXBBA
Pseudocode	result = in[A[4:0]]

Description	Use the 4 lsb of input A to multiplex the 16 inputs of B
Flags Affected	SN bit[31] of result
Name	SHIFTBBA

Pseudocode	if (A[5])
	result = B << A[4:0];
	else
	result = B << A[4:0];

Description	Shift B by A. The 16/32 bit is decided by con32 bit in config memory. In either case, the shifted out bit will be passed to the flag.
Flags Affected	bit[31] of result

1.10 DPU Interface Signals (32 bit DPU)

Signal Name	Width	Dir	Description
		Misc Signals	
braddrmask	[9:6]	input	OR mask for LSM address
clk	[0]	input	clock
clken	[0]	input	clock enable
reset	[0]	input	reset
loadconfig[a/b/o]	[0]	input	load the abo registers from config bus
arc r w	[0]	input	arc reading/writing
arc bus	[31:0]	inout	bus from arc
arc_access_[a/b/o]	[0]	input	arc reading/writing abo registers
		Input Data	
in0 -- in7	[31:0]	input	Local 8 data connections from above
in8 - in14	[31:0]	input	Local 7 data connections from below
gnet0 - gnet15	[31:0]	input	Global data connections
lsmval	[31:0]	input	LSM value
		Input Muxing	
muxasel	[2:0]	input	Select 8 output above (Shared with G0 and G1 mux)
muxbsel	[2:0]	input	Select 8 output above (Shared with select for 7 output below and self-loopback)
muxafgsel	[1:0]	input	Select above,below or global for A
muxbfsel	[0]	input	Select local or global for B
muxbcf	[0]	input	Select shift or no shift for B
muxop[a/b]sel	[1:0]	input	Select latched or unlatched or ANDed or ORed data
		Configuration Bus & Control	
[a/b/o]bus	[31:0]	input	Configuration bus
muxlat[a/b/o]sel	[0]	input	Select config bus or DPU data
lat[a/b/o][m/s]en	[0]	input	Master and Slave latch enable (Active and docking)
		ALU Signals	
shiftamt	[4:0]	input	Shift amount
shiftdir	[0]	input	Shfit direction
shiftmode	[2:0]	input	Shift mode of shifter
ctrl	[0]	input	Control as carry in of adder
conF	[2:0]	input	Flag control
opcode	[7:0]	input	Opcode for ALU
cin	[0]	input	Carry out from previous row

Output Muxing			
muxalulsmsel	[0]	input	Select ALU output or LSM value
muxoutsel	[0]	input	Select latched or unlatched
muxlsmsel	[0]	input	Select sending latched or unlatched data to lsm
Output			
flags	[1:0]	output	Flags
out	[31:0]	output	Data output
aluout	[31:0]	output	To gnet for recirculating the result
dpu_lsm_data	[31:0]	output	To LSM data
dpu_lsm_addr	[9:0]	output	To LSM address

1.11 DPU Functions – After the shifter/Mask

	16 bit	32 bit
Arithmetic	Add	Add
	Sub	Sub
	Saturation Add	Saturation Add
	Saturation Sub	Saturation Sub
	Inc	Inc
	Dec	Dec
	Neg	Neg
Logic	AND	AND
	OR	OR
	XOR	XOR
	NAND	NAND
	NOR	NOR
	XNOR	XNOR
	NOT	NOT
	PASS(NOP)	PASS(NOP)
Special Func.	ABS	ABS
	MIN	MIN
	MAX	MAX
	Rxor	Rxor
	N/A	DIV
	N/A	LFSR
	N/A	PENC
	N/A	MUXB_by_A[1]
	N/A	SHFTB_by_A[2]

NOTE: We can build a 896 bit (28*32) operator per slice.

[1] To implement N:1 multiplexers where 2<=N<=8
[2] A[4:0] is the shift amount, A[5] is the direction of the shift

CS2212 ALU Opcode Additions

The following opcodes will be added to the CS2212:

ADD8
SUB8
ADDSUB16
SUBADD16

Operation :	**ADD8:** 8 bit addition Out[7:0] = A[7:0] + B[7:0] Out[15:8] = A[15:8] + B[15:8] Out[23:16] = A[23:16] + B[23:16] Out[31:24] = A[31:24] + B[31:24]
Opcode :	? ? ? ? ? ? ? ?
Bit Granularity :	8 bit operation
Flags Affected :	No flags are available

Operation :	**SUB8:** 8 bit addition Out[7:0] = A[7:0] + ~B[7:0] + 1 Out[15:8] = A[15:8] + ~B[15:8] + 1 Out[23:16] = A[23:16] + ~B[23:16] + 1 Out[31:24] = A[31:24] + ~B[31:24] + 1
Opcode :	? ? ? ? ? ? ? ?
Bit Granularity :	8 bit operation
Flags Affected :	No flags are available

Operation :	**ADDSUB16:** 16 bit addition and subtraction Out[31:16] = A[31:16] + B[31:16] Out[15:0] = A[15:0] + ~B[15:0] + 1
Opcode :	? ? ? ? ? ? ? ?
Bit Granularity :	16 bit operation
Flags Affected :	CO, OV, EQ, SN

Operation :	**SUBADD16:** 16 bit addition and subtraction Out[31:16] = A[31:16] + ~B[31:16] + 1 Out[15:0] = A[15:0] + B[15:0]
Opcode :	? ? ? ? ? ? ? ?
Bit Granularity :	16 bit operation
Flags Affected :	CO, OV, EQ, SN

CS2212 Multiplier Output Mux Specification

The output mux for the Multiplier will be changing for the CS2212 in order to allow the A or B operand to be latched at the O register. This effectively bypasses the multiplication operation. This will require the addition of one bit to the 'muxmultlsmsel' field in the MULT CSM. The signal 'muxmultlsmsel' will select the input to the O register in the following fashion:

muxmultlsmsel[1:0]	O register or MULT output result
2'b00	multiplication output
2'b01	LSM Read Data
2'b10	A operand
2'b11	B operand

This functionality allows the user to employ the multiplier as a dynamically configurable routing resource when it is not being used for its primary function.

CS2212 Pipeline Register Specification

The CS2112 has registers that can be used as either mask registers or pipeline registers. In order to allow the user to use pipeline registers in the A and B operand paths, and use the mask registers, the CS2212 will include additional registers. These registers are inserted after the A and B input and will be referred to as 'apipe' and 'bpipe'. These registers can be bypassed by the signals 'muxapipe' and 'muxbpipe' respectively. Refer to the CS2212 DPU Block Diagram for the placement of these registers and muxes. The muxes are selected in the following fashion:

'muxapipe'	A operand	'muxbpipe'	B operand
0	register bypass	0	register bypass
1	pipeline register	1	pipeline register

CS2212 LSM Write Data Mux Specification

The CS2212 will be able to write to the LSM with the shifter output, as well as with the ALU output. In order to facilitate this added functionality, a mux will be added to the LSM write data path. This mux will be referred to as 'muxlsmwd', and will be selected by 'muxlsmwdsel' in the following fashion:

'muxlsmwd'	LSM Write Data
0	ALU Output
1	Shifter output

Refer to the CS2212 Block Diagram for the placement of 'muxlsmwd'.

2.1 General Description

The fabric is reconfigurable and it is controlled by config bits. The config bits are loaded into the fabric by issuing arc instructions (through load store) and then the config controller will transfer the config bits into the fabric's configuration plane.

The following table is to provide software information about what address does each config signal correspond to in the address space. So far the base address of the configuration has not been determined yet and so the following address will start from 0.

2.2 Details

- The top 16 bits will be the embedded address
- The top bit (bit[127]) is the parity bit. The hardware will check for even parity in each line of 128 bit. For behaviour of parity, please check ARC extension spec by Dani.
- The address below is relative to some base_address
- Each 128bit line will carry 112 bits of config data
- During config loading, the hardware can store at most 112 bit line each cycle
- For the lines which are not needed for a certain configuration, those particular line(s) can be skipped.
- The slice address will not be embedded in the current address map and will be erased from the current mapping for now. User should refer to ARC extension spec to see how to configure multiple slice in one operation.

2.3 Address Map

Address	Data			
	127 DPU/MULT Registers 32 31 0			
0x0	RESERVED			row0 register A
0x1				row0 register B
0x2				row0 register O
0x3				
0x4				row1 register A
0x5				row 1 register B
0x6				row 1 register O
0x7				
•				
•				
0x1C				row7 High mult register A
0x1D				row7 High mult register B
0x1E				row7 High mult register O
0x1F				
0x20				row7 Low mult register A
0x21				row7 Low mult register B
0x22				row7 Low mutl register O
0x23				
0x24-0x3F				RESERVED
	84 83 CSM 56 55 28 27 0			
0x40	row 0 DPU config1		row 0 DPU config0	
0x41	row 0 DPU config3		row 0 DPU config2	
0x42	row 0 DPU confiig5		row 0 DPU config4	
0x43	row 0 DPU config7		row 0 DPU config6	
0x44	row 1 DPU config1		row 1 DPU config0	
0x45	row 1 DPU config3		row 1 DPU config2	
0x46	row 1 DPU confiig5		row 1 DPU config4	
0x47	row 1 DPU config7		row 1 DPU config6	
•				
•				
0x5C	row7 MULT high config3	row7 MULT high config2	row7 MULT high config1	row7 MULT high config0
0x5D	row7 MULT high config7	row7 MULT high config6	row7 MULT high config5	row7 MULT high config4

0x5E	row7 MULT low config3	row7 MULT low config2	row7 MULT low config1	row7 MULT low config0
0x5F	row7 MULT low config7	row7 MULT low config6	row7 MULT low config5	row7 MULT low config4
0x60-0x7F	RESERVED			

DPU config Details	
2:0	muxasel
5:3	muxbsel
7:6	muxbfsel
9:8	muxafgsel
14:10	shiftamt
15	shiftdir
23:16	opcode
24	muxbcfsel
25	latapipe
26	latbpipe
27	latopipe
29:28	muxopasel
31:30	muxopbsel
34:32	shiftmode
37:35	conf
38	muxalulsmsel
39	muxoutsel
40	ctrl
41	lsm read/write enable (overlapped for both read and write enables, even row for read and odd row for write)
55:42	RESERVED

MULT config Details	
2:0	muxasel
5:3	muxbsel
7:6	muxbfsel
9:8	muxafgsel
11:10	muxbcfsel
12	muxafghsel
13	muxopasel
14	muxopbsel
15	muxmultlsmsel
16	muxoutsel
17	latapipe
18	latbpipe
19	latopipe
20	lsm read/write enable (just like the editilepipe, we use the read enable for the 3rd lsm from the 6th dpu, write enable from the high multiplier)
27:21	RESERVED

	CONTROL CONFIG
0x80	Tile A PLA product term 0
0x81	Tile A PLA product term 1
0x82	•
0x83	•
....	•
0x9D	•
0x9E	Tile A PLA product term 30
0x9F	Tile A PLA product term 31
0xA0-0xBF	RESERVED
0xC0	Tile A Tile Basis Control
0xC1	Tile A Row Basis Control L0
0xC2	Tile A Row Basis Control L1
0xC3	Tile A Row Basis Control L2
0xC4	Tile A Row Basis Control L3
0xC5-0xFF	RESERVED

PLA X product Term Y Details (where X in (A,B,C,D) & Y in [0,15])	
15:0	let bit position be variable b. This means product term Y in PLA X ANDs ~input[b]
31:16	let bit position be variable b. This means product term Y in PLA X ANDs input[(b-16)]
63:32	let bit position bit variable b. This means result[(b-32)] in PLA X ORs product term Y
111:64	Reserved

Tile Basis Control Details	
1:0	SRBDMA: muxdmasetsel
33:2	GNET: hortnetsel:{net5,net4,...,net0}
45:34	GNET: vertnetsel: {net2,net1,net0}
53:46	LSMNET: raddr {transen3to4,trien3,transen2to3,trien2,transen1to2,trien1,transen0to1,trien0}
61:54	LSMNET: rdata {transen3to4,trien3,transen2to3,trien2,transen1to2,trien1,transen0to1,trien0}
69:62	LSMNET: waddr {transen3to4,trien3,transen2to3,trien2,transen1to2,trien1,transen0to1,trien0}
77:70	LSMNET: wdata {transen3to4,trien3,transen2to3,trien2,transen1to2,trien1,transen0to1,trien0}
Row Basis Control Details	
Lx[1:0]	row(2x+1)-(2x): SRBR clken[0] x 2
Lx[7:2]	row(2x+1)-(2x): SRBR csmaddrmask[2:0] x 2
Lx[11:8]	row(2x+1)-(2x): SRBR muxinterruptsetsel[1:0] x2
Lx[13:12]	row(2x+1)-(2x): SRBR inten[0] x 2
Lx[37:14]	row(2x+1)-(2x): PLA_in_sel[5:0] x 4
Lx[39:38]	row(2x+1)-(2x): lsmregsel[0] x 2
Lx[47:40]	row(2x+1)-(2x): HSTATE_sel[1:0] x 4
Lx[51:48]	row(2x+1)-(2x): HSTATE_regsel[0] x 4
Lx[55:52]	lsm x: addrmatch[3:0]
Lx[59:56]	lsm x: matchen [3:0]
Lx[61:60]	lsm x: rmode [1:0]
Lx[63:62]	lsm x: wmode [1:0]
Lx[71:64]	row(2x+1)-(2x): SRB_reginit[3:0] x2
0x100-0x1FF	Tile B Config
0x200-0x2FF	Tile C Config
Slice Basis Config	
0x300[1:0]	Done Mux Select
Other Slice Address (Unused in the current implementation)	
0x400-0x7ff	Slice 1 Config
0x800-0xbff	Slice 2 Config
0xc00-0xfff	Slice 3 Config
Fabric Basis Config	
0x380-0x383	Slice 0 selections
0x780-0x783	Slice 1 selections
0xb80-0xb83	Slice 2 selections
0xf80-0xf83	Slice 3 selections
Fabric Basis Config Details	
L0 7:0	tile A gnet tristate enable 1 bit x 8
L0 23:8	tile A hctl net tristate enable 1 bit x 16
L1 7:0	tile B gnet tristate enable 1 bit x 8
L1 23:8	tile B hctl net tristate enable 1 bit x 16
L2 7:0	tile C gnet tristate enable 1 bit x 8
L2 23:8	tile C hctl net tristate enable 1 bit x 16
L3 1:0	PIO: control input mux sel (input to fabric) 2 bit x 1
L3 5:2	PIO: control output mux one hot sel (output from fabric) 1 bit x 4
L3 7:6	PIO: data input mux sel (input to fabric) 2 bit x 1
L3 11:8	PIO: data output mux one hot sel (output from fabric) 1 bit x 4

CSMMULT	muxasel	same as dpu
	muxbsel	same as dpu
	muxafgsel	same as dpu
	muxbfsel	same as dpu
	muxbcfsel	0 low 16 bit 1,2 high 16 bit 3 high 24 bit (for 24 bit multiply)
	muxafghsel	0 low 16 bit 1 high 16 bit
	latapipe	same as dpu
	latbpipe	same as dpu
	latopipe	same as dpu
	muxopasel	0 latched value 1 unlatched value
	muxopbsel	
	muxmultlsmsel	0 mult output 1 lsm value
	muxoutsel	0 latched value 1 unlatched value
LSM	rmode	0 byte 1 word 2 dword 3 dword
	wmode	
	wen	0 write disable 1 write enable
	addrmatch	the 4 bit of the address provided by DPU will be match with these values in order to enable a read/write operation
	matchen	individually enable(1)/disable(0) these 4 bits of matching address
	lsmregsel	the signals from DPU (addr/data) and CSM (wen) are selectively registerable. 0 = registered 1 = unregistered
PLA	PLA_in_sel	please see below
GNET	hortnetsel	select the 9 datapath output from a tile (7 dpu and 2 mult), there are 6 of this in each tile (we have 6 horizontal global net and 3 vertical global net) the horizontal global net drives, besides the horizontal global net, also drives the input for vertical global mux 0 dpu 0 1 dpu 1 2 dpu 2 3 dpu 3 4 dpu 4 5 dpu 5 6 dpu 6 7 mult 7 others mult 8
	vertnetsel	select from the 6 horizontal global net or the 'local' horizontal global net 0,1,2 (note those 2 values can be different since the 'local' tristate can be disabled)
	hortnettrien	the tristate enable for the horizontal global net. Each global net has 4 tristate driver sitting on it. The tristate enable is a one hot value, one bit driven by each slice.

Name of Unit	Name of signal	Explanation
DPU/MULT	a/b/o register	Value of the pipeline registers in the DPU and MULT
CSMDPU	muxasel	Overlap with muxF, muxG0 and muxG1 selects: val muxA muxF muxG0 muxG1 0 previous 8 itself gnet0 gnet8 1 previous 7 next 1 gnet1 32'h0 2 previous 6 next 2 gnet2 32'h0 3 previous 5 next 3 gnet3 32'h0 4 previous 4 next 4 gnet4 LSM value 5 previous 3 next 5 gnet5 ALU out (before regO) 6 previous 2 next 6 gnet6 {DPUout[30:0],cin} 7 previous 1 next 7 gnet7 32'h0
	muxbsel	same as above
	muxafgsel	00 selects MuxA 01 selects MuxF 10 selects MuxG0 11 selects MuxG1
	muxbfsel	same as above
	muxbcfsel	0 selects MuxBF 1 selects Shift Output
	latapipe	0 register A won't latch input 1 register A will latch input
	latbpipe	0 register B won't latch input 1 register B will latch input
	latopipe	0 register O won't latch input 1 register O will latch input
	muxopasel	00 selects Latched Data 01 selects Unlatched Data 10 selects ANDed Data 11 selects ORed Data
	muxopbsel	
	shiftamt	Shift amount presented to shifter
	shiftdir	Please see s:\doc\teams\dpu\dpu_specX.doc section 1.12
	shiftmode	
	conf	Flag select 0 previous flag 1 RXOR 2 GT 3 LT 4 EQ 5 SN 6 CO 7 OV
	ctrl	Control carry in for addcnt,subcnt,etc ops
	opcode	please see s:/doc/teams/dpu/dpu_spec#.doc
	muxalulsmsel	0 selects ALU output 1 selects LSM data
	muxoutsel	0 selects Latched Data. 1 selects Unlatched Data

		4'bxxx1 or 4'b0000: slice 0 drives the tristate bus 4'bxx10: slice 1 drives the tristate bus 4'bx100: slice 2 drives the tristate bus 4'b1000: slice 3 drives the tristate bus
LSMNET	tristate enable	there are 4 lsmnet, read/write address/data. each has 15 tristate enable since there are 16 lsm in one tile. 0 disable 1 enable
	transmission gate enable	tristate enable 0 disable 1 enable
SRBDMA	muxdmasetsel	This is the mux control before the DMA request flop. There is one of this for each tile. It takes the state value from the 0th row of each tile. 0 0th row state bit [3] 1 0th row state bit [2] 2 0th row dpuflag[1] 3 ground
SRBR	There is one SRBR for each row. So the following signals all are on a row basis.	
	clken	this is a row basis clock enable 0: use SRB[3] as clock enable 1: always clock the SRB
	csmaddrmask	this is the 'AND' mask before the state bits drives to the csm
	muxinterruptsetsel	the mux select for the mux before the interrupt register and it drives the 'SET' input of the flop 0 local state [3] 1 local state [2] 2 dpuflag[1] 3 ground
	inten	Interrupt enable 0: disable 1: enable
	SRB reg_initval	the initial value of the state bits
Hstate	hstate_sel	please see below
	hstate_trien	similar to horizontal net tristate enable, it's value is set from one bit from each slice and then prioritized as follow: 4'bxxx1 or 4'b0000: slice 0 drives the tristate bus 4'bxx10: slice 1 drives the tristate bus 4'bx100: slice 2 drives the tristate bus 4'b1000: slice 3 drives the tristate bus
	hstate_regsel	selectively register the hstate bit before entering the pla input 0 registered 1- unregistered.
PIO	mux sel (input to fabric)	0: selects bank 0 1: selects bank 1 2: selects bank 2 3: selects bank 3
	mux sel (output from fabric)	this is one hot encoding, each bit coming from the corresponding bit position of each slice's fabric level config: 4'bxxx1 or 4'b0000: output from slice 0 4'bxx10: output from slice 1 4'bx100: output from slice 2

		4'b1000: output from slice 3

- Input_mux Slice 0 PLA0 (each bit has individual enables)
- PLAin[0] is the multiplexer input associated with DPU0
- PLAin[1] is the multiplexer input associated with DPU0
- STATE0[n] is the primary state output of slice 0
- LSTATE0[n] is the lower state output of slice 0
- FLAG[n] is the primary flag output of slice 0
- LSTATE0[n] is the lower flag output of slice 0
- HBIT0[n] is the Horizontal State bus
- HBIT0[n] needs an optional register before the input mux.
- IO[7:0] are 8 I/O bits coming from the pins associated with each slice.
- Bit 0 input, DATA, is the hortnet mux output from the datapath. Each tile has 8 hortnet mux. The control will pick up the lower 16 bit hortnet mux 7 from each tile. Notice it is the value at the output of the mux 7, input of the tristate.
- Tile D is taken out and so data it provides (STATE*[31:24],FLAG[27:21]...) are connected to ground)

The PLA_in_sel are 64to1 mux select. There are 16 of them for each PLA (each bit are individually controlled).

	bit 63:32	bit32:4	bit3	bit2	bit1	bit0
PLAin0	STATE0[31:0]	FLAG0[27:0]	HBIT0[0]	LSTATE0[0]	IO[0]	DATA[0]
PLAin2	STATE0[31:0]	FLAG0[27:0]	HBIT0[2]	LSTATE0[1]	IO[1]	DATA[2]
PLAin4	STATE0[31:0]	FLAG0[27:0]	HBIT0[4]	LSTATE0[2]	IO[2]	DATA[4]
PLAin6	STATE0[31:0]	FLAG0[27:0]	HBIT0[6]	LSTATE0[3]	IO[3]	DATA[6]
PLAin8	STATE0[31:0]	FLAG0[27:0]	HBIT0[8]	LSTATE0[4]	IO[4]	DATA[8]
PLAin10	STATE0[31:0]	FLAG0[27:0]	HBIT0[10]	LSTATE0[5]	IO[5]	DATA[10]
PLAin12	STATE0[31:0]	FLAG0[27:0]	HBIT0[12]	LSTATE0[6]	IO[6]	DATA[12]
PLAin14	STATE0[31:0]	FLAG0[27:0]	HBIT0[14]	start	IO[7]	DATA[14]
PLAin1	STATE0[31:0]	FLAG0[27:0]	HBIT0[1]	LFLAG0[0]	IRQ[0]	DATA[1]
PLAin3	STATE0[31:0]	FLAG0[27:0]	HBIT0[3]	LFLAG0[1]	IRQ[1]	DATA[3]
PLAin5	STATE0[31:0]	FLAG0[27:0]	HBIT0[5]	LFLAG0[2]	IRQ[2]	DATA[5]
PLAin7	STATE0[31:0]	FLAG0[27:0]	HBIT0[7]	LFLAG0[3]	IRQ[3]	DATA[7]
PLAin9	STATE0[31:0]	FLAG0[27:0]	HBIT0[9]	LFLAG0[4]	IRQ[4]	DATA[9]
PLAin11	STATE0[31:0]	FLAG0[27:0]	HBIT0[11]	LFLAG0[5]	IRQ[5]	DATA[11]
PLAin13	STATE0[31:0]	FLAG0[27:0]	HBIT0[13]	LFLAG0[6]	IRQ[6]	DATA[13]
PLAin15	STATE0[31:0]	FLAG0[27:0]	HBIT0[15]	DMA_REQ	IRQ[7]	DATA[15]

- Horz_mux Slice0 PLA0 (each bit has individual enables)
- HMUX[0] is one of 16 Horizontal State Lines;
- Each mux needs also tristate enable.

	bit3	bit2	bit1	bit0
HMUX0[0]	STATE0[24]	STATE0[16]	STATE0[8]	STATE0[0]
HMUX0[2]	STATE0[25]	STATE0[17]	STATE0[9]	STATE0[1]
HMUX0[4]	STATE0[26]	STATE0[18]	STATE0[10]	STATE0[2]
HMUX0[6]	STATE0[27]	STATE0[19]	STATE0[11]	STATE0[3]
HMUX0[8]	STATE0[28]	STATE0[20]	STATE0[12]	STATE0[4]
HMUX0[10]	STATE0[29]	STATE0[21]	STATE0[13]	STATE0[5]
HMUX0[12]	STATE0[30]	STATE0[22]	STATE0[14]	STATE0[6]
HMUX0[14]	STATE0[31]	STATE0[23]	STATE0[15]	STATE0[7]

HMUX0[1]	LSTATE0[24]	LSTATE0[16]	LSTATE0[8]	LSTATE0[0]
HMUX0[3]	LSTATE0[25]	LSTATE0[17]	LSTATE0[9]	LSTATE0[1]
HMUX0[5]	LSTATE0[26]	LSTATE0[18]	LSTATE0[10]	LSTATE0[2]
HMUX0[7]	LSTATE0[27]	LSTATE0[19]	LSTATE0[11]	LSTATE0[3]
HMUX0[9]	LSTATE0[28]	LSTATE0[20]	LSTATE0[12]	LSTATE0[4]
HMUX0[11]	LSTATE0[29]	LSTATE0[21].	LSTATE0[13]	LSTATE0[5]
HMUX0[13]	LSTATE0[30]	LSTATE0[22]	LSTATE0[14]	LSTATE0[6]
HMUX0[15]	LSTATE0[31]	LSTATE0[23]	LSTATE0[15]	LSTATE0[7]

31 OCT 2005

Claims

What is claimed is:

1. A reconfigurable chip including:

multiple reconfigurable functional units adapted to implement different functions, the reconfigurable functional units including multiplexers, at least one shifter unit and at least one arithmetic logic unit, the reconfigurable functional units being configured by a reconfigurable functional unit instruction, the instruction controlling the configuration of the multiplexers, shifter unit and arithmetic logic unit; and

interconnect elements adapted to selectively connect together some of the reconfigurable functional units.

2. The reconfigurable chip of claim 1 wherein the reconfigurable functional unit instruction is divided into a number of fields, including a multiplexer field, a shifter unit field and an arithmetic logic unit field.

3. The reconfigurable chip of claim 1 wherein the reconfigurable functional unit instruction controlling the configuration of the multiplexers, shifter unit and arithmetic logic unit is provided by a state machine.

4. The reconfigurable chip of claim 1 further comprising an instruction memory storing multiple instructions for the reconfigurable functional units.

5. A reconfigurable chip including:

multiple reconfigurable functional units, the reconfigurable functional units including multiplexers, at least one shifter unit and at least one arithmetic logic unit, the shifter unit adapted to allow the arithmetic logic units to operate on different bits within the word-length input data of the reconfigurable functional unit; and

31 OCT 2005

interconnect elements adapted to selectively connect together some of the reconfigurable functional units, the interconnect elements adapted to transfer word-length data.

6. The reconfigurable chip of claim 5 wherein the configuration of the multiplexers, shifter unit and arithmetic logic unit is changed by reconfigurable functional unit instructions.

7. A reconfigurable chip including:

multiple reconfigurable functional units, the reconfigurable functional units including multiplexers, at least one shifter unit and at least one arithmetic logic unit, the shifter unit being configurable with a number of modes; and

interconnect elements adapted to selectively connect together some of the reconfigurable functional units.

8. The reconfigurable chip of claim 7 wherein the shifter modes include modes other than logical and arithmetic left and right shifts.

9. The reconfigurable chip of claim 7 wherein at least one mode rearranges blocks of an the input word.

10. The reconfigurable chip of claim 7 wherein one of the modes comprises swapping some of the groups of bits with other groups of bits.