Correlator IP Core

April 2005 IP Data Sheet

Features

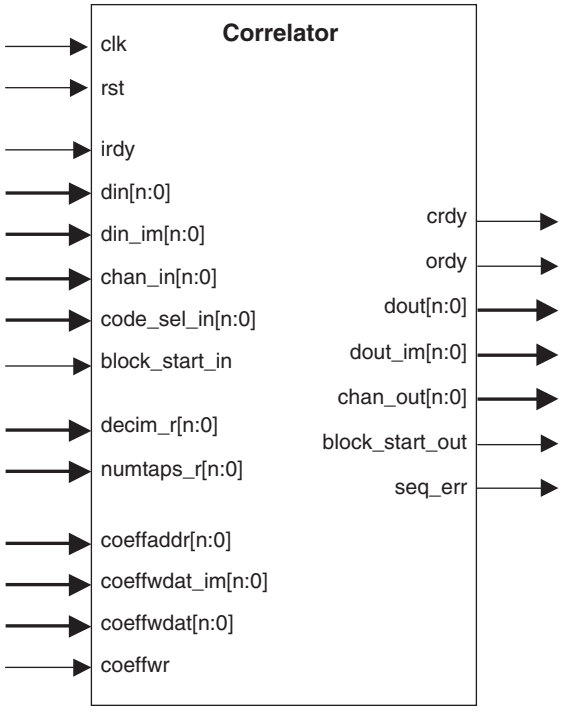
- Supports 1- to 8-Bit Input Data Width
- Supports 1 to 256 Channels
- Supports a Correlation Window from 8 to 2048 Taps
- Supports Oversampled Input Data from 2x to 8x
- Supports Real Correlations for Either Signed or Unsigned Data
- Supports Complex Correlations for Signed Data
- Allows the User to Tune the Performance of the Design by Specifying the Values of Several Parameters
- Provides a Selectable Input FIFO for Maximum Data Throughput
- Allows the User to Specify the Number of Coefficient Sequences Desired, from 1 to 256

Introduction

The function of this core is to correlate an incoming data stream to a stored binary pattern called a code sequence or coefficient sequence. The data stream may be binary or multi-valued, either signed or unsigned, and is provided to the core one sample at a time. The core can be configured to perform either a real correlation with a single data input stream and a single coefficient sequence, or a complex correlation with two input data streams representing the real and imaginary input terms, and two coefficient sequences representing the real and imaginary coefficients.

The core provides multiple channel capability and can support up to 256 channels. Correlations for each channel operate independently from each other. Also, up to 256 different coefficient sequences may be stored in the core, and each channel can select which coefficient sequence is correlated to that channel, so one coefficient sequence could be used for all 256 channels if desired.

Figure 1. Correlator IP Core External Interface Diagram



General Description

A correlation function determines how closely a data sequence matches a reference, or "coefficient" sequence. A high correlation value means that the data sequence closely matches the coefficient sequence. A low correlation value means that the data sequence is dissimilar to the coefficient sequence. The basic correlator equation is given by:

$$r_k = \sum_{i=0}^{corr\_win-1} d_{i+k} c_i \qquad k = 0,1,\ldots,num\_lags-1 \tag{1}$$

The terms of the equation are:

- **d_i** – Input data sequence. The Correlator IP core allows the input sequence to be from 1 to 8 bits wide, and either signed (two's complement) or unsigned data.
- **c_i** – Coefficient (or code) sequence. In the Correlator IP core, this sequence must be loaded into internal memory prior to a correlation operation. This sequence is always binary {1,0}; however, the coefficient sequence may represent either an unsigned sequence {1,0} or a signed sequence {+1,-1}. In the case of a signed coefficient sequence, a 1 in the sequence represents a value of +1 and a 0 in the sequence represents a value of -1. The d_i and c_i inputs must both be of the same type, either signed or unsigned.
- **r_k** – Correlation result output sequence (correlation between d_i and c_i inputs).
- **corr_win** – Correlation window. This is the number of elements in the input data sequence over which the correlation function is calculated. This is also referred to as the number of "taps." For the Correlator IP core this number is determined by the user when configuring the core. Once selected, the number of data elements (and coefficient terms) is then fixed at this number for all correlation operations. The number of terms in the coefficient sequence is always equal to the number of taps (corr_win) specified.
- **num_lags** – Total number of lags for which the correlation is performed. This is also the length of the correlation result sequence, r_k.

From Equation 1, a correlation operation takes an input data sequence d_i of length "corr_win" and multiplies each term in the sequence against the terms of the reference coefficient sequence c_i, summing the results of all of the multiplications to produce the result r_k. The input data sequence is then shifted by one element and the operation is repeated to produce the next term in the r_k sequence. This is done "num_lags" times.

In the Correlator IP core, each time a new data term is input to the core, one correlation operation is performed across "corr_win" data and coefficient terms, producing one result, r. The "k" index in Equation 1 does not apply since the Correlator always produces one new result when it receives one new data value. Old data beyond the defined correlation window size is not held in memory.

In addition to the basic correlation function described above, the Correlator IP core can be configured to perform complex correlations, defined by the equation:

$$r_k = \sum_{i=0}^{corr\_win-1} d_{i+k} \overline{c}_i \qquad k = 0,1,\ldots,num\_lags-1 \tag{2}$$

In this case, the data and coefficient input sequences are both complex and each contains a real input sequence and an imaginary input sequence. For the Correlator IP core, the input data sequence is a sequence of signed (two's complement) numbers from 1 to 8 bits wide, and the coefficient sequence is a binary sequence where a coefficient value of 1 represents +1 and a coefficient value of 0 represents a -1. Equation 2 represents the complex conjugate of c_i. The complex conjugate multiplication expressed in Equation 2 is given as:

$$d\overline{c} = (d_{re}\, c_{re} + d_{im}\, c_{im}) + j(d_{im}\, c_{re} - d_{re}\, c_{im}) \tag{3}$$

Since the coefficients in a complex correlation are restricted to the values {+1,-1}, the multiplications in Equation 3 simplify to inversions of the d_{re} and d_{im} terms, and the whole equation reduces to a series of additions and subtractions. The Correlator IP core performs these inversions and sums the results to produce a new result value r for each new d_{re} and d_{im} term input to the core. In this case, the result sequence will have two terms, a real term and an imaginary term.

Functional Description

The Correlator IP core is composed of the following functional blocks:

- **State Machine** – Controls the flow of data received from the user. Generates the starting pointer values necessary to read/write the Tap Memory and stores the pointer values in the Channel Memory. Stores the starting pointer values for the next correlation operation in the input FIFO. Generates the "shiftby" value for the aligner.
- **Channel Memory** – Stores the pointer to the location in Tap Memory to write the next data value.
- **Input FIFO** – Stores the pointer to the starting point in Tap and Coefficient Memories for the next correlation.
- **Tap Memory** – Stores the data terms (d_i).
- **Coefficient Memory** – Stores the coefficient terms (c_i).
- **Aligner** – Aligns data and coefficients read from memory for the correlation operation.
- **Correlator** – Performs the equivalent of the multiplication operations in Equations 1 and 2.
- **Adder/Accumulator** – Performs the addition and subtraction operations in Equations 1 and 2.

Correlator Input and Output Data

The Correlator IP core accepts a new input data value for a channel and writes that value into Tap Memory. When it is ready to perform the next correlation operation for that channel, the new data value will be included in the correlation, along with enough "old" data already in memory to completely fill the correlation window. The Tap Memory is a circular buffer which contains a correlation window's worth of data. As each new value is added to Tap Memory for a particular channel, a correlation operation needs to be run and completed before the next new value is added to memory for that channel. New data can be written into Tap Memory for channels other than the one the Correlator is operating on, however it is the user's responsibility to insure that a correlation operation is done for a particular channel before new data is written into memory for that same channel. This is easy to do for a large number of channels where new data is written to channels in a round-robin sequence, or if the Correlator throughput is not stressed to its limit (i.e. unused cycles appear between correlations), but the problem can be difficult to manage for small numbers of channels. The Correlator IP core will automatically prevent new data being written into Tap Memory and corrupting a correlation as long as the input FIFO depth (parameter 9) is set to 1.

Figures 2 and 3 show the timing of the user interface. The state machine accepts one new data value from the user interface at a time. When the state machine is ready to accept a new input data word from the user interface it asserts the `crdy` signal. The user interface then inputs `din`, `chan_in`, `code_sel_in`, `block_start_in`, and asserts the `irdy` signal. When the state machine sees `irdy` go active, it will take the new data value from the user interface. If the design has been configured for multiple channels, the state machine reads the pointer for that channel from the Channel Memory. This pointer value tells the state machine where in Tap Memory to write the data value just received from the user interface. This pointer value will also be the starting point for the next correlation operation, so this pointer value is stored in the Input FIFO until the next correlation operation is ready to start. Once the present correlation operation finishes, or if no correlation operation was in progress when a new data value was received, then the Input FIFO is read to determine the starting pointer for the next correlation. The state machine begins reading the Tap and Coefficient Memories at the starting pointer location, and it reads until it has read an entire correlation window's worth of data and coefficients.

Figure 2 shows at time 173.5µs that `crdy` went active. The user provided a value of 0x3 for channel 0, and set the `code_sel_in` to 1 which indicates which coefficient sequence is to be used for the correlation of channel 0 data.

In this example, two channels and two different coefficient sequences have been configured. Each channel can be correlated to either of the two coefficient sequences.

Figure 2. User Interface Timing Diagram for Two-channel Correlator

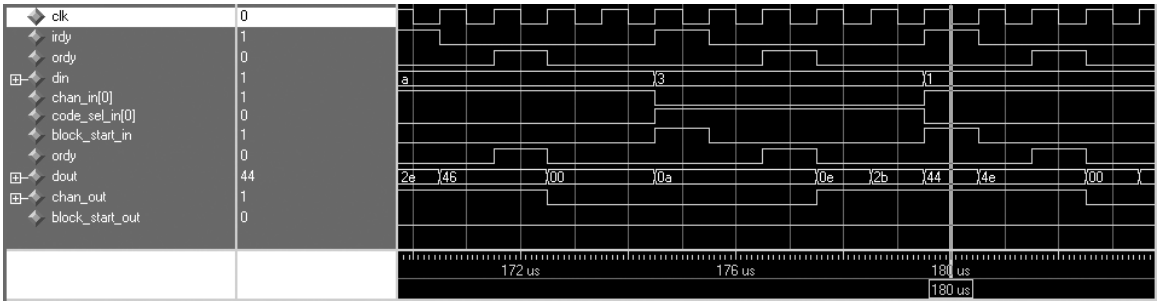
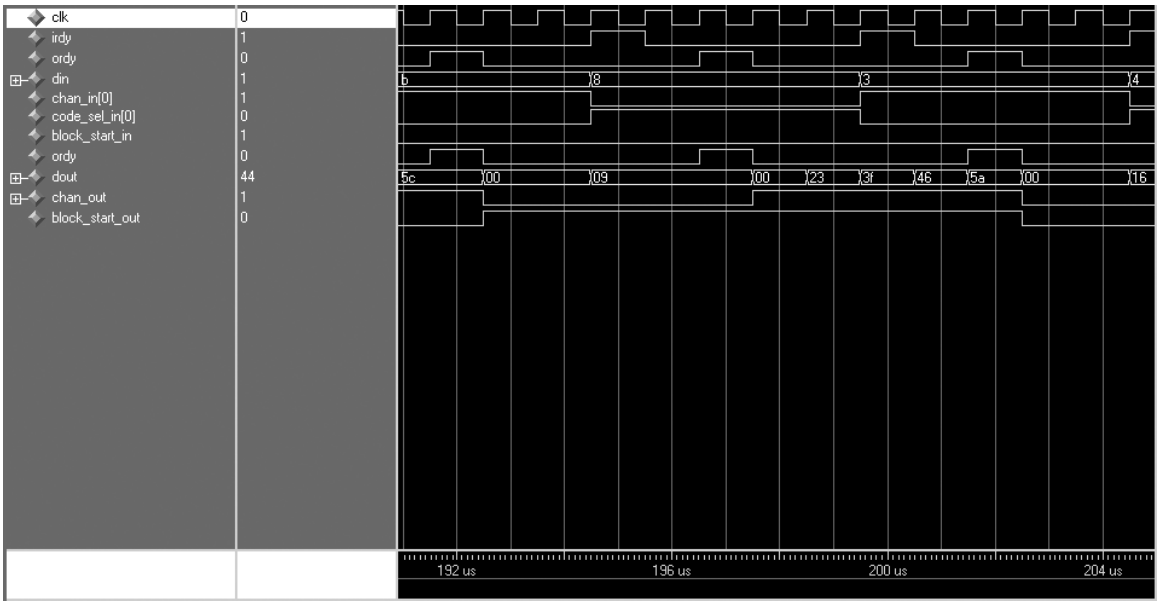


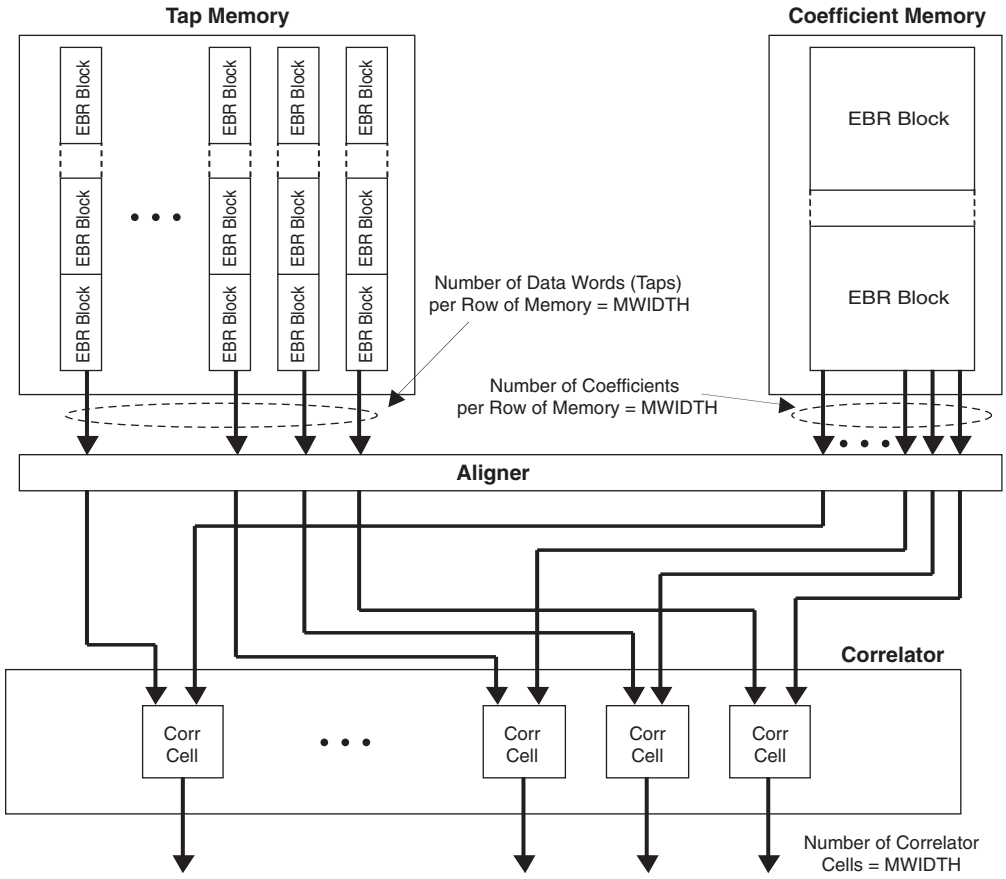
Figure 3. User Interface Timing Diagram for Two-channel Correlator



The user drives the `irdy` signal high for one clock cycle indicating that the input values are valid. Along with the input data, the user also sets the `block_start_in` signal. This signal will be taken as a marker by the Correlator and aligned with the input data as it passes through the Correlator IP core. The next time the `block_start_out` signal is set and `chan_out` = 0, it will indicate that the output data was associated with this input data value. The `block_start` signals act as markers for the user to do frame alignments between the input and outputs of the Correlator IP core. This is necessary since the core operates on one input data sample at a time. It does not perform multiple correlations over “num_lag” values as expressed in Equation 1. This allows the simplest and most versatile Correlator IP core design. If it is necessary to operate for multiple “lags,” then the user application will need to add input and output FIFOs around the core to feed data values one sample at time.

At time 178.5µs, `crdy` again goes active indicating that the Correlator IP core is ready to accept the next input value, and in the example of Figure 2 the user inputs data for channel 1. At time 196.5µs, the correlation result for channel 0 is ready at the `dout` outputs, and the core outputs a value of 0x9 on `dout`, sets the `chan_out` to 0, and asserts the `ordy` signal. It also asserts the `block_start_out` signal to indicate that this output value was associated with the `din` value from time 175.5µs.

Figure 4. Tap and Coefficient Memories



Tap and Coefficient Memories

While the Tap and Coefficient Memories are being read, the values read are passed to the Aligner. Under the control of the state machine the Aligner shifts the tap data and coefficients to be passed to the Correlator block. The state machine also generates strobe signals to the Aligner which indicate, in any given clock cycle, which tap and coefficient values are valid for the correlator block to work on.

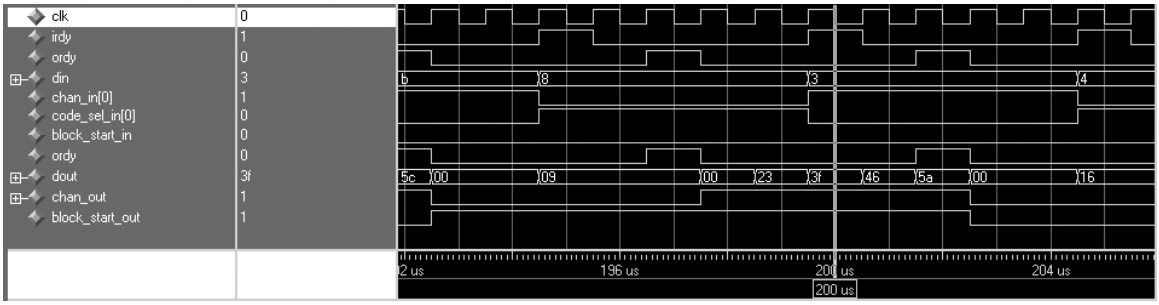
The Tap and Coefficient Memories are implemented with EBR blocks as shown in Figure 4. The Correlator IP core will automatically configure and instantiate the proper number of EBR blocks in the design based on the parameters selected by the user. In the case of the Tap Memory, the number of correlator cells, number of taps, number of channels, and the oversampling rate all determine how many EBR memories are needed. The number of correlator cells (parameter MWIDTH) determines how many words of data can be operated on during a single clock cycle. The more correlator cells which are configured, the more multiplication operations can occur in a clock cycle and the overall data throughput goes up. At least one EBR memory is required to feed each correlator cell. All Tap Memory EBR blocks in the design will be configured to be at least the word width of the input data (DWIDTH) wide. The EBR blocks can be sized 1, 2, 4, or 9 bits wide, and must be equal to or greater than DWIDTH. Since each EBR block can store 8192 bits, if the value of [TAP_EBR_WIDTH * (NUM_TAP / MWIDTH) * NUM_CHAN * OS_FACTOR] exceeds 8192 bits, then multiple EBR blocks will be stacked in columns to feed the correlator cells, as shown in Figure 4. TAP_EBR_WIDTH is the minimum allowed EBR width which is at least DWIDTH wide. The Tap Memory EBRs will be configured automatically for the user; however, the user is responsible for determining the total number of EBR blocks needed for the design and insure that the target LatticeEC™ device contains enough memories.

The Coefficient Memories are also implemented in EBR blocks. Since each coefficient is constrained to be 1 bit, the total amount of memory required for coefficients is generally less than that required for tap data. Each EBR

block can be a maximum of 36 bits wide, therefore if the number of correlator cells (MWIDTH) is not greater than 36, only one column of EBR memories is required for the Coefficient Memories. If MWIDTH > 36, then multiple columns will be configured. As in the case of the Tap Memories, if the total number of coefficients which needs to be stored exceeds one row of EBR memories, then multiple rows will be configured in a stacked arrangement as shown in Figure 4. For MWIDTH ≤ 36, the number of coefficients required is [NUM_TAP * NUM_COEF_SEQ]. If this number is less than 8192 then only one EBR is needed for the Coefficient Memory.

Unlike the Tap Memories which are written with new user data under the control of the state machine, the Coefficient Memories must be written with the coefficient sequences before any correlation operations can be done. This is done via the Coefficient Memory Configuration interface shown in Figure 5. This interface consists of the input signals: `coeffaddr`, `coeffwdat`, `coeffwdat_im`, and `coeffwr`. Figure 6 shows the timing for this interface for a two-channel design with MWIDTH=4, NUM_TAP=16 and NUM_COEF_SEQ=2.

Figure 5. Coefficient Memory Configuration Interface Timing

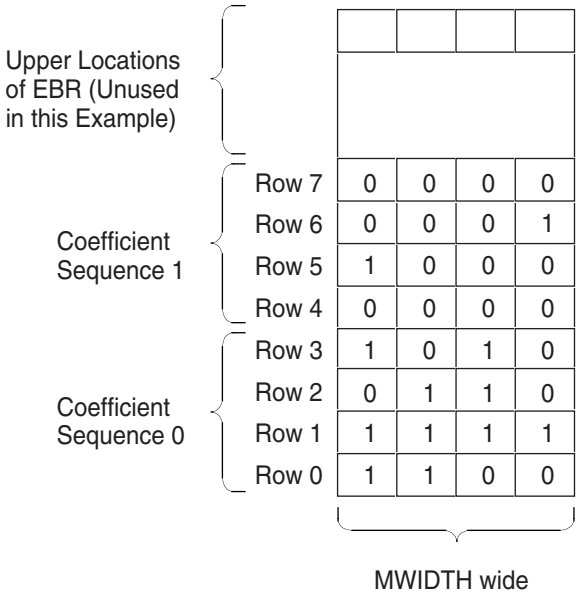


In this case, the Coefficient Memory is implemented in one EBR block. Each row of Coefficient Memory is required to store MWIDTH=4 coefficients, so each write to the memory writes four bits. Each coefficient sequence is NUM_TAP=16 bits long, and it will occupy (NUM_TAP / MWIDTH)= 4 rows in the Coefficient Memory. In addition, in this example there are two separate coefficient sequences, so the coefficients will occupy a total of eight rows in the Coefficient Memory.

Writes to the Coefficient Memory are enabled by asserting the `coeffwr` input. The `coeffaddr` input selects the row of memory to be written, and `coeffwdat` (and `coeffwdat_im` for complex correlations) is set to the desired value. This is a very simple interface, however it is essential to make sure that the coefficient sequence is written in the correct order. In the example above, the first four values written are for coefficient sequence 0. The values written are 0xa6fc (or in binary: 1010 0110 1111 1100) with the LSB being the first bit in the correlation sequence. This bit will be multiplied against the newest data value received by the Correlator. The MSB in this string will be multiplied against the oldest data read from Tap Memory. This is explained further in the Correlator Evaluation Package section of this document.

The second coefficient sequence written into the Coefficient Memory is 0x0180, and is written into rows 7, 6, 5 and 4. This will be selected as coefficient sequence 1 by setting the `code_sel_in` to 1 when a data value is input to the Correlator. Figure 6 shows how the coefficient values from this example would appear in the Coefficient Memory:

Figure 6. Example Coefficient Sequences Written in Coefficient Memory



In configurations where the number of taps is not a power of 2, the coefficient sequences will need to be padded with zeros so that all coefficient sequences written to the Coefficient Memory are a power of 2 long. This is because the Coefficient and Tap Memories must be divided up evenly into sections representing the individual channels and coefficient sequences. In the example above, if the number of taps were less than 16, the coefficients would still be written into memory the same way except that padding zeros would be added before the MSB. The padding zeros would be written into memory starting at the MSB of row 3 for coefficient sequence 0.

Correlator and Adder/Accumulator Blocks

The Correlator block performs the multiplication operations in Equations 1 and 2. The coefficients are configured by the user to be either unsigned or signed. If unsigned, then the binary coefficient values simply represent {1,0} and the multiplications reduce to either passing the tap values read from memory to the Adder/Accumulator, or passing a zero value. If the coefficients are signed, then the binary coefficients {1,0} represent values of {+1,-1}. If a tap value is multiplied by 1, then the Correlator block does nothing other than pass the tap value read from memory to the Adder/Accumulator. If a tap value is multiplied by -1, then the Correlator block does a two's complement conversion of the tap value read from memory and passes the result to the Adder/Accumulator, which in turn completes the summation of the correlation sequence to generate the final result.

Decimation

The Correlator IP core allows the input data to be oversampled from two to eight times the normal sampling rate. The OS_FACTOR parameter should be set to the correct oversampling rate. When this is done, the core will automatically decimate the amount of data which is included in the correlation operations by the correct amount. For example, if the number of taps is eight and an oversampling rate of two is chosen, then the circuit will correlate the eight coefficient values with the newest input tap data value and the odd numbered tap data values from the past 15 "old" data values. The correlation will look like this:

$$r = d1c1 + d3c2 + d5c3 + d7c4 + d9c5 + d11c6 + d13c7 + d15c8 \quad (4)$$

The number of data values stored in Tap Memory for a given channel becomes [OS_FACTOR*NUM_TAP], or in this case 16. The number of coefficients per channel is still equal to NUM_TAP.

Parameter Descriptions

The parameters used for configuring the Correlator IP core are listed below. The values of these parameters must be set prior to synthesis or functional verification.

Table 1. User Configurable Parameters

Parameter Number	Parameter	Parameter Description	Input Range	Default Input Value	Parameter Values
1	DWIDTH	Input data width	1-8	4	—
2	NUM_TAP	Number of taps	8-2048	16	—
3	MWIDTH	Number of correlator cells	Minimum = 1 Maximum = the number of EBR blocks in the target LatticeEC device	4	—
4	NUM_CHAN	Number of channels	1-256	2	—
5	DTYPE	Input data type	Signed, unsigned	Unsigned	“UNSIGNED” “SIGNED”
6	COMPLEX	Correlation type	Real, complex	Real	Real = 0 Complex = 1
7	OS_FACTOR	Oversampling rate	1-8	1	—
8	PERFORMANCE	Performance	1, 2, 3	1	—
9	FIFO_DEPTH	Input FIFO depth	1, 2, 3	1	—
10	NUM_COEF_SEQ	Number of coefficient sequences	1-256	NUM_CHAN	—

The basic configuration parameters should be selected based on the type of correlation desired. These include parameters 1, 2, 4, 5, 6, 7 and 10. The remaining parameters 3, 8 and 9 are selected based on the desired performance of the circuit.

For parameter 3, a higher f_{MAX} can be achieved by generating a much smaller circuit (smaller number of correlator cells). However, for long data sequences (number of taps, or “corr_win”), this will mean that many clock cycles are needed for each correlation result to be calculated resulting in very poor overall data throughput and long latency times. For higher data throughput, and at the expense of a larger and therefore more complicated circuit, a higher number of correlators should be chosen. The Correlator IP core is architected to be highly pipelined, so even for large numbers of correlator cells, the penalty in f_{MAX} is small; however, as the design becomes more complicated, it will eventually reach a point where the f_{MAX} is impacted by routing in the FPGA.

Parameter 8 should be set to 1 for the highest performance circuit. A value of 2 or 3 will result in a smaller, but significantly lower performance design.

Parameter 9 sets the depth of the input FIFO. This improves throughput performance by allowing the next input data sample to be presented to the device while the present correlation result is being calculated. However, care must be used when changing this parameter. If the FIFO depth is set above 1, then the user must insure that a new data sample will not be presented to the Correlator IP core for the same channel as is presently being serviced or the new data sample will be written into the core’s internal tap memory and will corrupt the correlation which is already in progress for that channel. If the core has been configured for multiple channels, and input data values for the same channel are never presented to the core adjacent to each other in time, then the FIFO depth can be safely increased beyond 1. For example, if the core is configured for eight channels, and data for each of the eight channels is always presented in sequence, then the FIFO depth may be increased to 2 or 3. However, if the core is configured for one or two channels, or the input data sequences through channels at random, then the FIFO depth should never be increased beyond 1.

Custom Core Configurations

For core configurations that are not available in the Evaluation Packages, please contact your Lattice sales representative to request a custom configuration.

Related Information

For more information regarding core usage and design verification, refer to the *Parallel RapidIO Physical Layer Interface IP Core User's Guide*, available on the Lattice web site at www.latticesemi.com.

Appendix for LatticeECP™ and LatticeEC™ Devices

Table 2. Performance and Resource Utilization[1]

Parameter Filename	Parameter Settings	SLICEs	LUTs	Registers	External Pins	sysMEM™ EBRs	f_{MAX}
Corr_8bit_e2_1_001.lpc	See Table 3	212	140	310	41	5	212.54 MHz

1. Performance and utilization characteristics are generated using LFEC20E-5F672C in Lattice's ispLEVER® 4.2 software. When using this IP core in a different density, speed, or grade within the Lattice ECP/EC family, performance may vary.

Supplied Netlist Configurations

The Ordering Part Number (OPN) for the Correlator IP Core on LatticeECP/EC devices is CORR-8BIT-E2-N1 (for all configurations of the netlist package). Table 3 lists the evaluation netlists that can be downloaded from the Lattice web site at www.latticesemi.com. To load the preset parameters for this core, click on the “Load Parameters” button inside the IP Manager tool. Make sure that you are looking for a file inside this core's directory location. The Lattice Parameter Configuration files (.lpc) are located inside this directory.

Table 3. Parameter Settings of the Evaluation Packages

Input Data Width	# of Taps	# of Correlators	# of Channels	Input Data Type	Correlation Type	Over-sampling Rate	Performance	Input FIFO Depth	# of Coefficient Sequences
4	16	4	2	Unsigned	Real	1	1	1	2

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Lattice:

CORR-8BIT-E2-UT2 CORR-8BIT-PM-UT2 CORR-8BIT-XM-UT2