Implementing WiMAX OFDM Timing and Frequency Offset Estimation in Lattice FPGAs

A Lattice Semiconductor White Paper

November 2005

Lattice Semiconductor
5555 Northeast Moore Ct.
Hillsboro, Oregon 97124 USA
Telephone: (503) 268-8000
www.latticesemi.com

Introduction

Orthogonal Frequency Division Multiplexing (OFDM) is the basis of some WLAN and WiMax air interfaces. It also has been proposed for use in next generation cellular systems such as Super-3G and HSOPA (High Speed OFDM Packet Access) in the 3GPP standards group. OFDM has a number of advantageous features, such as good tolerance to multi-path fading and inter-symbol interference (ISI). By using a number of sub-carriers, the symbol length can be kept long and a guard period (the cyclic prefix) used to mitigate ISI. Data is allocated to a number of sub-carriers so that any nulls in the frequency domain do not knock out a whole allocation. This gives forward error correction (FEC) a greater chance to recover the data in the receiver.

These advantages do come at a cost. First, the orthogonal modulated carriers need to be generated. Fortunately, the Inverse Fast Fourier Transform (IFFT) algorithm can be used to convert suitably constructed frequency domain signals into the required time domain waveform. Equally, the receiver can use the FFT algorithm to convert back to the frequency domain before de-modulation. These algorithms are efficient and can be implemented in either software in a DSP or, for higher bandwidth and increased processing capacity, in a DSP-enabled FPGA, such as the LatticeECP device. Another cost associated with OFDM is the use of FFTs and the fact that the receiver has no prior knowledge of either the symbol timing or exact frequency of the local oscillator at the transmitter end.

Any time offset between the start of the orthogonal waveforms and the first sample point used in the FFT will affect the result of a Fourier Transform. Also, any frequency offset between transmitter and receiver, if not corrected, will result in a blurring of the

information between the frequency bins. This is called inter-carrier interference (ICI), and is caused by a loss in the orthogonality of the carriers. In the worst case, in which the offset is of the order of the sub-carrier spacing, the information will mostly land in the adjacent bin, causing all the bits on that carrier to be lost.

A number of approaches to estimation timing and frequency offset in OFDM systems have been presented in the literature. Many of these operate in the time domain (before the FFT) and use the repeating pattern of the preamble or the cycle prefix, or both, to gain information about the symbol timing and frequency offset. The timing is determined by noticing that the correlation of the signal with a delayed version of itself will reach a peak when the repeated pattern is located. The frequency offset can be estimated by, for example, calculating the phase offset between one occurrence of a pattern and the next. Moose [1] presented a simple method using the cycle prefix, as did Van de Beek et al [2]. This approach is useful in systems in which a continuous stream of OFDM symbols are transmitted (e.g. DTB-T). The algorithm proposed by Schmidl and Cox [3] uses the repetition in the preamble, which proves more robust compared to methods that use the cycle prefix when this is short. Other, more complex, methods are described in references [4] and [5].

The remainder of this paper shows the data flow for two selected algorithms and how they can be implemented by a single flexible architecture that uses the DSP resources found in LatticeECP devices. The IEEE 802.16-2004 standard is used to illustrate the principles described, but they are applicable to a number of OFDM-based wireless systems. The architecture has been used in the implementation of an OFDM PHY reference design base on WiMAX requirement. Details of this can be found in references [6] and [7].

Data Flow

The data flow for the two algorithms is discussed in this section. The Van de Beek and Schmild and Cox algorithms were chosen for this study because they occupy the middle of the complexity range covered by the surveyed work. The first was described using the cyclic prefix of the OFDM symbol, but it can be adapted to use the burst preamble. The WiMAX OFDM preamble is defined differently for the uplink and the downlink. The specification also defines variants for doing initial ranging on the downlink. In both cases, the time domain signal has a repeated pattern. The long preamble, used for downlink ranging, consists of two symbols: a 4x64 pattern symbol, where a 64-sample pattern is repeated 4 times, and a 2x128 pattern symbol with two repetitions of a 128-sample pattern. The uplink uses a short preamble with just a 2x128 pattern symbol. These preamble symbols all have the usual cyclic prefix attached.

Figure 1 shows the data flow for the Van de Beek algorithm configured for the 4x64 preamble. The preamble pattern also is shown for reference. The diagram includes a modification of the algorithm to exploit the 64-sample repeating pattern, so that frequency offsets of greater than half a sub-carrier spacing can be handled. The labels in brackets (i.e. x1, x2, ms1, ms2, ms3) are included to help relate this diagram to the hardware mapping shown later in this paper.

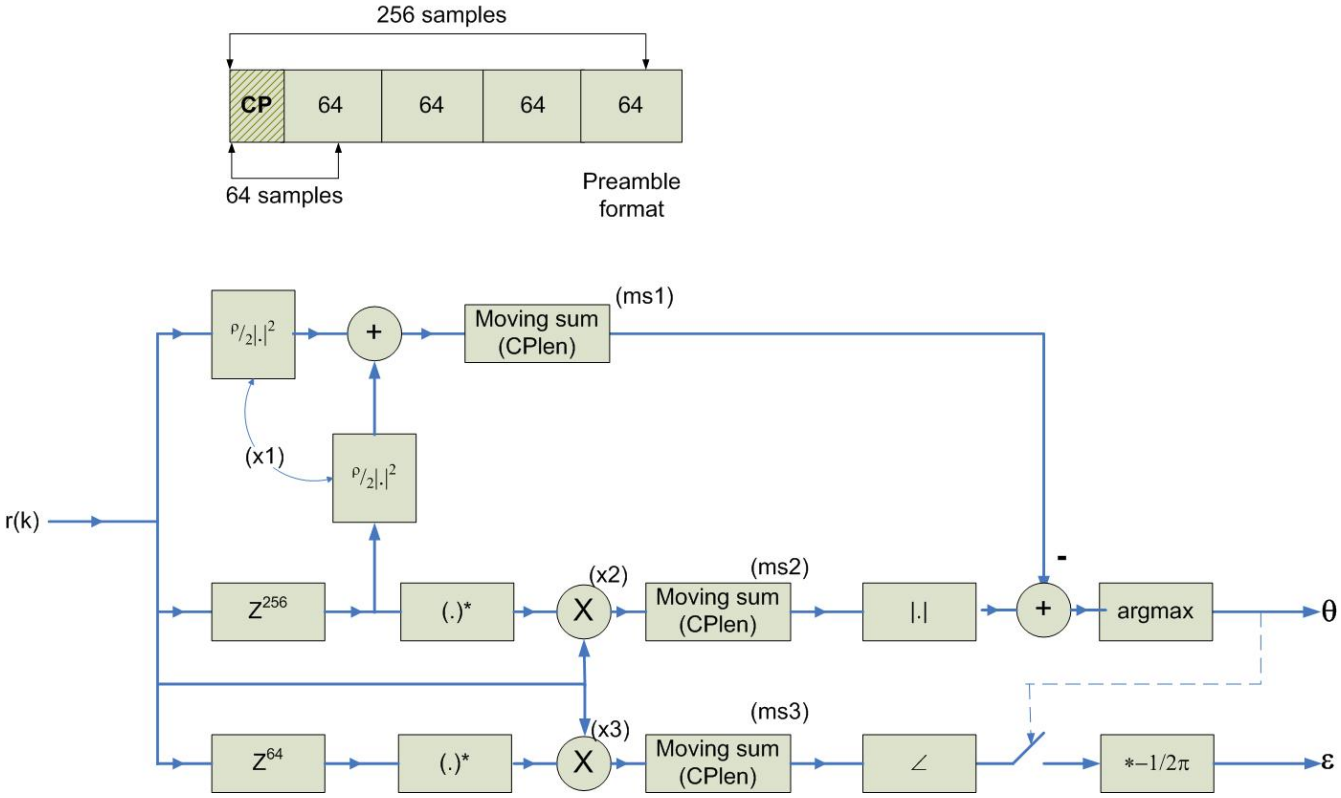

Figure 1 - Data Flow Diagram of Modified Van de Beek Algorithm

This algorithm data flow consists of two branches, one calculating an energy term and the other a correlation term. In the modified algorithm, the correlation term is calculated between data samples separated by the length of a symbol, exploiting the cyclic prefix. A second correlation is performed between data that is one repetition of the preamble pattern apart. This second value is used to calculate the frequency offset, ε. Equation 1 shows the calculation of the energy part where N is the symbol length of 256 samples and L is the length of the cyclic prefix (shown as Cplen in Figure 1). In IEEE802.16-2004 the cyclic prefix can be 64, 32, 16 or 8 samples long. Equation 2 shows the calculation of the correlation part. Here, both N and L are equal to the

length of the repeated pattern in the preamble: that is, 64 samples. The factor $\rho/2$ should depend on the signal to noise ratio (SNR), but it was found by simulation that it was safe to set this to one.

$$ms23 \equiv \sum_{k=m}^{m+L-1} r^*(k)r(k+N) \qquad (1)$$

$$ms1 \equiv \frac{\rho}{2} \sum_{k=m}^{m+L-1} \left|r(k)\right|^2 + \left|r(k+N)\right|^2 \qquad (2)$$

The moving sum ms2 and ms3 only differ in the value of N so equation (1) describes them both as ms23. The correlation with delay 64 is used to calculate an angle that is used only when the magnitude of the difference between ms1 and ms2 reaches a maximum (argmax). This operation ensures that the frequency offset calculation is done at the best time, i.e., when the correlation over the actual received symbol cyclic prefix is complete.

Figure 2 shows the data flow for the Schmidl and Cox algorithm. This appears to be slightly less complex because it requires one less moving sum. However, an additional multiplication is required in this implementation to avoid taking a square root. The pipelining of the data out signal has been shown for completeness in this diagram. This extra delay is used to make sure the data output is correctly aligned for the next stage of processing.

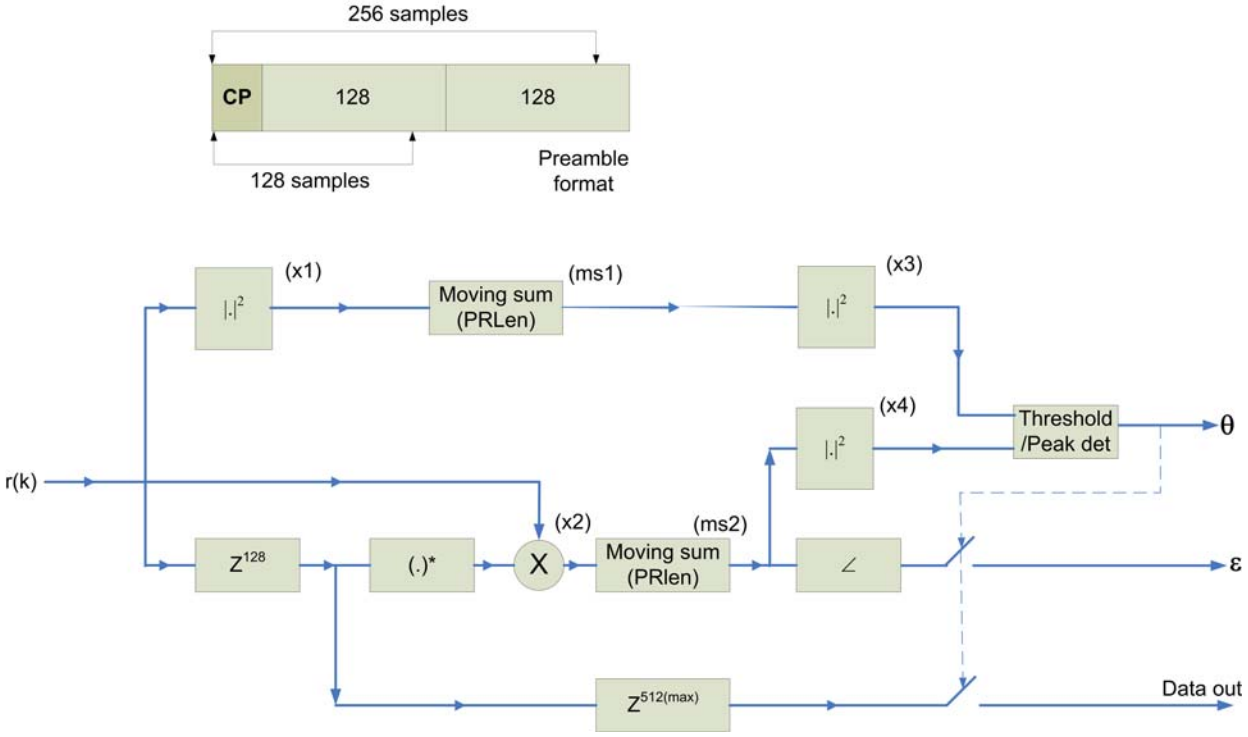

Figure 2 - Data Flow for Schmidl and Cox Algorithm with 2x128 Preamble

The Schmidl and Cox paper [3] derives the algorithm using the follow equation:

$$M(k) = \frac{\left|P(k)\right|^2}{(R(k))^2} \qquad (3)$$

where:

$$P(k) = \sum_{m=0}^{L-1} (r_{k+m}^{*} r_{k+m+L}) \qquad (4)$$

and:

$$R(k) = \sum_{m=0}^{L-1} \left| r_{k+m+L} \right|^2 \qquad (5)$$

In these equations, L is equal to the length of the preamble repeated pattern, i.e., 128 on the uplink. The R(k)[1] term is the energy term and P(k) the correlation term and M(k) is a metric used to identify the location of the start of a burst. The frequency offset is given by using θ = angle(P(d)) and $\Delta f = \theta/(\pi T)$ where T is L*Ts and Ts is the sample period. In Figure 2 ms1 is equal to R(k) and ms2 is equal to P(k) and the threshold/peak detect block replaces the division in equation 3. The frequency offset estimation, ε, is taken when the correlation term reaches a peak as long as it is greater than one eighth of the energy term. This peak also identifies the start of the burst and is used to set the data output offset pointer.

Implementation

The LatticeECP low cost FPGA family features memory blocks (EBRs), fast carry adders and dedicated sysDSP blocks. The latter, as shown in Figure 3, contain a combination of multipliers, adder/subtractor/accumulators, as well as a summation unit. These are capable of carrying out a multiply-add/sub operation at up to 250Mhz. This valuable resource can be shared across a number of operations in a given algorithm if the data rate is a fraction of the target clock speed. Further, multiple channels of data can be processed if this fraction is small enough.

[1] The Van de Beek paper used d; we use k for consistency.

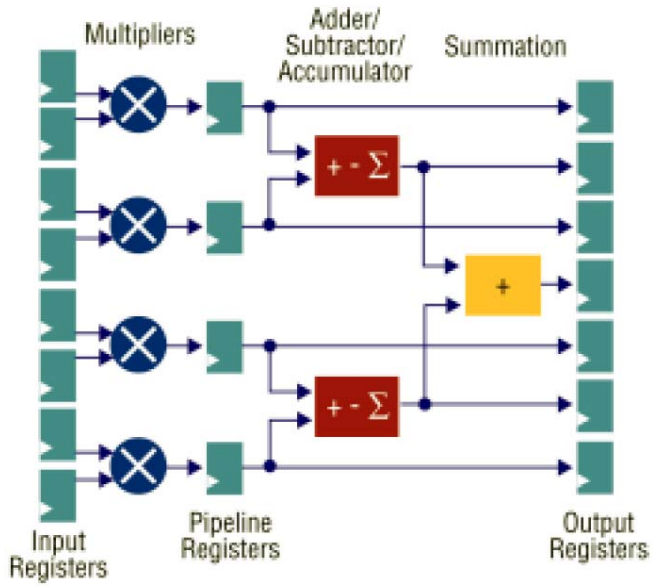

Figure 3 - Lattice DSP Block Structure

The sysDSP blocks can be configured to operate on 9, 18 or 36-bit inputs. Figure 3 shows the resources available in 18-bit mode. In 9-bit mode there are twice as many multipliers available, so it is attractive to attempt to limit the bit-widths to get more processing capacity.

Figure 4 shows a block diagram of an architecture that is capable of implementing either algorithm using 2 multiply-add/sub blocks (i.e., one half of a DSP block when using 9x9 multipliers) and one or two EBRs. To do this, the circuit needs to run at least 5 times the sample frequency (i.e. about 55Mhz for a 10Mhz bandwidth 256 point OFDM signal). The additional EBR is needed when using a preamble of 2x128 samples. Multiplexing is kept to a minimum with the inputs to the DSP block selected by the control signals called multimode and x*y. The multimode control determines if a magnitude squared ($|\bullet|^2$) or complex multiply (X) is being done. Note that two

magnitude-squared operations can be done in one cycle. The x*y control signal multiplexes a zero onto one operand so a real multiply can be done using one of the mult-add/sub units.

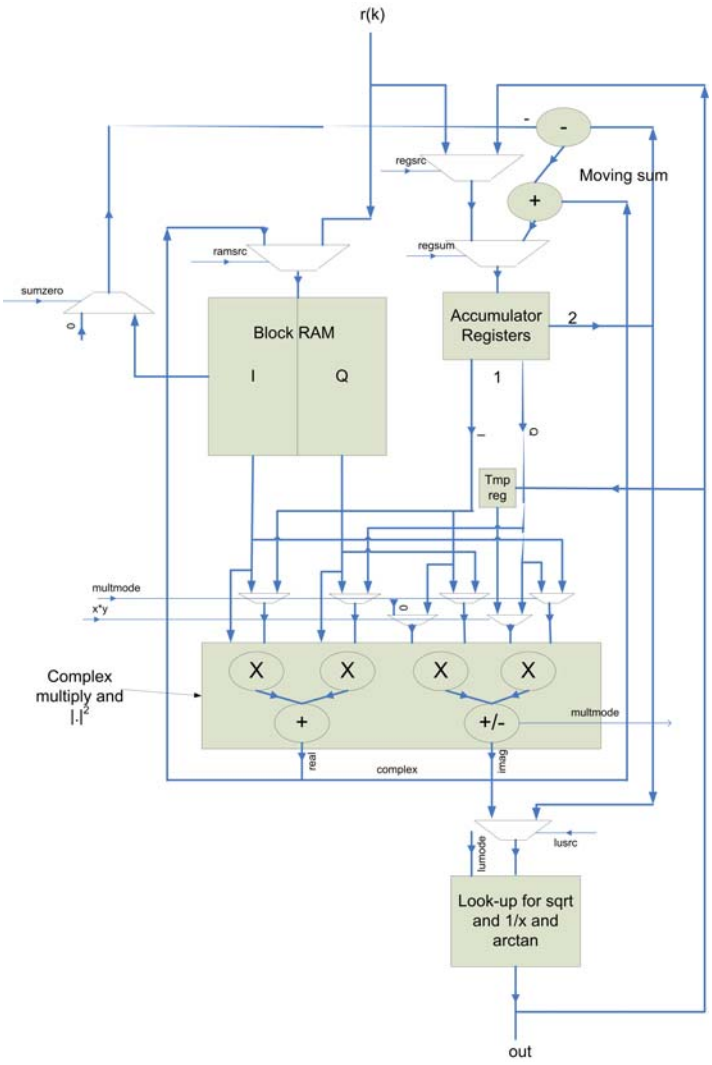

Figure 4 - Block Diagram of FPGA Hardware Capable of Implementing Either Algorithm

Data Storage

Delaying the input data by 256 and 64 samples only consumes 256 locations, as the delay of 64 can be taken as a tap off the larger delay. Each moving sum requires a number of locations equal to the length of the cyclic prefix in the Van de Beek case, and the preamble repeat length for Schmild and Cox, so 128 locations per moving sum are needed in the worst case. Assuming the input data, r(k), is complex 2x8 bits wide and the required output precision for frequency offset is 8 bits, then it can be determined that at least 11 bits are needed in the moving sum calculation. One EBR will store 512 words of 18 bits or 1024 words of 9 bits. The complex data will require 16 bits. Because 256+64+64+64 is less than 512, 1 EBR will be sufficient.

Each moving sum requires a register. One option is to use distributed RAM to implement this storage, as it will reduce the number of multiplexers required. An alternative is to use an N-bit wide shift register with output taps as required to supply data in the appropriate processing cycles. The second option was chosen in the OFDM reference design.

Mapping the van de Beek Algorithm to the architecture

Table 1 shows the allocation of operations to clock cycles, taking into account the data dependencies identified in the data flow. The operations carried out by the DSP block are magnitude (x^2+y^2), complex multiply (X) and real multiply (x*y). This mapping does not take into account any pipelining that may be necessary to achieve the target processing speed. Given that it takes 5 cycles to complete the required calculations, the circuit will need to run at least five times the sample rate. For a 10Mhz nominal channel in IEEE 802.16-2004, the target speed will be about 60Mhz. The accumulator

register has 2 ports (write and read) and holds the moving sums. A temporary register (tmp reg) holds intermediate results for multi-cycle calculations. The EBR needs to be dual port to allow the moving sum data to be updated, while also writing and reading the input data to implement the delays. The memories required by the algorithm, shown as Z256, Z64, ms1, ms2 and ms3 in Figure 1, are implemented as one EBR. The complex multiply and the DSP block in mult-add/sub mode implements magnitude. The angle operation '<' (arctan(y/x)) is broken down and a look-up table used for 1/x, a real multiply to give y/x and a look-up table for arctan. The look-up table can be implemented in a single shared ROM. The sumzero signal is asserted in the initial cycles to zero the memory contents and registers.

Table 1: Sequencing to Map the van de Beek Algorithm to Architecture

Cycle	Operation	EBR wr	EBR read1	EBR read2	Reg rd	Reg wr	tmp reg	look-up mode
1	x^2+y^2 x^2+y^2	Msum 1	Z256	Msum1	Msum1	Msum 1	-	-
2	X	Msum 2	Z256	Msum2	Msum2	Msum 2	-	-
4	x^2+y^2	-	-	-	Msum2	-	-	sqrt
3	X	Msum 3	Z64	Msum3	Msum3	Msum 3	wr	1/x
5	x*y	r(k)	-	-	-	-	rd	atan

Implementing the "Argmax" Function

The architecture in Figure 4 implements the algorithm separately from the "argmax" function. An implementation of this is described here. The receiver does not know when the transmitted signal will arrive, and it is assumed that the input to the circuit will be noise before the preamble is seen. The correlation between data at a distance of the symbol length averaged over CPlen points will tend to be zero for noise, but the energy will be non-zero. This means that the output of the subtractor will be negative until the start of the preamble (or any symbol). Taking advantage of this, the "argmax" circuit can defer searching for a maximum until its input crosses zero and goes positive. This point is about one half of the CPlen or less distance away from the peak. The search window can start at the zero crossing point and continue for CPlen/2 plus an error margin to account for noise-induced error in the zero-crossing point. In the absolute worst case, the search window size would equal CPlen, which is 64. The consequence is that memory is needed to store 64 results from the arctan(y/x) calculation. This will take 64 locations in an EBR, which is small compared to the other storage.

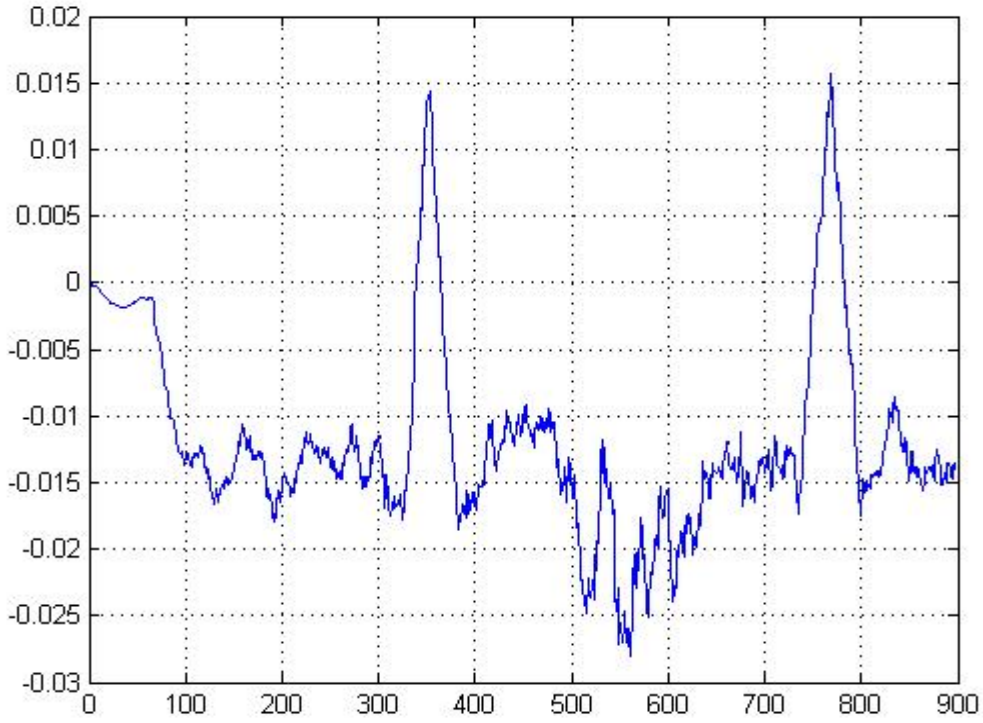

Figure 5 - Matlab Plot of the Signal Feeding Argmax

Mapping the Schmidl and Cox Algorithm

Table 2 shows how the Schmidl and Cox algorithm can be mapped to the hardware architecture as shown in Figure 4. The table can be compared to Table 1 and it can be seen that this algorithm can be mapped in to one less clock cycle.

Table 2: Sequencing to map Schmidl and Cox algorithm to architecture

Cycle	Operation	EBR wr	EBR rd	Reg rd	Reg wr	tmp Reg	look-up mode

1	x^2+y^2	Msum1	Msum1	Msum1	Msum1	rd	atan
2	x^2+y^2 x^2+y^2	Msum2	Msum2	Msum2	Msum2	-	-
3	X	r(k)	Z128	-	-	-	sqrt
4	x*y	-	-	-	-	wr	1/x

Table 3 shows how operations in Table 2 need to be scheduled to allow for pipelining in the multipliers. Each mult-add/sub unit is shown as ALU_r and ALU_I, denoting their outputs in cases in which they are performing a complex multiply operation. The DSP blocks support latencies from zero to three clock cycles, depending on which pipeline registers are enabled in the configuration. Each row in the Table shows the input, operation and output of a given resource. The first two columns indicate the resource and what is it doing in each cycle. Each subsequent column represents a clock cycle. This algorithm only requires 4 clock cycles for implementation. The Table includes extra pipeline balancing delays (p1, del, P1, P2, P3) and shows how the overall latency means that the calculation takes 4 sample periods, assuming the hardware is clocked at four times the sample rate. For example, the operation |•|1 in cycle 2, the first magnitude squared operation, produces a result in cycle 5. This is shown as |•|1' in clock cycle 1 of the next computation cycle. The computation cycle takes 4 clock cycles, so a throughput meeting the sampling rate can be achieved with a clock running at 4 times the sample rate. Values obtained from operations on previous results from the previous computation cycle have another tick added. So ALU_I out in cycle 5 is |•|4' and it is seen as |•|4'' in cycle 1.

Table 3 - Scheduling of the Schmidl and Cox Algorithm

Resource	port	1	2	3	4	5	6	7

ALU_r	in	-	r(k) r(k)	r(k) Z^{128}	-			
	op	-	|•|1	X	-			
	out	|•|1’	X’	-	-	|•|1	X	
ALU_i	in	Ms1’ Ms1’	Ms2’ Ms2’	r(k) Z^{128}	1/x’’			
	op	|•|3	|•|4	X	x*y’’			
	out	|•|4’		x/y’’’	|•|3	|•|4	X	x/y’’
p1	out	|•|3’				|•|3		
del	out	|•|3’’						
Mem	rdaddr	Z^{128}	dout	Ms1	Ms2			
	wraddr	Ms1’	Ms2’	r(k)	-			
	rddata	Ms1’	Ms2’	Z^{128}	dout			
	wrdata	Ms1’	Ms2’					
Reg	out	Ms1’’	Ms2’’	Ms1’	Ms2’			
P1	out		Ms1’	Ms2’				
P2	out	Ms2’’			Ms1’	Ms2’		
P3	out	Ms1’’	Ms2’’					
Lookup	in		Ms2’’	x/y’’’				
	op		1/x	atan				
	out			1/x’’	atan(x/y)’’’			

Implementing the "Angle" Function

There are a number of alternative ways to implement the angle function. Taking the arctangent of the ratio of imaginary to real parts of the complex number is the most obvious. This involves a division and the arctangent function that is shown in Figure 4 as implemented by a look-up table in the form of a block RAM initialized with appropriate data. This is also shown in the mapping and scheduling as it makes more use of the DSP block.

Another option is to use the CORDIC algorithm that can be implemented using a small amount of hardware, if the iterations can be executed over a number of clock cycles. As it happens, the angle calculation needs to be done only once per burst and a small amount of latency is not a problem. There is, then, a trade-off between using block RAM (EBR) resource for a look-up table and FPGA LUTs for the adders in the CODRIC. There are scaling problems when using a 1/x look-up to implement y/x, so the CORDIC implementation was used in the Lattice OFDM reference design. The read address for the data output buffer was adjusted to compensate for the extra latency introduced by the serial implementation of the CORDIC.

Schmidl and Cox Simulation Results

The Schmidl and Cox algorithm was implemented in Verilog RTL and simulated in Modelsim. Figure 6 shows the Matlab reference results for a SUI-1 channel model and frequency offset of 0.13 of a sub-carrier spacing. The green horizontal block on the correlation plot shows where the peak detector is active. The red vertical line shows the point where the burst start has been detected. The metric shown in the plot is taken from the Schmidl and Cox paper. As this involves a division to form the metric, it

was decided to use a different method to locate the burst start in the hardware implementation, as discussed below.

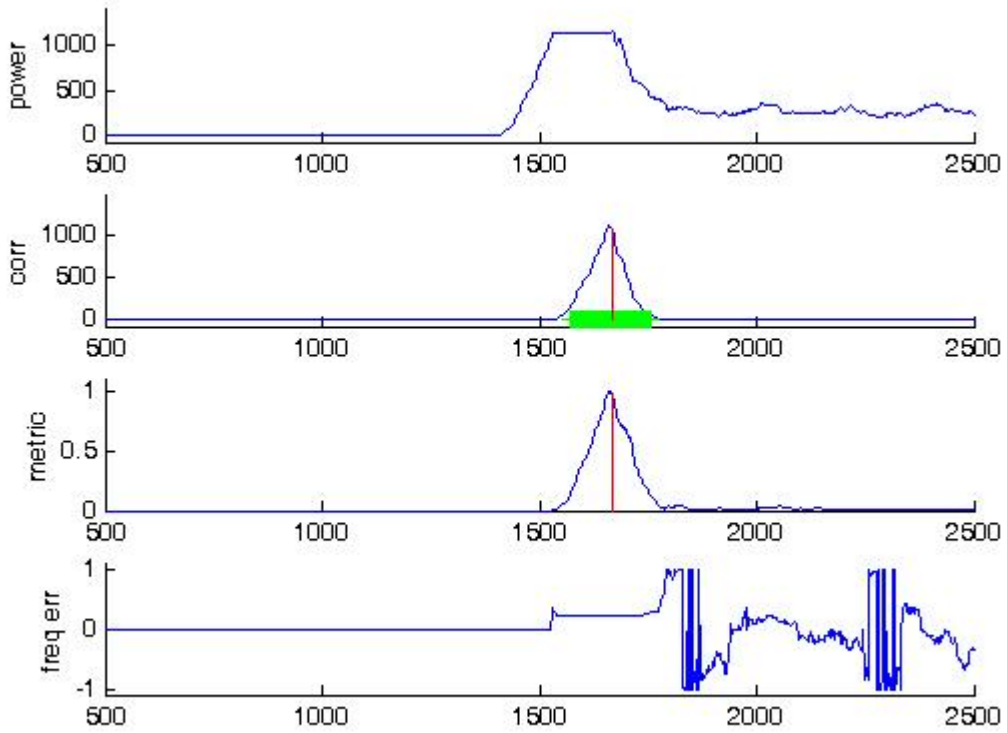

Figure 6: Matlab Plot Showing Power, Correlation and Frequency Error Against Sample Number

The RTL simulation results are shown in Figure 7. The burst start is detected by locating the peak in the correlation value under the condition that it is at least 1/8 of the power level. To mitigate the effects of noise, the peak detector locates the point on the downward slope that is 7/8 of the peak value and then backs-off from this by a number of samples that is determined by the cyclic prefix length. This back-off value was determined empirically by collecting data from a number of Matlab simulations.

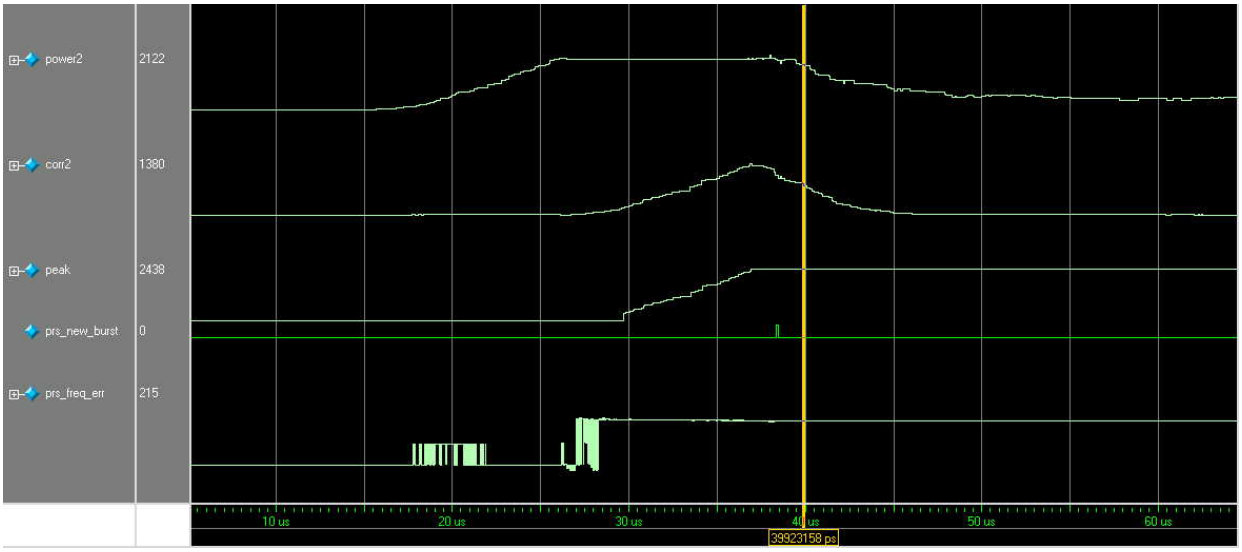

Figure 7 - RTL Simulation Waveform Plot of Key Signals in Schmidl and Cox Implementation

The prs_freq_err signal is scaled such that 215, as seen in the plot, equates to 0.132 of a carrier spacing. This compares well with the Matlab result, representing an error of 1.5%.

Implementation Results

The design was implemented as part of the complete OFDM reference design. This section gives the resource usage and Fmax for the timing and frequency offset estimation block.

Registers	406

Slices	602
EBR	2
DSP blocks	½

The design achieved an Fmax of 110Mhz, which is twice as fast as necessary. This means it would be possible to process two bursts in parallel by increasing the storage. Such a capability would be useful in a basestation supporting more than one channel.

Conclusions

This paper has shown how the flexible sysDSP blocks in a LatticeECP FPGA can be combined with a small amount of logic implemented in LUTs and 2 block memories to create a simple engine capable of performing two variants of a key algorithm in WiMAX OFDM signal processing. This generic structure could be adapted to perform a wide range of signal processing tasks at sample rates of up to 50Ms/s. More complex algorithms could be implemented in this way by using a small amount of extra resource for additional registers while still being able to support respectable sample rates.

References

[1] Moose P., "A Technique for Orthogonal Frequency Division Multiplexing Frequency Offset Correction", *IEEE Transactions on Communications*, Vol. 42, No. 10, pages 2908-2914, October 1994

[2] Jan-Jaap van de Beek, Magnus Sandell, Per Ola Börjesson, "ML Estimation of Time and Frequency Offset in OFDM Systems", IEEE Transactions on Signal Processing, April 1996.

[3] T. M. Schmidl and D. C. Cox, "*Robust frequency and timing synchronization for OFDM*," IEEE Trans. Commun., vol. 45, pp. 1613--1621, Dec. 1997.

[4] T. Yücek and M. K. Nezami, Joint Channel and Frequency Offset Estimation for OFDM Systems, IEEE Military Communications Conference, Monterey, CA, October 31 - November 3, 2004.

[5] H. Liu and U. Tureli, "A high efficiency carrier offset estimator for OFDM communications," IEEE Communications Letters, CL-2 (4), 1998.

[6] “OFDM Transceiver Reference Design”, Lattice Semiconductor OFDM Transceiver design package, 2005.

[7] “Implementation of an OFDM Wireless Transceiver using IP Cores on an FPGA”, Lattice Semiconductor white paper, 2005.

###