EFFICIENT CHIP DESIGN FOR PULSE SHAPING

Markus Rupp[1] *Jaiganesh Balakrishnan*[2]

[1]Lucent Technologies, Holmdel, NJ, USA
[2]Cornell University, Ithaca, NY, USA

ABSTRACT

Pulse shaping is a crucial operation for every modem transceiver. It usually requires a large portion of the modem's computational power and, therefore, an efficient implementation can be very beneficial to the overall modem design. In particular, in the transmitter part where well defined signals are to be computed, the typical pulse shaping operation can be greatly simplified. While a standard filter design requires many mult/add operations, the transmitter pulse shaping filter could also be designed by a pure look-up table. However, for higher constellations and large pulse shaping filter lengths the look-up table can become too large to implement. This paper presents a new idea for implementing the pulse shaping filter for a modem transmitter. It will be shown that for QPSK and other higher order modulations, like 16/32/64 QAM, no multiplication is required at all. Further, the number of multiplications for 8PSK (and 16/32PSK) can be reduced to a very small number that is independent of the number of coefficients necessary for implementing the filter function.

1. INTRODUCTION

Pulse shaping and the spectral forming operations involved with it are important functions for every transmitter and receiver (see, for example, [1, 2, 3]). The pulse shaping filter has to perform the following function:

$$p_l(k) = \sum_{m=0}^{M-1} s(k-m)\, c(Lm+l)$$

with $s(k)$ the QPSK symbols to be transmitted, $c(k)$ the filter coefficients and L the oversampling factor. The index l on the outcome defines the phase to be transmitted. There are L phases with M coefficients each, thus LM is the complete number of coefficients. Let us assume that the (real-valued) coefficients are organized in a two-dimensional array of the form:

m	0	1	2	3
$l=0$	c_0	c_4	c_8	c_{12}
1	c_1	c_5	c_9	c_{13}
2	c_2	c_6	c_{10}	c_{14}
3	c_3	c_7	c_{11}	c_{15}

Implementing a filter of this form requires $2ML$ real multiplications and additions. Depending on the hardware design this would require a considerable number of multipliers and adders. In the following sections we describe how this operation can be implemented with only adders, saving a large number of gates in a VLSI implementation.

2. QPSK

For QPSK it is not important for the symbols to be $\frac{\sqrt{2}}{2}\{(1+j),(1-j),(-1-j),(-1+j)\}$. They could also be rotated and in our case it is assumed that they are of the set $\{1, j, -1, -j\}$. Thus, multiplying with a coefficient of the filter does not really require a multiplication, but rather a selection of whether the real or imaginary part is addressed and whether addition or subtraction is required. Assume that each QPSK symbol is represented by two bits $\{b_1 b_0\}$. The mapping could be as shown in Table1. Other mappings, for example Gray encoding, are possible by adding another look-up table at the input. If the mapping in Table 1 is used, then the bit b_1 selects whether real or imaginary part is used and b_0 selects whether addition or subtraction of the coefficient is required. Thus, the real and imaginary parts of the filter output can be written as:

b_1	b_0	S
0	0	1
1	0	j
0	1	-1
1	1	-j

Table 1. QPSK symbol mapping.

$$\text{Real}\{p_l(k)\} = \sum_{m=0}^{M-1} c(Lm+l)\{1-2b_0(k-m)\}\{1-b_1(k-m)\} \quad (1)$$

$$\text{Imag}\{p_l(k)\} = \sum_{m=0}^{M-1} c(Lm+l)\{1-2b_0(k-m)\}b_1(k-m). \quad (2)$$

A chip structure that is capable of performing this operation is depicted in Figure 1. Note that, with this basic operation, an arbitrary number of filter coefficients can be assigned to compute the pulse shape. This operation can be concatenated for various filter lengths. Further, the filter output for the various phases can be computed, either

0-7803-5599-7/99/$10.00©1999 IEEE

serially with just one of these operations, or in parallel with L such streams.

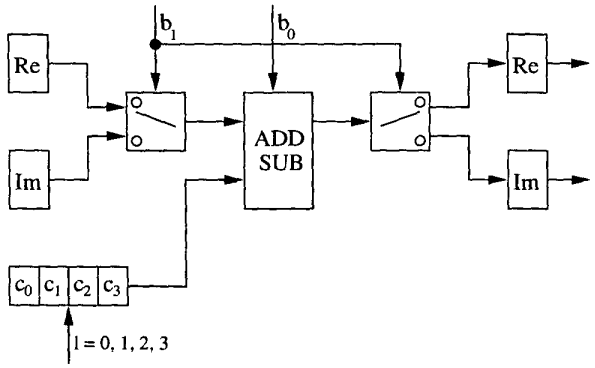

Figure 1. *Basic Operation for QPSK*

Alternately, we can use a single QPSK structure to perform all the multiplications corresponding to the different filter taps. The result of the earlier mult/adds needs to be fedback to the input of this structure. We need a latch to store the outputs of the previous computations. Further, the bits b_1b_0 and the coefficients corresponding to the different symbols need to be selected for the appropriate multiplications. This implementation is shown in Figure 2.

Example:
Assuming a 1ns delay for the add/sub structure and neglecting the multiplexer delays, a filter spanning six symbol durations ($M = 6$) with eight phases ($L = 8$) can be implemented serially in $M \times L \times$ 1ns=48ns with only one add/sub hardware or in parallel in 6ns with eight times the hardware effort. This limits the transmitted data rate to 40Mbit/s in the serial case and 318Mbit/s in the parallel case. A complete ROM solution allows faster implementation, but is only possible if the length of the filter is not too large. For the above example, an address space of 12 bits is required for each of the eight register banks corresponding to the eight phases. For an eight bit mantissa length of coefficients $2 \times 2^{12} \times 8 \times$ 8bit=512Kbit ROM is required. This ROM solution could give the result in 1ns allowing 2Gbit/s maximal transmission rate. Finally, a solution with adders and multipliers would require considerably more gates and not allow more than 53Mbit/s, assuming 6ns for a mult/add operation and six of them concatenated. Note that, our scheme offers many more possibilities for implementation allowing a wide range of timing and area constraints.

3. 16 QAM

Computing the same pulse shape for 16QAM is not very different. Note that the 16QAM can be dissected into four QPSKs. Figure 3 depicts how this can be done. The upper part of the figure shows the original 16 QAM and a version that is rotated by 90 degrees. From this rotated version four subsets are taken, all a QPSK on their own, but shifted from the origin by one of four values $\{2, 2j, -2, -2j\}$. Note that these values correspond to the symbols of a QPSK constel-

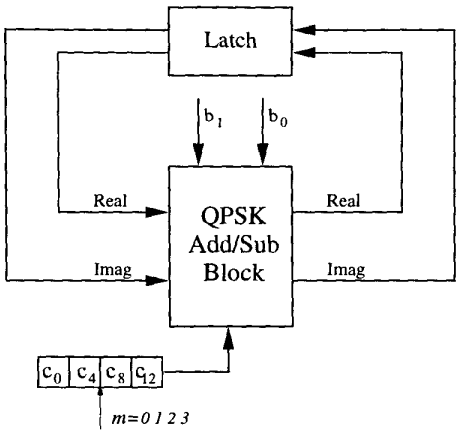

Figure 2. *Feedback Implementation Structure for QPSK*

lation multiplied by a factor of two. Since, all four subsets are just laterally shifted versions of each other, the basic operation for the QPSK can remain the same for each subset and only a correction needs to be made corresponding to the shift. It is assumed that the 16QAM symbol is represented by a four bit word $b_3b_2b_1b_0$. The two lower bits can be interpreted as before, while the two upper bits now define one of the four subclasses. This can be implemented, as shown in Figure 4, by cascading two of the basic QPSK structures. The first of them has b_1b_0 as the two input bits. The second structure has b_3b_2 as the inputs and the coefficients for this have to be multiplied by two. This is achieved by shifting the coefficients to the left by one bit. As in QPSK, this block can be concatenated serially to achieve any desired filter length. An alternate implementation is shown in Figure 5. The block labeled *L-Shift* performs a left shift operation by one bit.

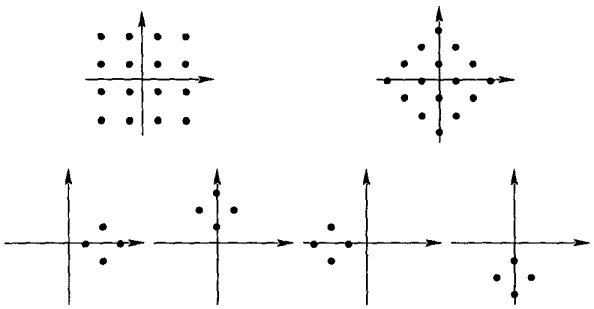

Figure 3. *Splitting a 16QAM into four QPSKs.*

Another interesting alternative is the implementation shown in Figure 6 (For the sake of convenience $M = 2$ is assumed for this figure). Here, the first step operations corresponding to b_1b_0 are combined in one cascade for the various filter taps, while the second steps are combined in a separate cascade. Although this method requires two additional adders, the complexity can be lower since the two streams can work with the same (lower) precision until they

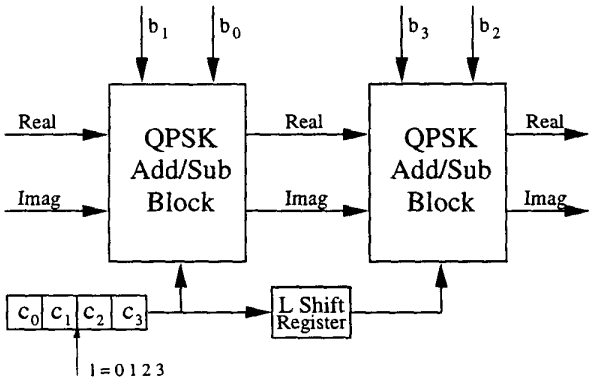

Figure 4. *Basic Operation for 16QAM.*

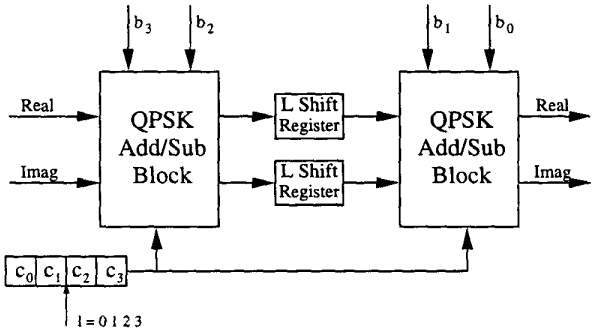

Figure 5. *Alternate Implementation for 16QAM.*

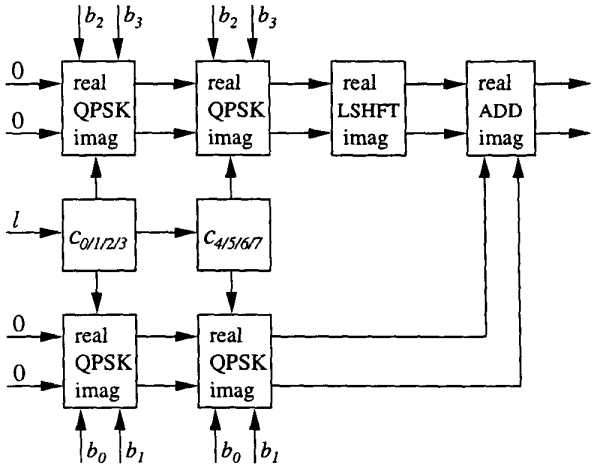

Figure 6. *Multicascade Implementation for 16QAM.*

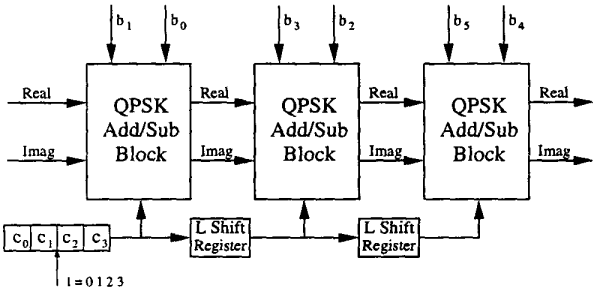

Figure 7. *Basic Operation for 64QAM.*

are combined at the very end. Thus, only the last adder explicitly requires one bit more on precision. Another advantage of this structure is that the overall computation time is now $Mt_c + t_{add}$, while for the previous two realizations a computational time of $2Mt_c$ is required (t_c being the time through a cascade element).

4. 64/32 QAM

We can, similarly, implement pulse shaping for a 64QAM modulation scheme. The 64QAM constellation can be split into four 16QAM constellations, each of which is, centered around $\{4, 4j, -4, -4j\}$. The pulse shaping operation is realised by cascading one more basic QPSK structure to the 16QAM structure, as shown in Figure 7. Bits b_5b_4 are the inputs to the third block and the coefficients for this have to be multiplied by four. This operation is performed using a shift register. As in 16 QAM, an alternate implementation using left shift registers, as shown in Figure 5, and the multicascade solution of Figure 6 are possible.

Since the 32QAM constellation points are a subset of the 64QAM constellation, it is sufficient to map the five 32QAM bits to six bits, corresponding to the 64QAM constellation, and use the 64QAM structure.

5. 8 PSK

The 8PSK constellation can be split into two QPSK constellations. The two sets being $\{1, j, -1, j\}$ and $\{1, j, -1, -j\} \times \exp(j\pi/4)$. Since the two subsets are just rotated versions of each other, the basic operation for QPSK can remain the same for each subset (however, with two real and two imaginary streams as indicated in Figure 8) and a correction needs to be made before the two sub-results are added. It is assumed that the 8PSK symbol is represented by a three bit word $b_2b_1b_0$. The two lower bits can be interpreted as before, while the upper bit now defines one of the two subclasses. Thus, the previous structure can remain the same with an additional two-way multiplexer at the beginning to select one of the two streams and a final procedure at the end that combines the two streams. Similar to (1) and (2), the streams for the two subsets are now given by:

$$\begin{aligned}\text{Real}\{p_{I,l}(k)\} &= \sum_{m=0}^{M-1} c(Lm+l)\{1-2b_0(k-m)\} \\ &\quad \times\{1-b_1(k-m)\}b_2(k-m) \qquad (3)\\ \text{Imag}\{p_{I,l}(k)\} &= \sum_{m=0}^{M-1} c(Lm+l)\{1-2b_0(k-m)\} \\ &\quad \times b_1(k-m)b_2(k-m) \qquad (4)\end{aligned}$$

$$\text{Real}\{p_{II,l}(k)\} = \sum_{m=0}^{M-1} c(Lm+l)\{1-2b_0(k-m)\} \times\{1-b_1(k-m)\}(1-b_2(k-m)) \quad (5)$$

$$\text{Imag}\{p_{II,l}(k)\} = \sum_{m=0}^{M-1} c(Lm+l)\{1-2b_0(k-m)\} \times b_1(k-m)\{1-b_2(k-m)\}. \quad (6)$$

Finally, the two streams need to be added:

$$\text{Real}\{p_l(k)\} = \text{Real}\{p_{I,l}(k)\} + 0.707\,[\text{Real}\{p_{II,l}(k)\} - \text{Imag}\{p_{II,l}(k)\}] \quad (7)$$

$$\text{Imag}\{p_l(k)\} = \text{Imag}\{p_{I,l}(k)\} + 0.707\,[\text{Real}\{p_{II,l}(k)\} + \text{Imag}\{p_{II,l}(k)\}] \quad (8)$$

Figure 9 shows the combiner. Note that, this combiner requires one complex multiplication and one complex addition. This operation is required for each phase. Depending on the required accuracy this multiplication might be realized in shift/adder form. One simple, but relatively accurate, operation is to approximate $1/\sqrt{2}$ by 5/7. In this case, the upper stream is multiplied by seven (can be done by scaling the coefficients differently), while the lower stream is multiplied by five (also by scaling the coefficients). Now the rotation for the lower scheme is simply done by multiplying with $(1+j)$, which is a subtraction and an addition operation for the real and imaginary parts, respectively.

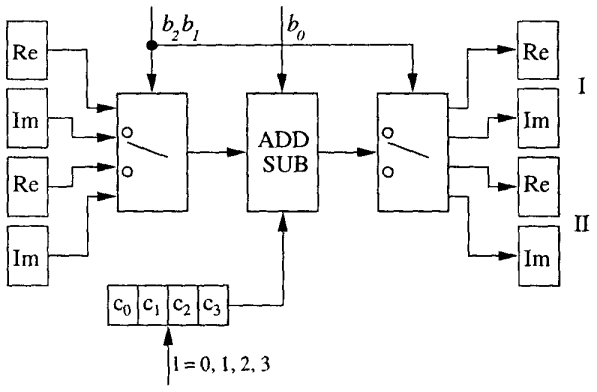

Figure 8. *Basic Operation for 8PSK.*

Note that, for 16/32PSK a similar scheme applies. As demonstrated for 8PSK in Figure 8, the number of parallel streams needs to be increased, while the number of add/sub operations remains the same. Thus, the slight increase in complexity is only due to a greater number of registers and multiplexers and the cost remains essentially that of a QPSK with some overhead for combining the streams.

6. IMPLEMENTATION

The proposed schemes for 4/8PSK and for 16/32/64QAM have been implemented in VHDL. A Synopsys FPGA compiler was used to optimize the code for Altera FPGAs.

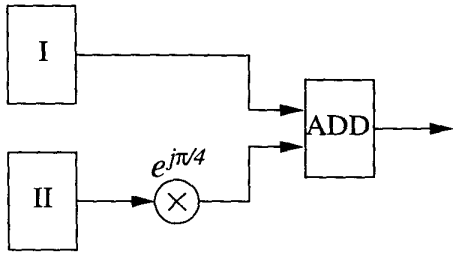

Figure 9. *Combining the two streams.*

A T/4 sampling rate was used with a pulse shape that stretches out over ±3 symbols, thus requiring 24 coefficients. Table 2 shows some of the results obtained by the FPGA compiler (Synopsys and Altera). 16QAM, 32QAM, and 64QAM were implemented using the realization in Figure 6. This accounts for the identical delay in all the realizations. The 32QAM and 64QAM have identical performance, since, 32QAM is different from 64QAM only in the sense that a different subset of symbols are used. Note that, the maximum delay and the number of Look-Up Tables (#LUTs) is a result of the Synopsys FPGA analyzer, while number of Logic Cells (#LCs) is an outcome of the MAXPLUS2 from Altera. A comparable FIR-filter, implemented with multipliers and adders, requires 3064LCs.

Modulation	Maximum Delay	# LUTs	# LCs
QPSK	36ns	879	309
8PSK	36ns	1906	665
16QAM	36ns	1746	602
32QAM	36ns	2599	906
64QAM	36ns	2599	906

Table 2. *Performance results based on Altera FPGAs.*

REFERENCES

[1] J.G. Proakis, "Digital Communications," McGraw Hill, 1989.

[2] T.S. Rappaport, "Wireless Communications," Prentice Hall, 1996.

[3] H. Meyr, et. al., "Digital Communication Receivers," John Wiley & Sons, 1998.