Signal Routing

Objectives

After completing this module, you will be able to:

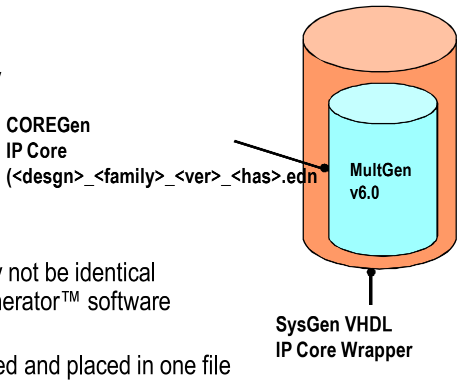
- Describe how signals are converted through Gateway In blocks
- State supporting blocks for routing signals
- List the blocks which use additional resources and which do not
- Identify circumstances under which blocks consumes additional resources

Outline

- **Signal Conversion**
- Bit Picking
  - Reinterpret Block
  - Convert Block
  - Concat Block
  - Slice Block
- BitBasher Block
- Expression Block
- Summary

IP Wrappers

- Every IP core has a wrapper to interface between the Simulink™ tool and the hardware
- Each SysGen block has an RTL HDL wrapper to:
  - Extend the IP core functionality
  - Simplify the IP core interface
- Supports fixed-point arithmetic
  - Number of bits, binary point
  - Overflow and quantization
  - Valid bit control on some cores
- **Note:** a Simulink parameter may not be identical to the corresponding CORE Generator™ software parameter
- Wrapper models will be generated and placed in one file

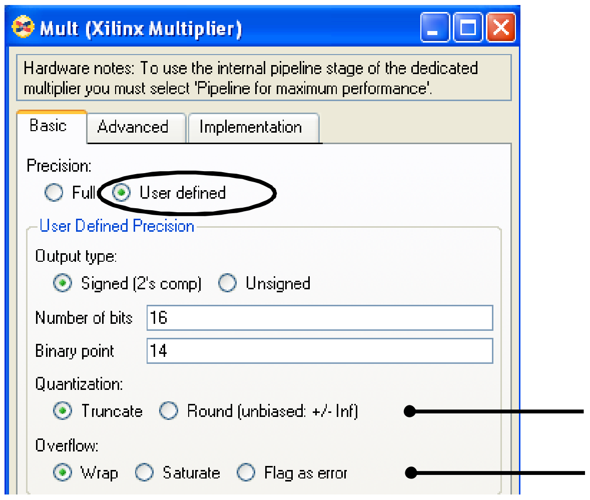


Implications of IP Wrappers

System-level abstraction is very expressive and powerful, but it comes at the expense of hardware. Be aware!

- Saturation arithmetic and rounding require hardware (full adder and comparator)
- Excess latency is implemented with a shift register (SRL16/32) on the core output
- Some System Generator blocks perform implicit conversion of inputs
  - Unsigned to signed
  - Sign extension
  - Zero padding
  - SysGen mask parameters may not be identical to COREGen parameters
- The valid-bit pipeline parallels the datapath; typically implemented by using SRL16/32

Quantization and Overflow



Hardware notes: To use the internal pipeline stage of the dedicated multiplier you must select 'Pipeline for maximum performance'.

Precision: Full [✓ User defined]

Output type: [✓ Signed (2's comp)] Unsigned

Number of bits: 16

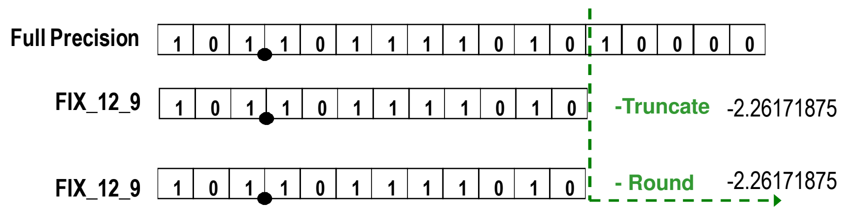
Binary point: 14

Quantization: [✓ Truncate] Round (unbiased: +/- Inf)

Overflow: [✓ Wrap] Saturate Flag as error

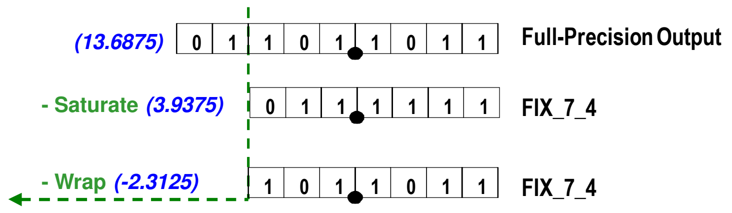
Quantization

- Quantization occurs when the number of fractional bits is insufficient to represent the fractional portion of a value
- You can:
  - Truncate: Discard bits to the right of the least significant bit
  - Round: Round to the nearest representable value or to the value farthest from zero if there are two equidistant nearest representable values



Overflow

- Overflow occurs when a value lies outside the representable range
- You can:
  - Saturate to the largest positive (or maximum negative) value
  - Wrap the value (that is, discard any significant bits beyond the most significant bit in the fixed-point number)
  - Flag an overflow as a Simulink error during simulation



Quantization and Overflow

- Whatever option is selected, the generated HDL model and Simulink model will behave identically
- This also means that rounding and saturation will use FPGA resources

Outline

- Signal Conversion
- **Bit Picking**
  - Reinterpret Block
  - Convert Block
  - Concat Block
  - Slice Block
- BitBasher Block
- Expression Block
- Summary

Picking Bits

- Why do you do it?
  - To reinterpret unsigned data as signed or the converse
  - To combine two data buses together to form a new bus
  - To force a conversion of data type, including the number of bits and binary bits
  - To extract certain bits of data, especially when there is bit growth

Xilinx Blocks

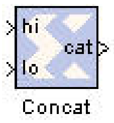
- Reinterpret
  - Available in basic elements, math, and index libraries

force
Reinterpret

- Convert
  - Available in basic elements, data types, math, and index libraries

cast
Convert

- Concat
  - Available in basic elements, data types, and index libraries



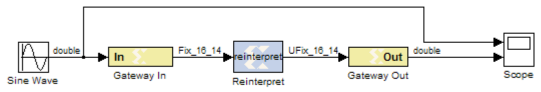
- Slice
  - Available in basic elements, control logic, data types, and index libraries

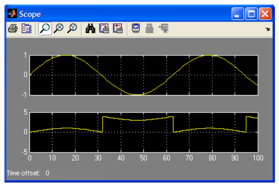
Outline

- Signal Conversion
- Bit Picking
  - **Reinterpret Block**
  - Convert Block
  - Concat Block
  - Slice Block
- BitBasher Block
- Summary

Reinterpret Block

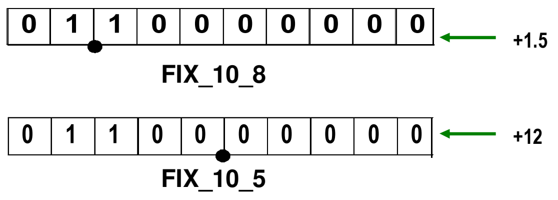
- This block forces its output into a new type without any regard for retaining the numerical value represented by the input
- The total number of bits in equals the total number of bits out
- This block allows unsigned data to be reinterpreted as signed data and the converse
- This block also allows scaling of the data through repositioning of the binary point
- Does not use the Xilinx LogiCORE™ solution and hardware resources





Reinterpret Block

- Reinterpret the FIX_10_8 number and force binary point to position 5

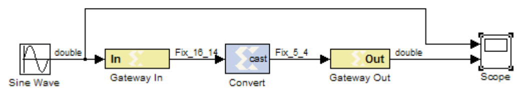


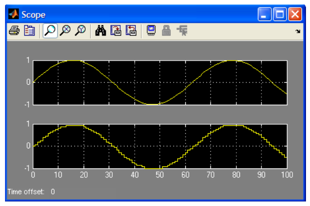
Outline

- Signal Conversion
- Bit Picking
  - Reinterpret Block
  - **Convert Block**
  - Concat Block
  - Slice Block
- BitBasher Block
- Summary

Convert Block

- The Xilinx convert block converts each input sample into a number of a desired arithmetic type
  - A number can be converted to a signed (twos complement), unsigned value, or Boolean
  - The total number of bits and the binary point are specified by the user
  - Overflow and quantization options apply to the output value
  - Does not use Xilinx LogiCORE™ solution resources, but it may use additional hardware depending on the overflow and quantization options



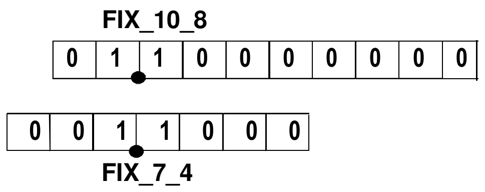


Convert Block

- What is it doing?
  - You specify the total number of bits, where the binary point is located, and the arithmetic type (signed or unsigned)
  - First, it lines up the binary point between the input and output port types
  - Next, the total number of bits and the binary point that you specify are used, and if overflow and quantization options are used, the output may change, as opposed to dropping bits

Convert Block

- The following through the convert block would result in the same value using a different number of bits and binary point

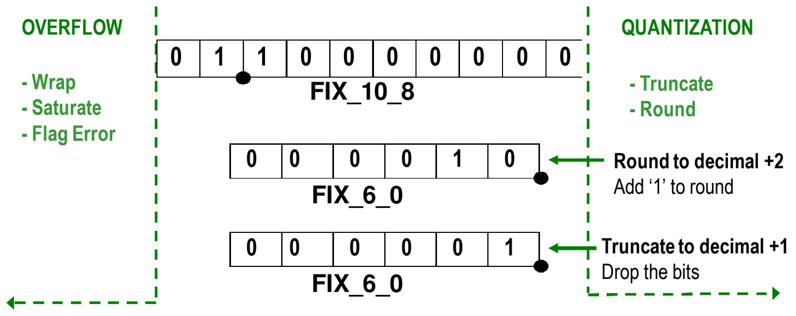


Convert Block

- Saturating the overflow can change the fractional number to get the saturated value
- Rounding the quantization can also affect the value to the left of the binary point (the whole number)

Convert Block

- When you convert to a Fix_6_0, how do you get two different values?



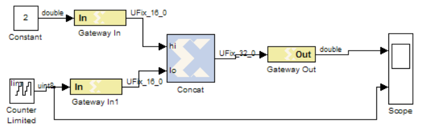
Outline

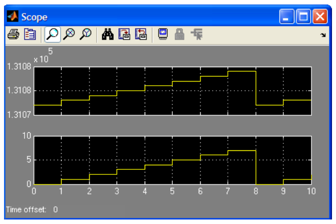
- Signal Conversion
- Bit Picking
  - Reinterpret Block
  - Convert Block
  - **Concat Block**
  - Slice Block
- BitBasher Block
- Summary

Concat Block

hi
lo
cat
Concat

- Performs a concatenation of up to 16 bit vectors
- All inputs must be unsigned integers
  - That is, unsigned numbers with binary points at position zero
- The Reinterpret block provides signed-to-unsigned conversion capabilities that can extend the functionality of the concat block
- Does not use the Xilinx LogiCORE™ solution and hardware resources



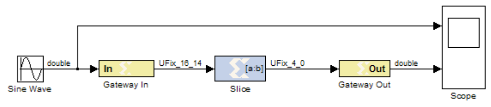


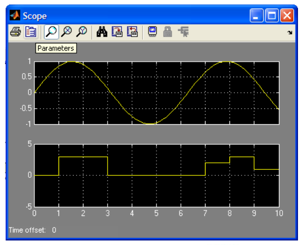
Outline

- Signal Conversion
- Bit Picking
  - Reinterpret Block
  - Convert Block
  - Concat Block
  - **Slice Block**
- BitBasher Block
- Summary

Slice Block

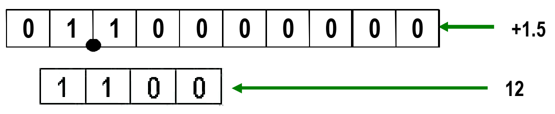
- The Xilinx Slice block allows you to slice off a sequence of bits from your input data and create a new data value
- The output data type is unsigned, with its binary point at zero
  - It can be of type Boolean when the output size is one bit



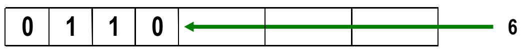


Slice Block

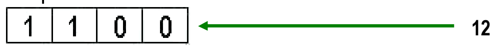
- Take a slice of the FIX_10_8 number by taking a four-bit slice and offsetting the bottom bit of the slice by five bits



- Upper-bit location + width: Offset of top bit from MSB = 0 and width = 4



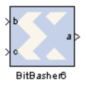
- Two-bit locations: Offset of top bit from MSB of input = -1 and offset of bottom bit from LSB of input = 5



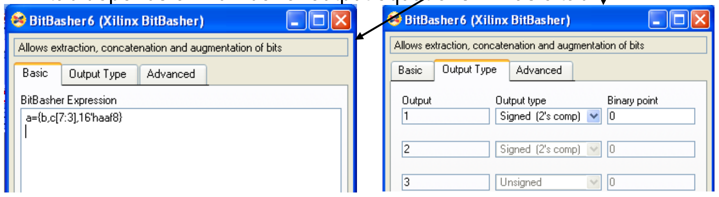
Outline

- Signal Conversion
- Bit Picking
  - Reinterpret Block
  - Convert Block
  - Concat Block
  - Slice Block
- **BitBasher Block**
- Expression Block
- Summary

BitBasher Block



- Bit manipulation and augmentation through textual specification
  - Based on Verilog syntax
  - Supports Concatenation, Slicing and Repeat operators
  - Allows augmentation with constants specified as binary, decimal, octal or hex
  - At least one of the inputs must be from input port
  - Supports up to four outputs
    - Number of Output type and Binary point fields available in Output Type tab depends on number of output equations in Basic tab



Allows extraction, concatenation and augmentation of bits

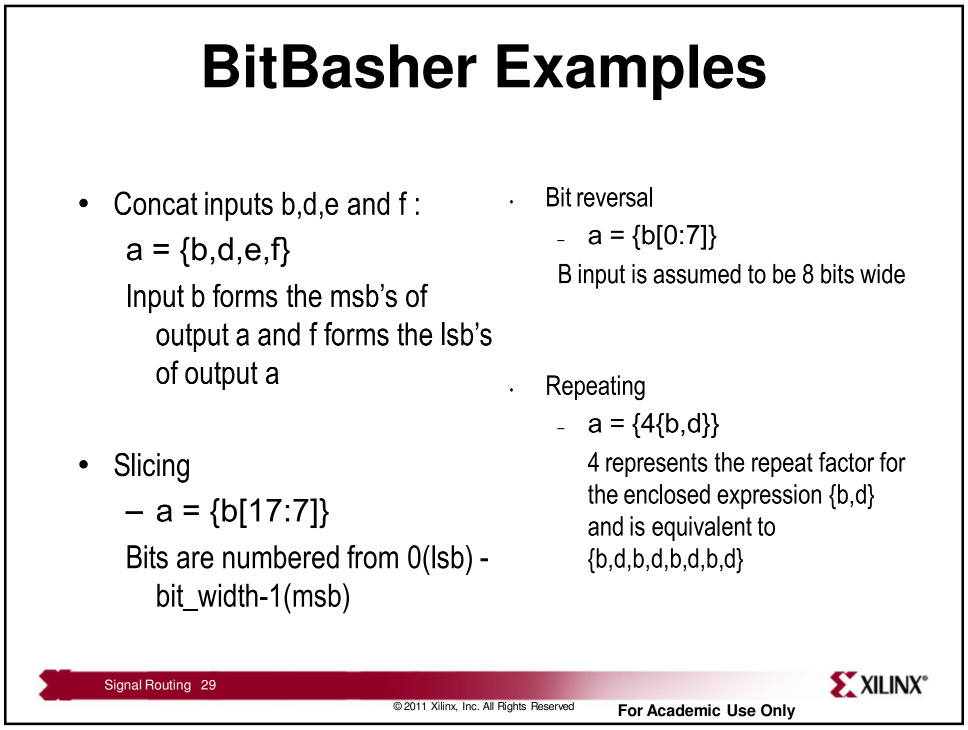
BitBasher Expression

```
a={b,c[7:3],16'haaf8}
```

BitBasher6 (Xilinx BitBasher)

Allows extraction, concatenation and augmentation of bits

Output	Output type	Binary point
1	Signed (2's comp)	0
2	Signed (2's comp)	0
3	Unsigned	0



BitBasher Examples

- Concat inputs b,d,e and f :

  a = {b,d,e,f}

  Input b forms the msb's of output a and f forms the lsb's of output a

- Slicing
  - a = {b[17:7]}

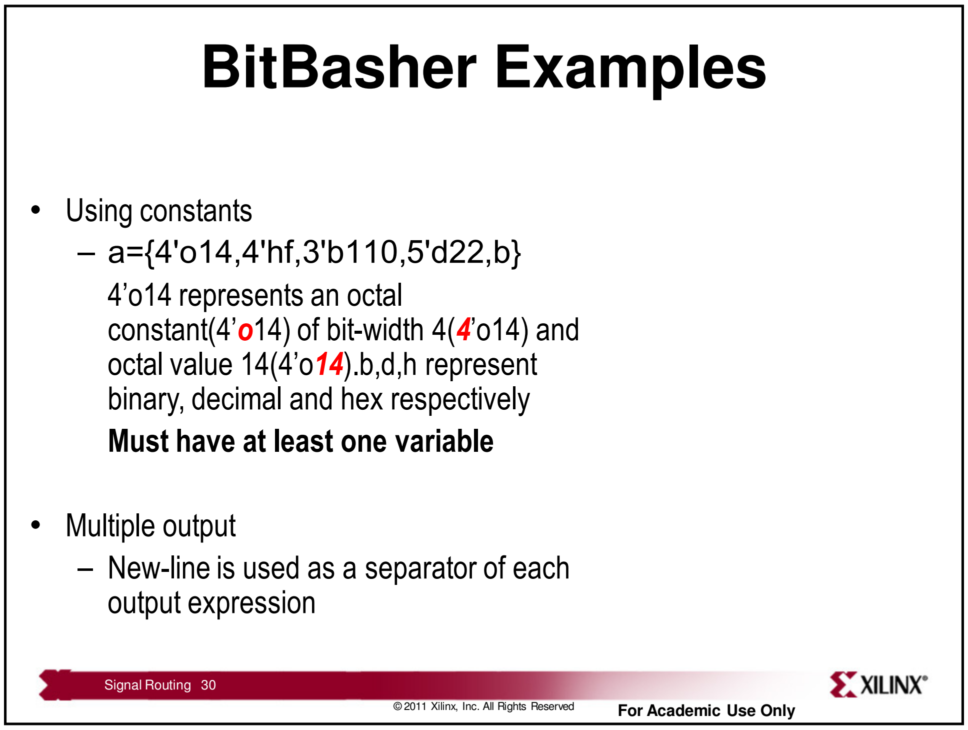
  Bits are numbered from 0(lsb) - bit_width-1(msb)

- Bit reversal
  - a = {b[0:7]}

  B input is assumed to be 8 bits wide

- Repeating
  - a = {4{b,d}}

    4 represents the repeat factor for the enclosed expression {b,d} and is equivalent to {b,d,b,d,b,d,b,d}



BitBasher Examples

- Using constants
  - a={4'o14,4'hf,3'b110,5'd22,b}

    4'o14 represents an octal constant(4'**o**14) of bit-width 4(**4**'o14) and octal value 14(4'o**14**).b,d,h represent binary, decimal and hex respectively

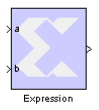
    Must have at least one variable

- Multiple output
  - New-line is used as a separator of each output expression

Outline

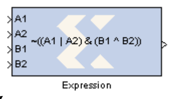
Expression Block

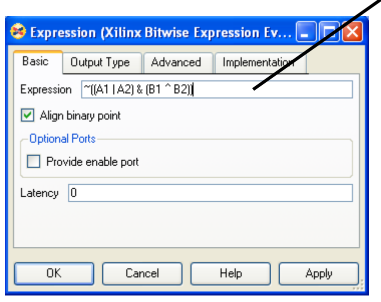
- This block performs a bitwise logical expression
- The expression is specified with operators
  - And - & (highest precedence)
  - Or - |
  - Not - ~
  - Xor - ^ (lowest precedence)
  - Precedence can be changed by using parenthesis
- The number of input ports is inferred from the expression
  - Maximum of 16 possible input ports
- The input port labels are identified from the expression, and the block is subsequently labeled accordingly
  - Block parameters
    - Expression: Bitwise logical expression
    - Align Binary Point: Specifies that the block must align binary points automatically. If not selected, all inputs must have the same binary point position



Expression Block

- Entering the following expression in the block parameters Expression field will generate the resulting symbol
  - ~((A1 | A2) & (B1 ^ B2))




Expression (Xilinx Bitwise Expression Ev...

Basic | Output Type | Advanced | Implementation

Expression: ~((A1 | A2) & (B1 ^ B2))

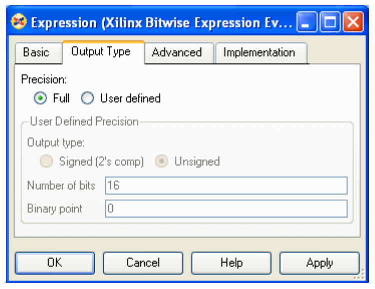
[✓] Align binary point

Optional Ports

[ ] Provide enable port

Latency: 0

OK | Cancel | Help | Apply


Expression (Xilinx Bitwise Expression Ev...

Basic | Output Type | Advanced | Implementation

Precision: [✓ Full] User defined

User Defined Precision

Output type: Signed (2's comp) Unsigned

Number of bits: 16

Binary point: 0

OK | Cancel | Help | Apply

Outline

- Signal Conversion
- Bit Picking
  - Reinterpret Block
  - Convert Block
  - Concat Block
  - Slice Block
- BitBasher Block
- Expression Block
- **Summary**

Knowledge Check

Knowledge Check

- What is quantization? Why does it occur? State the two options available to handle it
- What is an overflow? Why does it occur? State the three options available to handle it

Answers

- What is quantization? Why does it occur? State the two options available to handle it
  - Quantization is a process of handling higher-precision number representation with a lower-precision number representation
  - In the Simulink tool, the numbers are represented in double-precision; whereas in the Xilinx blockset, the numbers are represented in fixed point
  - Truncate and Rounding are the two available options to handle it
- What is an overflow? Why does it occur? State the three options available to handle it
  - An overflow occurs when a large number is represented in a smaller range representation
  - In the Simulink tool, the numbers are represented in double-precision; whereas in the Xilinx blockset, the numbers are represented in fixed point
  - Saturate, Wrap the value, and Flag an Error are the three available options

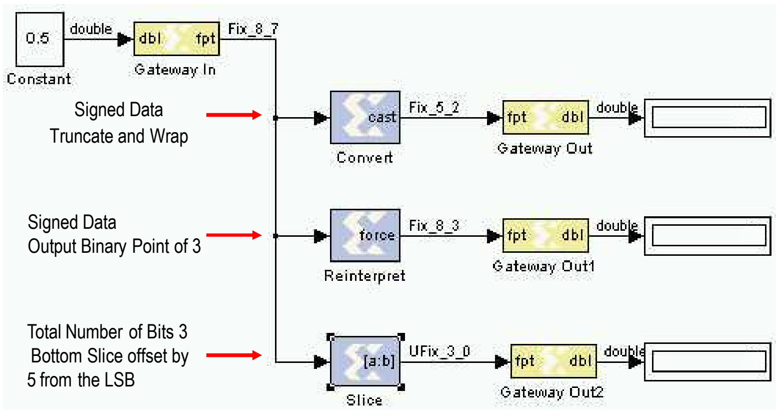
Knowledge Check

- What is the purpose of HDL wrapper?
- Why do you need bit picking? State the four blocks available for this purpose

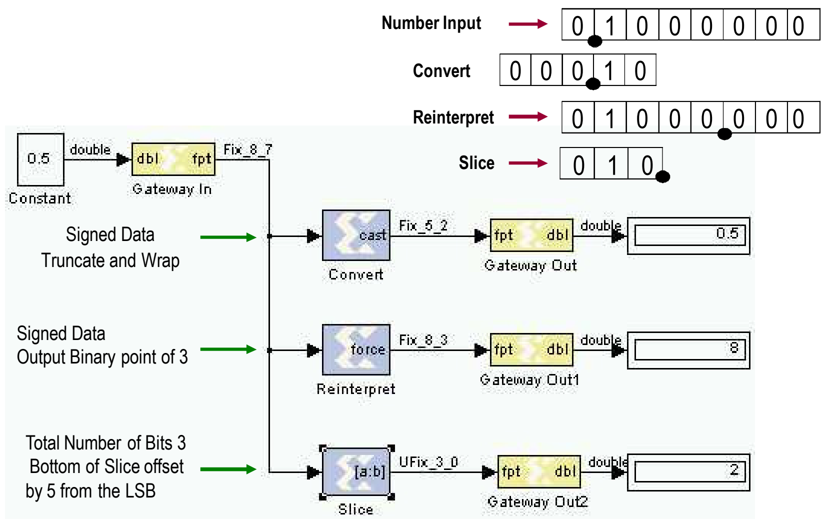
Answers

- What is the purpose of HDL wrapper?
  - Extend the IP core/block functionality, allowing various data types
  - Simplify the IP core interface, providing only necessary ports on a block
    - For example, DSP48 macro
- Why do you need bit picking? State the four blocks available for this purpose
  - There may be a need to:
    - Combine two data buses together to form a new bus
    - Force a conversion of data type, including the number of bits and binary bits
    - Reinterpret unsigned data as signed or the converse
    - Extract certain bits of data, especially when there is bit growth
  - The four blocks available are: Concat, Convert, Reinterpret, Slice

Knowledge Check



Answers



Summary

- Quantization and overflow options are available when the output of a block is user defined
- Quantization occurs when the number of fractional bits is insufficient to represent the fractional portion of a value
- Overflow occurs when a value lies outside the representable range
- Bit picking blocks allow combining of multiple busses into a single bus, force a conversion of data type without changing the number of bits, extract bits, and convert the number into different format
- BitBasher block allows bit manipulation and augmentation through textual specification
- BitBasher block may have up to four outputs
- Expression block provides a convenient way to implement complex combinatorial expression

Where Can I Learn More?

- Software documentation
  - *Simulink Browser → Help → Simulink Help → Xilinx System Generator*
    - *BlockSet Reference Pages → Block Reference Pages*
      - *Bitbasher, Concat, Convert, Expression, Gateway In, Reinterpret, Slice*
- On-line documentation
  - *www.xilinx.com/products/software/sysgen/sysgen_ug.pdf*
- Support Website
  - DSP Website: http://www.xilinx.com/dsp