SOFTWARE IMPLEMENTATION OF 802.11a BLOCKS ON SANDBLASTER DSP

Vaidyanathan Ramadurai, Sanjay Jinturkar, Sitij Agarwal, Mayan Moudgill, John Glossner

Sandbridge Technologies, 1 North Lexington Avenue, White Plains, NY 10601
vramadurai@sandbridgetech.com

ABSTRACT

In this paper, we describe the design and implementation of software blocks for 802.11a receiver on Sandblaster DSP. A software solution provides high reusability, low cost and short development time when compared to dedicated hardware solutions. A significant challenge faced is in achieving high throughput and stringent latency requirements.

802.11a is an IEEE standard that operates in the 5GHz band using Orthogonal Frequency Division Multiplexing (OFDM). OFDM divides a data signal across 48 separate sub-carriers to provide higher data rates and minimize the multi-path propagation effects. The standard supports multiple data rates from 6Mbps to 54Mbps and involves high computational complexity.

The steady state 802.11a receiver consists of an FFT and removing pilot/DC, demapper, deinterleaver, depuncture, FEC decoder and CRC. We explore techniques for optimizing individual blocks and also combining multiple blocks to increase the overall performance and to meet real time throughput and latency requirements. There is significant data movement between individual blocks like FFT, demapper, deinterleaver and depuncture and we explain how multiple blocks could be coupled to significantly reduce the instruction cycle count as well as data transfers.

Traditional software deinterleavers have been implemented using table look-ups. We explain how table-look ups could be merged with other compute intensive and data intensive blocks like demapper and depuncturer thereby speeding up the entire system. Instead of optimizing blocks for a specific data rate, we propose optimizations that could be exploited for any data rate specified by the 802.11a standard.

1. INTRODUCTION

The OFDM system provides a wireless LAN with data payload communication capabilities of 6, 9, 12, 18, 24, 36, 48, and 54 Mbit/s. The support of transmitting and receiving at data rates of 6, 12, and 24 Mbit/s is mandatory. The system uses 52 subcarriers that are modulated using binary or quadrature phase shift keying (BPSK/QPSK), 16-quadrature amplitude modulation (QAM), or 64-QAM. Forward error correction coding (convolutional coding) is used with a coding rate of 1/2, 2/3, or 3/4.

To handle the high data rate requirements, several hardware based solutions like ASICS and FPGAs exist for 802.11a/g. However, such solutions lack the flexibility and reusability of software based solutions. Also, software based methods reduce time to market by quick modifications.

In this paper, we consider a software implementation of 802.11a blocks on SandBlaster DSP. Since 802.11a demands very high throughput and real time latency requirements, a software implementation of the baseband functions becomes very challenging. Section 2 gives a brief overview of SandBlaster DSP architecture. Section 3 provides an introduction to the 802.11a receiver. In section 4, we show the software implementation and optimization of demapper. In Sections 5 and 6 we will discuss the optimizations of deinterleaver and depuncturer respectively. Finally, conclusions are drawn in section 7.

2. SANDBLASTER DSP

Sandbridge Technologies has developed the Sandblaster architecture for a convergence device [1,2]. The Sandblaster architecture supports the data types necessary for convergence devices including RISC control code, DSP, and Java.

As shown in Figure 1, the design includes a unique combination of modern techniques such as a SIMD Vector/DSP unit, a parallel reduction unit, and a RISC-based integer unit. Each processor core provides support for concurrent execution for up to eight threads of execution. All states may be saved from each individual thread and no special software support is required for interrupt processing. The machine is partitioned into a

RISC-based control unit that fetches instructions from a set-associative instruction cache. Instruction space is conserved through the use of compounded instructions that are grouped into packets for execution.

The memory subsystem has been designed carefully to minimize power dissipation. The pipeline design in combination with the memory design ensures that all memories are single ported and yet the processor can sustain nearly 4 taps per cycle for a filter (the theoretical maximum) in every thread unit simultaneously. A RISC-based execution unit, depicted in the center of Figure 1, assists with control processing.

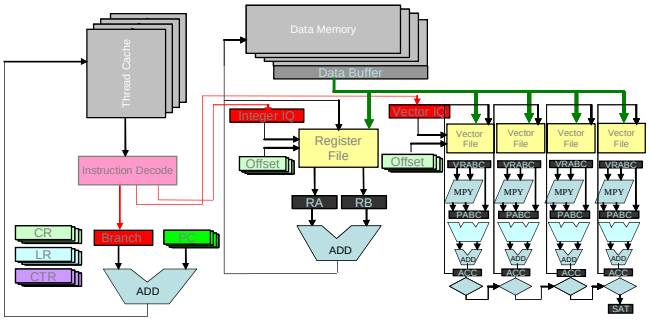

Fig 1: Sandblaster DSP

For the control code, a 16 entry, 32-bit register file per thread unit provides for very efficient control processing. Common integer data types are typically stored in the register file. This allows for branch bounds to be computed and addresses to be efficiently generated. Intensive loop processing is performed in the SIMD/Vector unit depicted on the right side of Figure 1. Each cycle, a 4x16-bit vector may be loaded into the register file while two vectors are being multiplied, saturated, reduced (e.g. summed), and saturated again. The branch bound may also be computed and the instruction looped on itself until the entire vector is processed. This may be specified in as little as 64-bits.

To enable signal processing in software, the processor supports many levels of parallelism. Thread-level parallelism is supported by providing hardware support for up to 8 independent programs to be simultaneously active on a single Sandblaster core. This minimizes the latency in physical layer processing. Since many algorithms have stringent requirements on response time, multithreading is an integral technique in reducing latencies. The data-level parallelism (SIMD) is supported through the use of a Vector unit.

3. 802.11a RECEIVER

The OFDM modulation scheme used in 802.11a distributes the data over 52 subcarriers on a 20MHz channel to mitigate the effects of multipath. Among the 52 subcarriers, 48 are for data and 4 are for pilot signals used for tracking. Each subcarrier is 312.5kHz wide, giving raw data rates from 125kbits/s to 1.125Mbits/s per subcarrier depending on the modulation type – binary phase shift keying (BPSK), quaternary PSK (QPSK), 16-quadrature amplitude modulation (QAM), or 64-QAM – and the error-correcting code rate (1/2, 2/3, or 3/4). The composite signal therefore has a data rate ranging from 6Mbits/s to 54Mbits/s in the 20MHz channel [11]. Table 1 lists the mode-dependent parameters for the 802.11a standard.

IEEE 802.11a Table 78—Rate-dependent parameters

Data rate (Mbits/s)	Modulation	Coding rate (R)	Coded bits per subcarrier (NBPSC)	Coded bits per OFDM symbol (NCBPS)	Data bits per OFDM symbol (NDBPS)
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16-QAM	1/2	4	192	96
36	16-QAM	3/4	4	192	144
48	64-QAM	2/3	6	288	192
54	64-QAM	3/4	6	288	216

Table1

The baseband physical layer block diagram of an 802.11a receiver is shown in Fig 2.

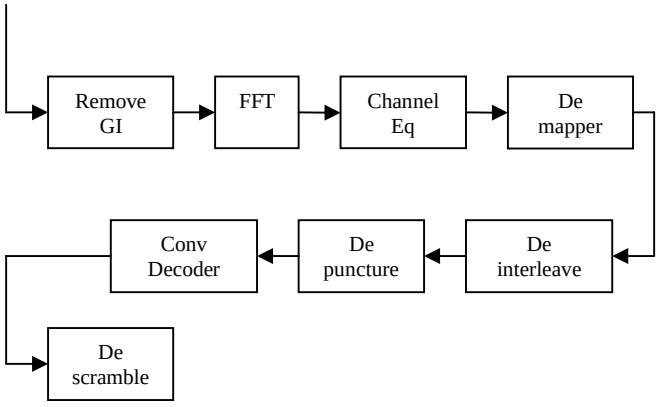

Fig 2. 802.11a receiver physical layer

The digital data is frequency corrected and after removing the guard intervals given to an. FFT block. The FFT

converts the time domain samples to frequency domain sub-carriers. 802.11a uses a total of 52 sub-carriers out of which 48 are data symbols are 4 are pilots. A channel equalizer is then employed to avoid inter symbol interference (ISI) and inter carrier interference (ICI). The demapper takes the 48 complex symbols and depending on the type of modulation demodulates the subcarriers into data bits (hard/soft bits). The demodulated bits are deinterleaved, depunctured, decoded using a convolutional decoder and descrambled. In this paper, we will consider the following blocks implemented on SandBlaster DSP:

1. Demapper
2. Deinterleaver
3. Depuncuturer

4. DEMAPPER

We employ the soft demapper with the channel state information (CSI) for 16QAM and 64QAM demodulation as described in [6]. It has been shown that the simplified soft output demapper greatly outperforms the hard decision demapper [6].

The matlab code for the 64QAM is shown in Fig. 3. chI, chQ are the CSI and rxI, rxQ are the inputs to the demapper. Every sub-carrier produces 6 soft output bits after demodulation.

The equivalent C code implemented in SandBlaster DSP is shown in Fig.4 The loop is written in such a way that it gets vectorized, i.e. for e.g. bit0 for 4 sub-carriers can be computed at the same time. 4 input elements and 4 CSI elements can be loaded, multiplied and added to get bit0 from each of the 4 sub-carriers. This is done similarly for calculating the other bits from all the sub-carriers.

```
for k = 1:48

  %% Determine the 3 inphase bits
  chMagSq(k) = chI(k).^2 + chQ(k).^2;
  % mag sq of each entry of the vector

  bit0 =  rxI(k)*chI(k) + rxQ(k)*chQ(k);
  bit1 =  4 * chMagSq(k) - abs(bit0);
  bit2 =  2 * chMagSq(k) - abs(bit1);

  %% Repeat computations to get the 3
  quadrature bits

  bit3 =  -rxI(k)*chQ(k) + rxQ(k)*chI(k);
  bit4 =  4 * chMagSq(k) - abs(bit3);
  bit5 =  2 * chMagSq(k) - abs(bit4);
```

Fig 3. Matlab code for 64QAM demapper

```
for(k=0; k<48; k++){
  softBits_t[48*0+k] = (rxI[k]*chR[k] +
            rxQ[k]*chI[k]) >> QAM64_SF;
  softBits_t[48*1+k] = 4*chMagSq[k] –
            SB_ABS(softBits_t[48*0+k]);
  softBits_t[48*2+k] = 2*chMagSq[k] –
            SB_ABS(softBits_t[48*1+k]);
  softBits_t[48*3+k] = (rxQ[k]*chR[k] –
            rxI[k]*chI[k]) >> QAM64_SF;
  softBits_t[48*4+k] = 4*chMagSq[k] –
            SB_ABS(softBits_t[48*3+k]);
  softBits_t[48*5+k] = 2*chMagSq[k] –
            SB_ABS(softBits_t[48*4+k]);
}
```

Fig 4. C code for 64QAM demapper

5. DEINTERLEAVER

In 802.11a, all encoded data bits shall be interleaved by a block interleaver with a block size corresponding to the number of bits in a single OFDM symbol. The interleaver is defined by a two-step permutation. The first permutation ensures that adjacent coded bits are mapped onto nonadjacent subcarriers. The second ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation and, thereby, long runs of low reliability (LSB) bits are avoided.

The first permutation is defined by the rule:

$i = s \cdot \text{floor}(j/s) + (j + \text{floor}(16 \cdot j/N_{CBPS})) \bmod s$
$\{j = 0,1,\ldots N_{CBPS} - 1\}$

The value of s is determined by the number of coded bits per subcarrier, N_{BPSC}, according to
$s = \max(N_{BPSC}/2,1)$

The second permutation is defined by the rule:

$k = 16 \cdot i - (N_{CBPS} - 1)\text{floor}(16 \cdot i/N_{CBPS})$, where
$\{i = 0,1,\ldots N_{CBPS} - 1\}$

NCBPS is the number of coded bits per symbol and NBPSC is the number of coded bits per sub-carrier.

Let us consider the deinterleaver for data modulated by 16QAM. Here the number of soft bits from the demapper is 192. We will observe the deinterleaver permutation by combining permutations 1 and 2 mentioned above. We will name the indices from 0 to 191. Given below is the input for the first 60 elements and final deinterleaved output of those 60 elements. Note that permutation 2 also implicitly produces a (12x16) transpose of the final output. If we ignore the transpose and look at

the pattern of the data shuffling, row 0, row 2 and row 4 are unchanged, in rows 1 and 3 the elements are shuffled as {x1,x0,x3,x2} taken 4 elements {x0,x1,x2,x3} at a time.

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59

0	1	2	3	4	5	6	7	8	9	10	11
13	12	15	14	17	16	19	18	21	20	23	22
24	25	26	27	28	29	30	31	32	33	34	35
37	36	39	38	41	40	43	42	45	44	47	46
48	49	50	51	52	53	54	55	56	57	58	59

Hence we can deinterleave 4 elements at a time, either using just a vector copy or a vector shuffle and a vector copy. Fig 5 shows the code snippet for the 16QAM case. The code shows only the shuffling for 4 elements for 4 rows. This will be executed 4x3 times for all rows and all columns. By inlining and software pipelining, the 16QAM deinterleaving is done in ~200 cycles. A table lookup or even direct copy would have taken atleast 384 cycles. Also, by doing in-place deinterleaving, we could perform the data shuffling only for the odd rows. This would improve the cycle count further by 50%. A similar approach has been used for all other modulation schemes like BPSK, QPSK and QAM64 which produces different number of softbits.

```
void
qam16_deinterleave(
                        short * restrict X,
                        short * restrict Y,
                        int instride,
                        int outstride
                )
{
  int i;
  int j;
  short A[16];

  A[4*0+0] = X[instride*0+0];
  A[4*0+1] = X[instride*0+1];
  A[4*0+2] = X[instride*0+2];
  A[4*0+3] = X[instride*0+3];

  A[4*1+0] = X[instride*1+1];
  A[4*1+1] = X[instride*1+0];
  A[4*1+2] = X[instride*1+3];
  A[4*1+3] = X[instride*1+2];

  A[4*2+0] = X[instride*2+0];
  A[4*2+1] = X[instride*2+1];
  A[4*2+2] = X[instride*2+2];
  A[4*2+3] = X[instride*2+3];

  A[4*3+0] = X[instride*3+1];
  A[4*3+1] = X[instride*3+0];
  A[4*3+2] = X[instride*3+3];
  A[4*3+3] = X[instride*3+2];

  for(i=0; i<4; i++){
    for(j=0; j<4; j++){
      Y[outstride*j+i] = A[i*4+j];
    }
  }
}
```

Fig 5. C code for 16QAM deinterleaving

6. DEPUNCTURER

Puncturing is a procedure for omitting some of the encoded bits in the transmitter (thus reducing the number of transmitted bits and increasing the coding rate) and inserting a dummy "zero" metric into the convolutional decoder on the receive side in place of the omitted bits. In 802.11a, there are two puncturing modes, rates r = 2/3 and r = ¾.

In r = ¾, there are two zero bits inserted for every 4 bits as shown in Fig 6. The C code for QAM16, ¾ depuncture is also shown. The input to the depuncture is 192 soft bits from the QAM16 demapper. The ¾ depuncture produces 288 bits output after inserting zeros. In here again, the copy {B2,A3,B3,A4}, {B5,A6,B6,A7}, etc vectorizes.

A0	A1	A2	A3	A4	A5	A6	A7	A8
B0	B1	B2	B3	B4	B5	B6	B7	B8

Fig 6. r=3/4 depuncture

```
k = 0;
out[0] = in[k++];
out[1] = in[k++];
out[2] = in[k++];

for(m=5; m<288; m+=6){
  for(i=0; i<4; i++){
    out[m+i] = in[k++];
  }
}
```

Fig 6. QAM16 r=3/4 depuncture

A table lookup based method can also be used that will combine the three blocks: demapper, deinterleaver and depuncturer. The output soft bits from the demapper could be directly routed to its appropriate location after depuncturing by using a lookup table. This method would be optimal in terms of memory access as we could avoid the data transfers from demapper to deinterleaver and from deinterleaver to depuncturer. Also this method is very useful in parallelizing the blocks. One could partition the sub-carriers across multiple threads and each thread could independently process the sub-carriers from demapping up to depuncturing.

In SandBlaster DSP we use 8 threads to process the 48 sub-carriers from demapper to depuncturing to keep up real time requirements. Since every 802.11a symbol arrives in 4uSec, a 75MHz thread processor has to complete processing a symbol in 300 cycles. In this case, every thread works on 6 symbols or sub-carriers and run completely in parallel.

7. CONCLUSION

In this paper, we have described a software implementation of 802.11a receiver blocks on SandBlaster DSP. We have discussed critical blocks like demapper, deinterleaver and depuncture and their software optimizations. Instead of using traditional table lookup based methods for interleaving, we have found special patterns on data shuffling for these blocks. Such patterns have been exploited to optimize the blocks. Also, table lookup based methods have been used to merge multiple blocks and achieve thread level parallelism.

REFERENCES

[1] John Glossner et al, “Sandblaster low power DSP”, IEEE 2004 Custom Integrated Circuits Conference, 2004, pp 575-581.

[2] Sanjay Jinturkar , John Glossner, Mayan Moudgill, Erdem Hokenek, “Programming the Sandblaster Multithreaded Processor”, GSPx 2003.

[3] IEEE 802.11b-1999, “Wireless LAN medium access control (MAC) and Physical layer (PHY) Specifications: High Speed Physical Layer Extension in the 2.4 GHz Band,” 1999.

[4] IEEE 802.11a-1999, “Wireless LAN medium access control (MAC) and Physical layer (PHY) Specifications: High Speed Physical Layer in the 5 GHz band,” 1999.

[5] M.J. Meeuwsen, O. Sattari, and B.M. Baas, “A full-rate software implementation of an IEEE 802.11a compliant digital baseband transmitter,” Proc. of IEEE Workshop on Signal Processing Systems (SIPS 2004), pp. 124-129, Oct. 13-15, 2004.

[6] F. Tosato and P. Bisaglia, “Simplified soft-output demapper for binary interleaved COFDM with application to HIPERLAN/2,” in Proc. IEEE ICC 2002, vol. 2, 2002, pp. 664–668.