XAPP212 (v1.1) January 10, 2001

CDMA Matched Filter Implementation in Virtex Devices

Author: Ken Chapman, Paul Hardy, Andy Miller, and Maria George

Summary

Code Division Multiple Access (CDMA) is a rapidly expanding data transmission technique in the emerging Universal Mobile Telecommunications System (UMTS). This application note describes the implementation of a CDMA matched filter using the architectural features of the Virtex™ series, Virtex-II series, and Spartan™-II devices.

Introduction

CDMA is a technique used for wireless data transmission. Every mobile handset and every wireless base station operate on the same frequency spectrum. In order to discriminate one conversation from another, every handset broadcasts a unique code sequence. CDMA base stations must be able to discriminate these different code sequences in order to distinguish one transmission from another. This discrimination is accomplished by means of a matched filter, whose output indicates when a particular code sequence is detected in the input data stream. This application note includes a reference design (HDL code) for implementing parallel matched filters.

Matched Filters

A matched filter is a filter whose frequency response is designed to exactly match the frequency spectrum of the input signal.

The operation of matched filters are the same as correlating a signal with a copy of itself. These filters are used as signal processors in communications receivers to calculate the correlation between the transmitted signal and the received signal.

In CDMA systems the matched filter is tuned to match a code sequence which is expected to be contained within the digital samples entering the system receiver. The matched filter indicates when this code sequence is detected in the input data stream. The output of a matched filter will be a score value. A higher score value indicates a more tuned match with the code sequence of the received data stream. This is also called correlation, and hence a high score value represents a good correlation of input with the code sequence of interest.

Unlike a transmission filter where the continuous data stream entering the filter is modified to form a new continuous data stream, a matched filter output must be considered to be a flow of individual results. These results must then be analyzed to identify the individual point where the match occurred. Figure 1 shows the basic operation of a matched filter.

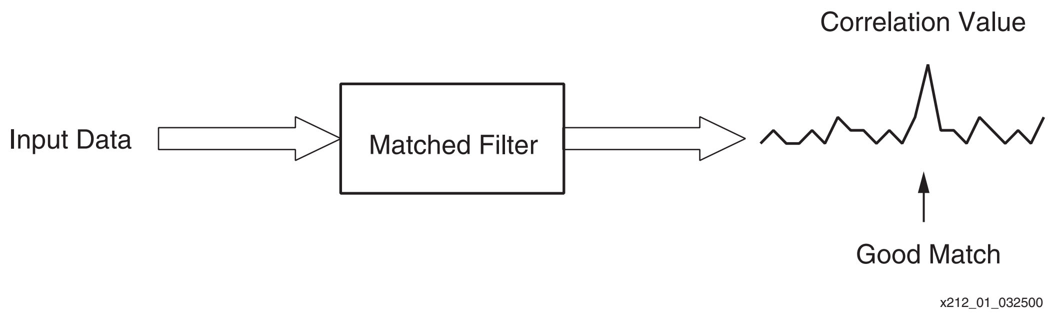

Figure 1: **CDMA Matched Filter Basic Operation**

Code Sequence

In a purely digital environment the data being transmitted is a sequence of ±1 values. Although any pattern of values may occur, all transitions will occur at a regular time known as the chip rate. One chip is therefore the minimum period of time spent at a given value.

Within the pattern of data there will be various code sequences of interest. In UMTS matched filter is used to detect the initial synchronization sequence of 256 chip periods, common to all users. Further stages know when to look for more unique user information.

In Figure 2, a more modest code sequence of 16 chips is considered for clarity. It is important to note that the transmitted values are ±1. In general a logic "0" represents +1 and a logic "1" represents −1.

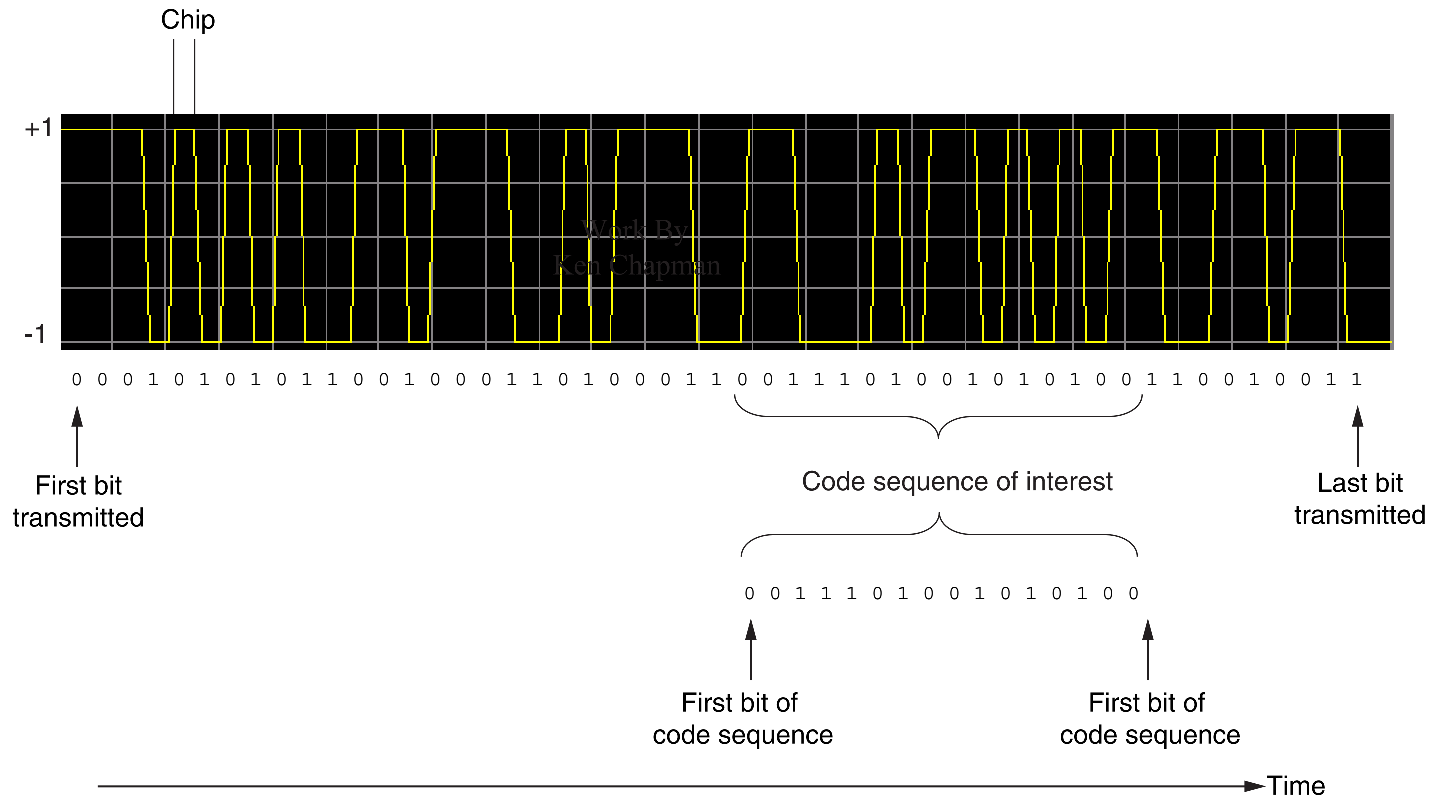

Figure 2: **Code Sequence Detection**

Manual Matching

In order to match the code sequence of interest using manual techniques, it is easiest to imagine sliding the input data stream past the desired code sequence until a perfect match is achieved.

With a code sequence of 16 chips, the ideal match will occur if every bit of the sequence corresponds to the data pattern. As shown in Figure 3 the perfect score is 16.

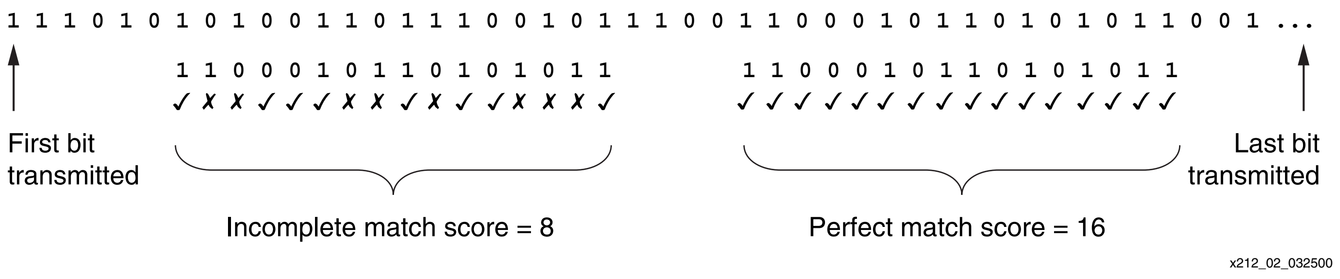

Figure 3: **Manual Matching**

Every result must be considered separately and does not reflect what might happen when a comparison is conducted in the next position. Consider the score on either side of the perfect match shown in Figure 4. The score of five is less than the earlier incomplete match, and does not predict an imminent perfect match.

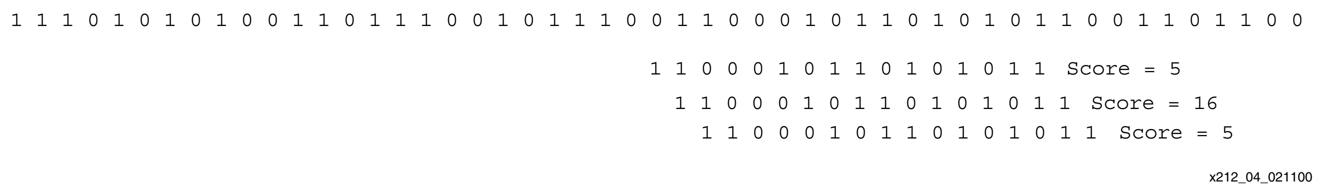
1 1 1 0 1 0 1 0 1 0 0 1 1 0 1 1 1 0 0 1 0 1 1 1 0 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 0 0
1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 1 Score = 5
1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 1 Score = 16
1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 1 Score = 5
x212_04_021100

Figure 4: **Matched Scores**

Simple Matched Filter

A matched filter can be realized with the same structure as FIR (Finite Impulse Response) filter with +1 and –1 coefficients.

The structure (Figure 5) is similar to the manual method, the code sequence of interest is tested at each bit position along the data pattern. One difference is the code sequence is held statically within the multipliers of the filter and the data pattern is passed through the tap registers of the filter.

The importance of the values in the data pattern, and in the code sequence, being ±1 are also realized. The effect of $-1 \times -1 = +1$ and the effect of $+1 \times +1 = +1$, shows both "bit match" conditions contribute to a larger positive result. Any bit failing to match contributes -1 to the final result and helps emphasize that the code sequence does not match.

It is vital that the code sequence is searched for in the correct order. Figure 6 considers the "1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 1" code sequence to correctly represent the ±1 format. It is important to identify the first bit of the sequence.

A perfect match result occurs when the first bit of the code sequence is compared with the first bit of the same sequence in the data pattern. In a FIR filter structure this comparison does not occur until the entire code sequence data pattern is held within the filter taps. The FIR filter structure in Figure 5 shows the correct code sequence. The perfect match is identified once all the bits of the code sequence are received. Each result corresponds to the previous 16 bits received.

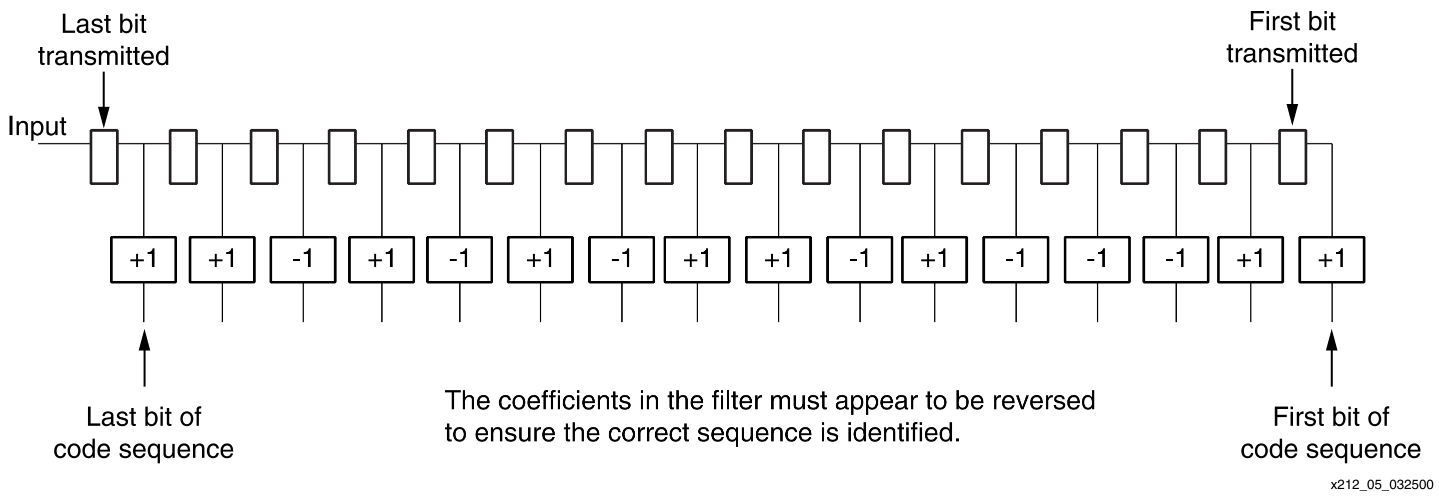
Last bit transmitted
First bit transmitted
Input
+1 +1 -1 +1 -1 +1 -1 +1 +1 -1 +1 -1 -1 -1 +1 +1
Last bit of code sequence
The coefficients in the filter must appear to be reversed to ensure the correct sequence is identified.
First bit of code sequence
x212_05_032500

Figure 5: **Matched Filter Using FIR Structure**

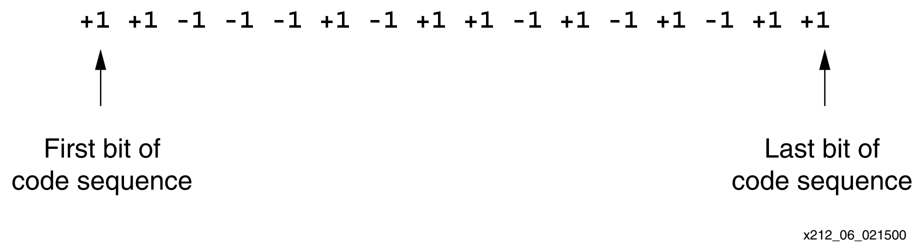

Figure 6: **Correct Sequence Order**

Input Sampling

The example shown in Figure 3 assumes the availability of a perfect digital data stream. If this data stream were viewed in a logic simulator, clean, sharp transitions would be observed with minimum high and low data periods as determined by the basic clock rate of the simulation. In UMTS, the minimum high and low periods are determined by what is termed a "chip rate", currently specified at 3.84 MHz, though this frequency is adjustable to 2× and 4× this rate in the future.

There is a problem with the transmission of very square transitions. Very rapid transitions result in high instantaneous frequencies, and these high frequencies fall outside the bandwidth allowed for UMTS. Figure 7 shows a perfect representation of a pulse shaped PN code sequence. The pulse shaping is used to limit the bandwidth of the pure digital signal. However, the pulse shaping requires interpretation of the analog representation of +1 and –1.

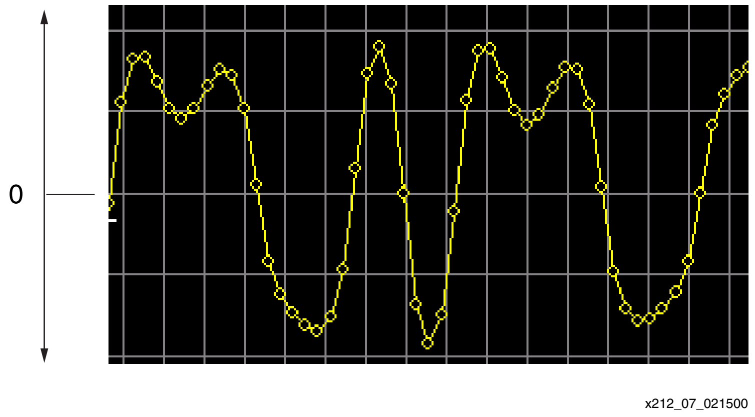

Figure 7: **UMTS Transmitted Waveform**

The circles shown in Figure 7 represent the points where the analog signal is sampled. In Figure 7, the waveform is over-sampled at four times the chip rate. The over-sampling of the input signal is an attempt to locate the mid-chip point. If this point can be determined for one chip period, given a constant, known chip frequency, then the midpoint of the next chip will be one chip period later.

The input data pattern is represented by digital words with values greater than ±1. The tap shift registers have a width of several bits (typically 4 to 16 bits). The multipliers are still performing a coefficient multiplication by +1 or –1, but on a complete data word with a corresponding output word. Figure 8 illustrates the summation of the tap results as being the result of adding larger values to equal a large variation in output correlation values.

If the over-sample rate is 4× the chip rate, then one in every four samples will be closest to the "mid-chip" point. These samples will yield the strongest correlation result.

The matched filter observes all the input samples, but only performs the correlation process with the samples separated by the chip period.

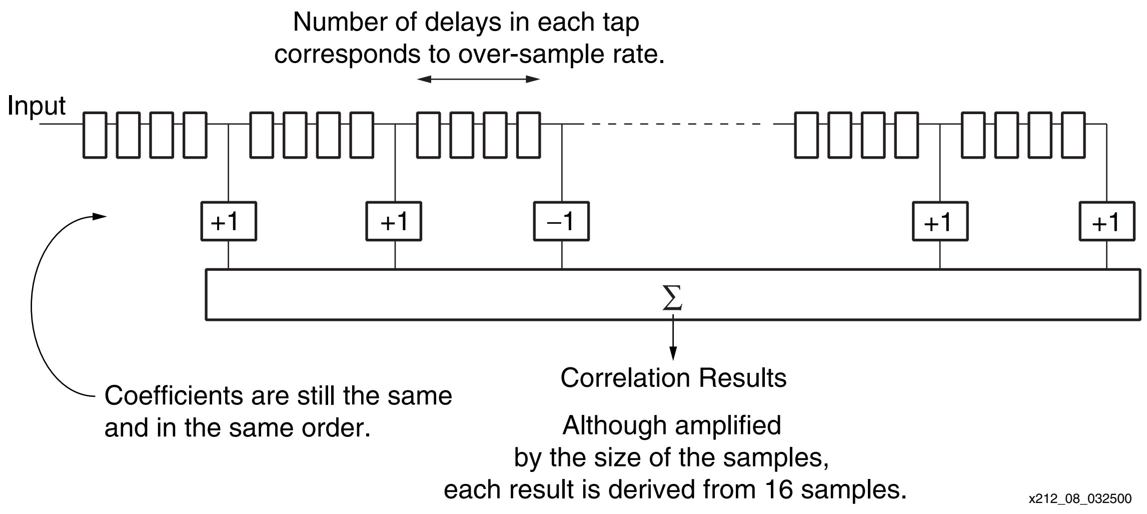
Number of delays in each tap
corresponds to over-sample rate.
Input
+1
+1
–1
+1
+1
Σ
Correlation Results
Coefficients are still the same
and in the same order.
Although amplified
by the size of the samples,
each result is derived from 16 samples.
x212_08_032500

Figure 8: **Matched Filters with a 4× Over-sample Rate.**

In practice, the higher the sampling rate, the greater the chance of recovering the mid-chip point. However, higher sampling rates result in higher processing rates, and a presumed requirement of more FPGA resources. Consider the following equation:

# of Flip-Flops consumed = (# of bits per sample) × (over-sample rate) × (code sequence bit length)

Thus, assuming 8-bit data samples, Figure 8 will require:

(8 bits per sample) × (4× over-sampling) × (256-bit code sequence length)

= 8192 FFs = 4096 Virtex, Virtex-E, Virtex-EM or Virtex-II slices

4096 slices consume 85% of an XCV400 device or 80% of an XC2V1000. From the above equation, it would seem impractical to move to 8× or 16× over-sampling, based on an area consumption argument. Another factor to consider is that with a UMTS data rate of 3.84 MHz, 4× over-sampling requires a clock frequency of 15.36 MHz, which is extremely slow for even the slowest Virtex device. The Virtex high frequency capability can be used to increase the over-sampling rate to 16×. From the equation above, this would appear to require 24,576 additional flip-flops to the area requirement and thus prove impossible. The ability to overcome this apparent impossibility is the strength of the implementation shown in Figure 12. This application note explains the reduction in flip-flop utilization. The analysis for reducing consumption of resource is the focus of the following section.

Transposed Form FIR Filters Structural Advantages

Due to the adder tree structure, the implementation of the parallel matched filter with the traditional FIR filter structure requires a total of 6,776 slices (XCV600 or XC2V1500). The solution to a difficult adder tree structure is found in the transposed form FIR filter format (Figure 9). The input data sample passes at the same time to all tap multipliers. The effect of tap delay is restored by delays in an addition chain. The order of the coefficients is reversed.

All the tap registers can be removed if the input can deliver signals to all the tap multipliers without loading problems. The number of adders is essentially the same. The delays associated with each adder do not take additional resources in the Virtex device. The transposed form FIR filter flow of data aids performance and enables a parameterizable number of filter taps. Although the total number of adder bits is larger, the performance is increased and the ease of design is improved.

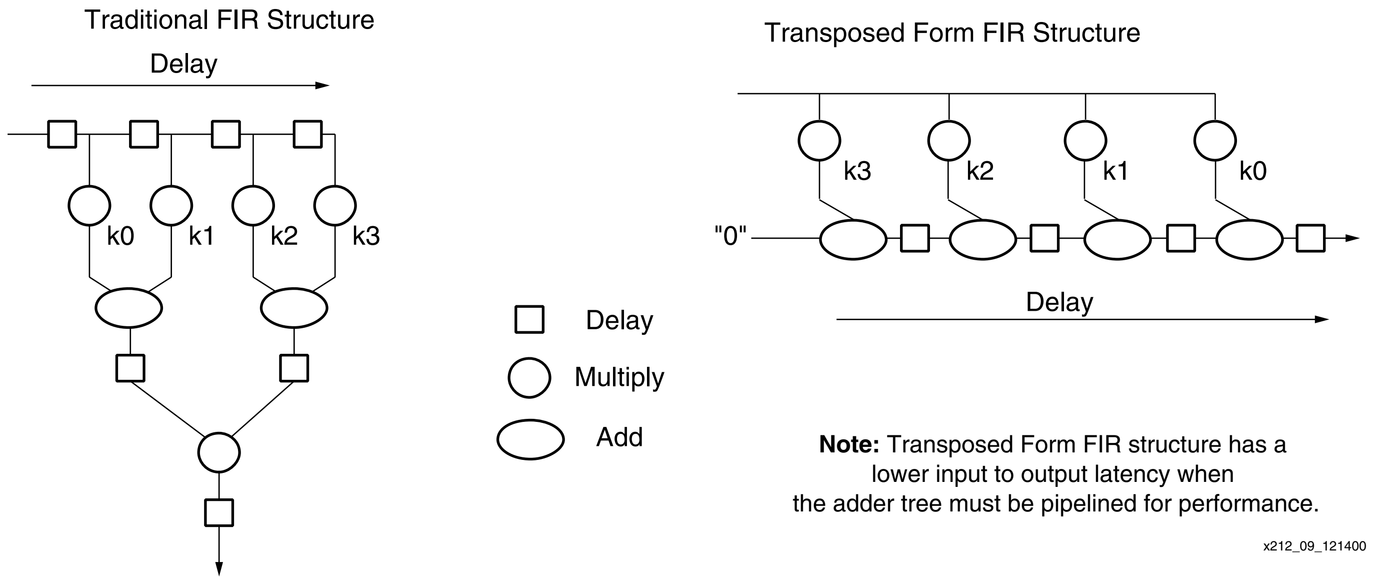
Traditional FIR Structure
Delay
k0
k1
k2
k3
Delay
Multiply
Add
Transposed Form FIR Structure
k3
k2
k1
k0
"0"
Delay
Note: Transposed Form FIR structure has a lower input to output latency when the adder tree must be pipelined for performance.
x212_09_121400

Figure 9: **Comparison of Traditional and Transposed Form FIR Filter Structures**

The transposed form FIR filter structure is applied to the matched filter with over-sample data input (Figure 10).The reorganization of the building blocks requires the code sequence coefficients to be reordered.

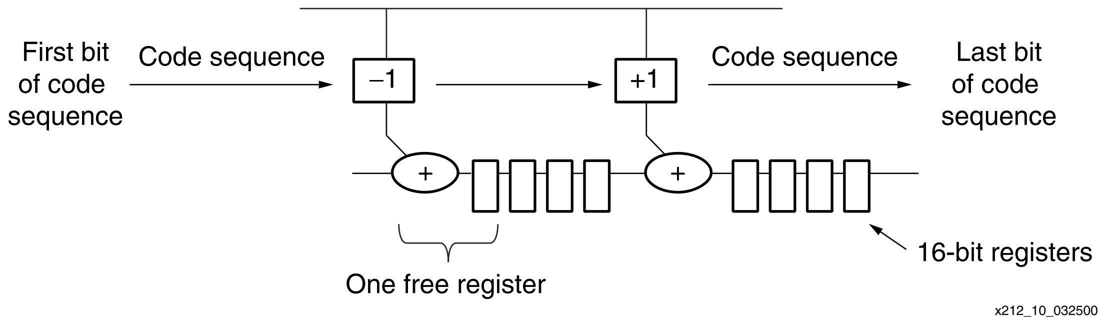

Figure 10: **Transposed Form FIR Filter Structure With Over-sampling**

In the traditional FIR structure, each of the multipliers takes the input value and multiplies it by either +1 or -1 which produces a value passed to an adder. For example, (A × [±1]) + (B × [±1]) could be expressed as (±A±B). However, it is not possible to implement an add/subtract on both input ports efficiently with (+A±B) being the basic operation. Only half of the multipliers are removed in the traditional FIR method. In the transposed form FIR structure, each adder in the chain is used to add the tap multiplier result to the total. Therefore, one input is added and the multiplier is absorbed into the adder chain. For example, Total + (B × [±1]) could be expressed as (+Total±B).

Each tap is reduced by the size of a multiplier and the utilization of the 256 chip matched filter will be = 256 × (8 slice for 16-bit adder with register + 24 slices for 48 flip-flops) = 8,192 slices, (20% bigger than the traditional FIR).

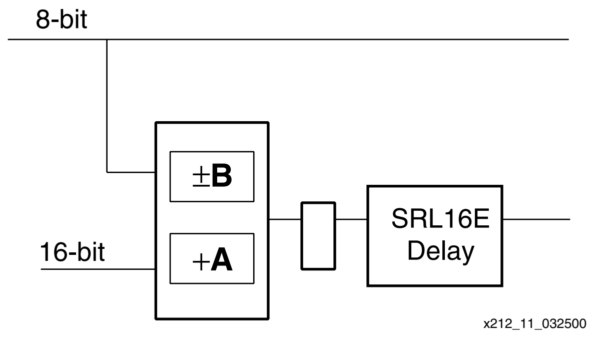

Figure 11: **Transposed Form FIR Filter Tap With an SRL16E**

Virtex devices provide a shift register macro to convert any LUT into a shift register of up to 16 stages. This macro is accessed through the SRL16E primitive.

The SRL16E is readily absorbed into the implementation of a matched filter, in both the traditional and transposed form formats (Figure 11). However, due to the wider data path in the transposed form format, the saving is greater.

A 256 chip example with 8-bit data and 4× over-sample:
256 × (8-slice adder with register + 8 slices for delay)
= 4,096 Virtex, Virtex-E, Virtex-EM or Virtex-II slices.

Although the traditional FIR consumes 3,704 Virtex slices, the transposed form FIR is easier to design and implement.

Parallel Matched Filter

The structure and implementation style of the parallel matched filter with 4096 Virtex slices is shown in Figure 12 (85% of a XCV400 device or 80% of an XC2V1000).

The code sequence must be applied in the correct order. The code might be encoded into the logic by inserting a tap module containing an add or a tap module containing a subtract. See **Multiple Channels with Different Code Sequences**, page 9.

In the transposed form FIR filter structure, the only effect of 4× over-sample rate is the SRL16 based delay blocks. In Case 1 of Figure 12, each SRL16 inserts three delays in addition to the register associated with the add/sub unit. The input sample rate is assumed to be 4 × 3.84 MHz = 15.36 MHz. The structure operates at a slow clock rate of 15.36 MHz. This UMTS specified frequency is especially low for Virtex devices with capabilities of operation in the 100 MHz range.

In Case 2, increasing the sample rate to 61.44 MHz makes 16× over-sampling possible. Normally this would add 24,576 flip-flops to the transposed form FIR matched filter structure, however, no additional flip-flops are required because of the available SRL16 delays. Without the SRL16, this design would not fit in a XCV400 or an XC2V1000, in fact, it would require four XCV400 devices (or five XC2V1000 devices).

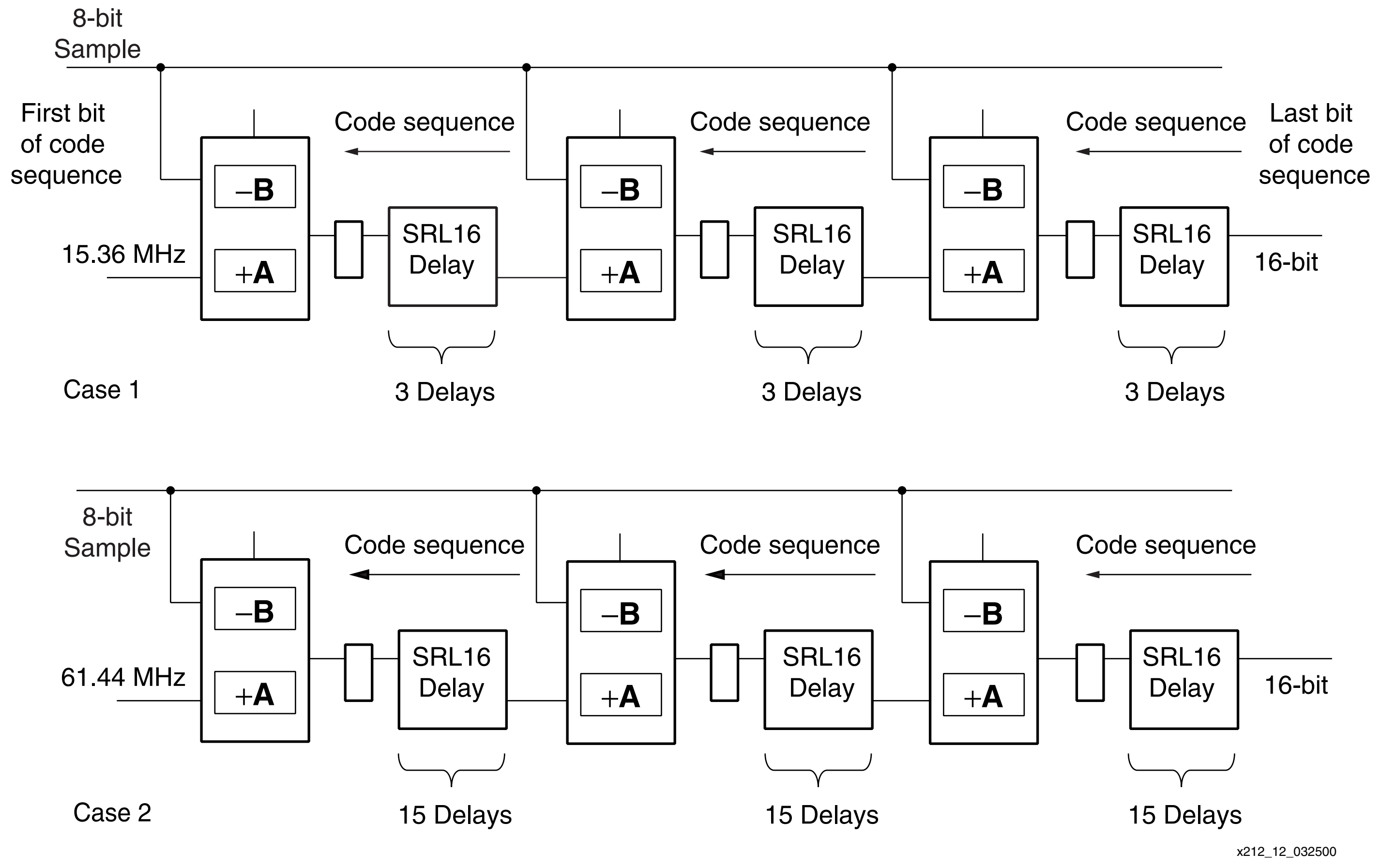

Figure 12: **Parallel Matched Filter with 4× and 16× Oversampling**

Advantages of SRL16s in Parallel Matched Filters

Designing For Enhancements

Options exist for the UMTS standard to include 8.192 MHz and 15.36 MHz "chip" rates. Performance can be reserved for "programmable base stations" of the future. By using SRL16s in parallel matched filters higher data rates can be achieved without additional resources.

Multiple Channels with the Same Code Sequence

By time-multiplexing data streams, multiple channels can be interlaced to form a 16× data flow into the matched filter as either eight channels of 2× , four channels of 4× , or two channels of 8x. It is important to ensure that samples of associated channels and sample spacing are tested correctly. This is ensured by the matching of SRL16 delays to channels and an over-sample rate for correlation with the sequence. This is ideal for any of the fixed codes.

Multiple Channels with Different Code Sequences

Another use of the SRL16 is the application of different code sequences in different clock cycles of the same time sharing period. Using sets of flip-flops and a selection multiplexer can again become expensive. (Eight slices for 16 code bits and a 16:1 multiplexer).

In Figure 13 four different codes are applied to the add/sub control pin. The four bits will be stored interlaced in an SRL16E by serial loading with clock enable (CE) High and setting the address (A0 to A3) to "3" (4-bit delay). During filter operation, CE is held Low to prevent shifting of the code sequence. The individual bits are accessed by adjusting the address value. This scheme supports up to 16 bits in the same Virtex series look-up table (LUT).

Besides testing input data with different code patterns, the combination of different codes and interlaced channels are used to construct complex matched filters. Constructing these complex matched filters is beyond the scope of this application note. Please contact **support.xilinx.com** for more information.

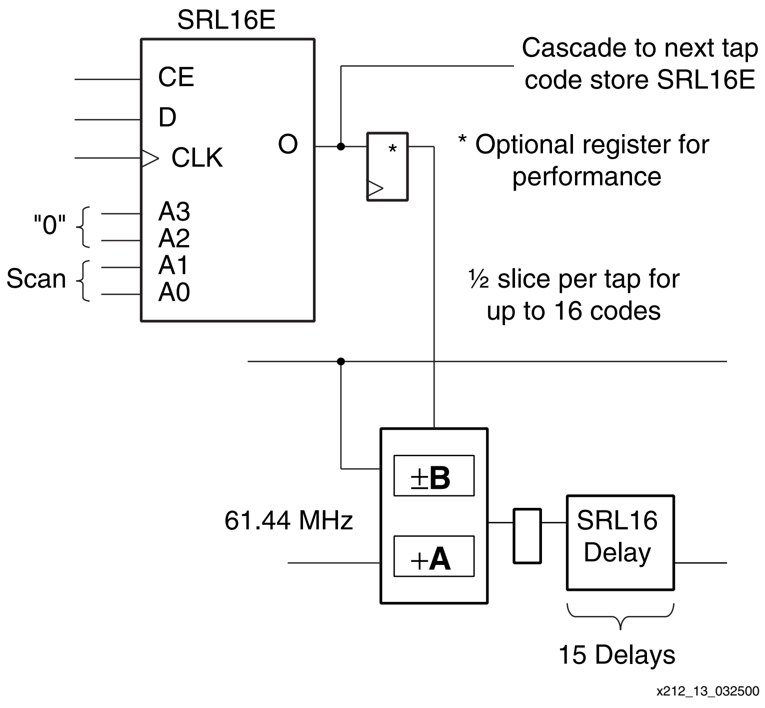
SRL16E
CE
D
CLK
O
A3
A2
A1
A0
"0"
Scan
Cascade to next tap code store SRL16E
*
* Optional register for performance
½ slice per tap for up to 16 codes
±B
+A
61.44 MHz
SRL16 Delay
15 Delays
x212_13_032500

Figure 13: **Different Codes**

Filter Folding

In Figure 14, the over-sample rate is a 4× chip rate (3.84 MHz) with a code sequence length of 16 chips. The design is scalable for other sample rates and sequence lengths. The filter is folded to achieve the functionality of 16-tap units with just four tap units.

As an example, with a 256-bit code sequence, 8-bit samples, and a 4× chip rate:

(256/4) × [8 slices for add/sub + 9 slices for SRL delay + 1 slice for code bits)] + 30 slices for control circuitry

= 1,118 Virtex slices (23% of an XCV400 or 22% of an XC2V1000).

This is the lowest utilization for a single filter with these specifications.

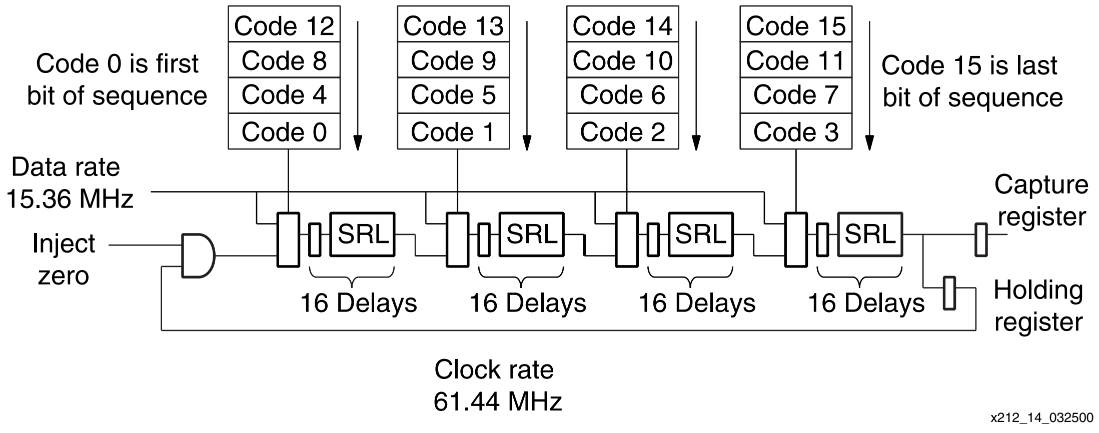

Figure 14: **Filter Folding**

Folded Filter Operation

With the folded filter approach, just four taps (instead of 16 taps) are required for a 16-bit code. Each data sample is applied for four clock cycles (clock rate is 61.44 MHz and data rate is 15.36 MHz). At the first clock cycle, Code 0, Code 1, Code 2, and Code 3 are applied at the four taps as shown in Figure 14. Also at the first clock cycle, a zero is injected into the Code 0 adder. At the end of the first clock cycle, the result of the addition is shifted into the holding register. In the second clock cycle, the same data sample is compared with Code 4, Code 5, Code 6, and Code 7 and the input to the Code 4 adder comes from the holding register. The final result of the addition in the second cycle is shifted into the holding register. In the third clock cycle, the same data sample is compared with Code 8, Code 9, Code 10, and Code 11 and the input to the Code 8 adder comes from the holding register. The final result of the addition in the third cycle is shifted into the holding register. In the fourth cycle, the same data sample is compared with Code 12, Code 13, Code 14, and Code 15 and the input to the Code 12 adder comes from the holding register. The final result of the addition in the fourth cycle is shifted into the capture register. Similarly in the next four clock cycles, a new data sample is compared and a final result is shifted into the capture register.

Code Loading for Folded Filters

The code must be stored in a folded form. Distributed dual port RAM is useful for loading the code. Figure 15 shows the transposed form FIR structure in a right to left flow.

To load a new sequence the write address is first held at value "0000" and the code is applied serially to the Code_in port with write enable (WE) High. Each RAM storage element behaves like a single flip-flop and the cascaded RAM blocks function as a shift register. After filling the cascaded RAM blocks (i.e, 64 clock cycles in a 256 chip filter folded 4x), the write address increments to "0001". After another 64 clock cycles, the write address increments to "0010". Eventually WE is removed and all the bits are stored. The scan address selects each group of code bits to apply to the add/sub module by using the dual-port access to each RAM.

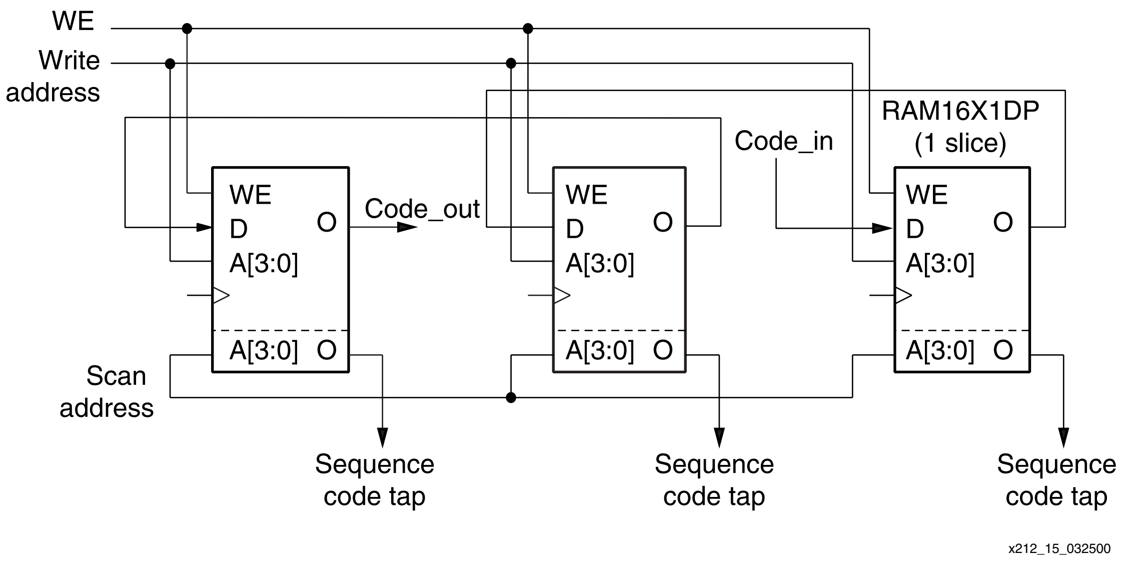

Figure 15: **Code Loading for Folded Filters**

HDL Code

The VHDL code for the parallel matched filter is provided as a reference design. The data width and the oversample rate for the parallel matched filter design is parameterizable. This design uses a scalable structure for each tap in the filter to be identical. Therefore, ensuring that the basic tap of the filter is properly implemented is important. Filters of any size can be easily built in this VHDL code using a FOR...GENERATE loop.

The basic architecture of the filter tap consists of an adder/subtactor followed by a delay line implemented using the SRL16 macro in Virtex series devices. A single adder/subtactor module is automatically inferred by the synthesis tool with a simple IF...THEN... ELSE statement. For example:

```
IF (1CodeData = ’0’) THEN
sADDSubOut <= sPrevTap + sInData;
ELSE
sAddSubOut <= sPrevTap - sInData;
END IF;
```

Here, 1CodeData is the input code being matched, determining if addition or subtraction should occur.

The delay line can be inferred by describing a generic pipeline. An asynchronous reset condition is not used because the SRL16 cells do not have this capability. However, this does not affect a pipelined design such as a filter. The following example is code to infer an SRL16 macro:

```
IF (SysClk ’EVENT AND SysClk = ’1’) THEN
sPipe <= sPipe(sPipe’high-1 DOWNTO sPipe’low) & NewData;
END IF;
```

The VHDL code for the implementation of a CDMA matched filter is available on the Xilinx FTP site at: ftp://ftp.xilinx.com/pub/applications/xapp/xapp212.zip or xapp212.tar.gz.

Conclusion

All devices in the Virtex and Virtex-II series of devices can implement highly efficient, high-performance CDMA matched filters, providing robust performance for emerging UMTS applications.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
3/31/00	1.0	Initial release.
01/10/01	1.1	Updated with Virtex-II series information.