CARMA DSP Utilization

D. W. Hawkins (dwh@ovro.caltech.edu)
Document Revision 1.0
December 12, 2003

Contents

1 Introduction 3
2 CARMA data processing 4
3 FFT benchmarking 6
4 Memory transfer benchmarking 8
5 Data accumulation benchmarking 10
6 Floating point conversion benchmarking 12
7 Multiplication benchmarking 13
8 Delay/lobe rotation calculation benchmarking 14
9 Data processing estimates 16
10 Performance improvement options 17
Table 1: CARMA correlator lag and spectral characteristics.

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Number of Lags</th>
<th>Number of spectral channels</th>
<th>Spectral channel spacing</th>
</tr>
</thead>
<tbody>
<tr>
<td>nominal (actual)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500MHz</td>
<td>32</td>
<td>17</td>
<td>31.25MHz</td>
</tr>
<tr>
<td>250MHz</td>
<td>64</td>
<td>33</td>
<td>7.81MHz</td>
</tr>
<tr>
<td>125MHz</td>
<td>80</td>
<td>41</td>
<td>3.13MHz</td>
</tr>
<tr>
<td>62MHz (62.5MHz)</td>
<td>96</td>
<td>49</td>
<td>1.29MHz</td>
</tr>
<tr>
<td>31MHz (31.25MHz)</td>
<td>112</td>
<td>57</td>
<td>558kHz</td>
</tr>
<tr>
<td>8MHz (7.81MHz)</td>
<td>128</td>
<td>65</td>
<td>122kHz</td>
</tr>
<tr>
<td>2MHz (1.95MHz)</td>
<td>128</td>
<td>65</td>
<td>31kHz</td>
</tr>
</tbody>
</table>

1 Introduction

The baseline plan for the CARMA First Light Correlator is to reuse the existing COBRA hardware. This reuse is conditional upon the following additional features over the COBRA correlator application:

- Support for narrower bandwidth (spectral line) modes.
- 125MHz data transmission over the LVDS cables for the 500MHz band.
- Fanout of the LVDS front panel data.
- Digital delay lines quantized to 1ns in the FPGAs.
- Sub-1ns delays applied by the DSPs.
- Lobe rotation at the DSP.
- DSP processing of 10 baselines per correlator board with up to 128-lags per baseline.
- Phase switching at \(\sim 1\)ms.

The last four items on this list can be grouped as DSP utilization issues. The COBRA Wideband correlator DSPs process at most 5 baselines each with 64-lags per baseline at a phase switch rate of 160Hz (6.25ms phase switch period), producing four bins of phase switched data. The CARMA correlator boards will process 10 baselines, with up to 128-lags per baseline (see Table 1). In addition to this, phase switching in CARMA will be increased to 1024 states per second, or a phase switch period on the order of 1ms. The 180-degree phase switching must complete and be demodulated before any delay compensation can be applied, a 16-state 180-degree sequence has been proposed, implying that delay correction can occur every 16ms.
2 CARMA data processing

The intended data flow for the CARMA correlation processing is

- Data is digitized at 1GHz.
- 180-degree phase switch demodulation is performed in hardware by either toggling the magnitude bit on the digitizer, or by toggling the magnitude bits once they are in the digitizer FPGAs.
- Delays quantized to 1ns will be compensated using the digital delay lines implemented in the digitizer FPGAs.
- Correlation data, phase switched at 1ms, will be accumulated either by the FPGAs directly, or by the DSP (integer data format).

**Note:** The proposed phase switching sequence consists of nested 180-degree and 90-degree phase states, with 180-degree phase switching at 1ms rate, and 90-degree phase switching at 16ms rate.

This means that the 1ms phase switches within each 16ms 180-degree sequence correspond to either a 0-degree bin or a 90-degree bin. Hence accumulation of the 1ms phase switches by the FPGA becomes a simple option (assuming of course that the 180-degree switching is removed in hardware).

- A 180-degree phase switch cycle completes every 16ms.
- The DSP will convert the single phase bins of new data per baseline to the DSPs floating point format every 16ms (each baselines phase bin data may correspond to a 0-degree or a 90-degree phase bin).
- The DSPs will FFT the floating point data every 16ms, and apply delay and lobe rotation corrections.
- The DSPs will then accumulate data for a complete Walsh cycle (500ms for CARMA).
- The 500ms data must be converted from the DSP floating point format to the hosts IEEE 754 floating point format. This task can be performed by the DSP, or by the host upon receipt of the 500ms data.
- Data is transferred to the host CPU every 500ms. A single CPU will collect data from a single CARMA band. A CARMA band requires two crates, so collecting data using one CPU requires that the crates be linked. PCI-to-PCI bridges have been tested that show this method is feasible ($1300 for the bridge option, versus $2400 if a second CPU was used).

Benchmarking tests for FFTs, memory transfers, complex vector multiplications, and processing algorithms are required to determine what will be the limiting factor for the correlator board DSPs.

The Texas Instruments DSPs used on the COBRA boards are the TMS320LC31 DSPs. These DSPs have two banks of internal SRAM of 32-bits by 1K each. The COBRA wideband correlator processes 5 baselines, and 64-lags per baselines, requiring 320 32-bit words of storage per 6.25ms integration. For COBRA, the phase switched data is stored in four phase bins, so 320 words are required for the current 6.25ms integration plus 1280 words for the accumulated phase switch bins. The total memory requirement is thus 1600 words.

The fastest FFT processing occurs when the FFT code and twiddle factors are in one 1K bank, and the data being processed is in the other. This places a limit of 1K on the available SRAM to place the data being processed. The COBRA system uses one of the DSP 1K banks to buffer the current
6.25ms integration data (320 words) and the phase bins into which this data will be accumulated (320 words). This data must be moved on and off chip for each phase switch integration.

The CARMA correlator system will process 10 baselines per board, and up to 128-lags per baseline. The maximum total number of words for a single phase switch integration is then 1280 words, which exceeds the internal memory of the DSP. In addition to this the phase switch period will be reduced to 1ms, with 180-degree phase switch demodulation required for 16ms before data can be FFTed and processed. If the 1ms data is processed by the DSP, then the 1280 words of the current integration must be moved off the FPGAs and accumulated with the 1280 words of the appropriate phase bin (with two phase bins requiring a total of 2560 words of storage). Since these numbers exceed the DSP internal memory storage, there will be additional memory transfers; FPGA-to-DSP SRAM for the current phase switch data (or perhaps FPGA-to-external SRAM, and then external SRAM-to-DSP SRAM), external SRAM-to-DSP SRAM for accumulation, DSP SRAM-to-external SRAM for storage. In addition to these data movements, every 16ms, the accumulated 16ms data will need to be moved on-chip for FFTing and delay/lobe rotation compensation, and the current accumulated spectra will need to be moved on-chip for accumulation with the data. A significant portion of the DSPs time is spent shuffling data, and since this data is over a common bus, it is unlikely the DMA operations will provide any benefit (though this will be investigated).

One option for reducing the DSP work-load is to perform the 1ms phase switch binning and accumulation in the FPGAs. This would reduce the DSP workload to processing data from the FPGAs every 16ms, and reduces the data traffic on the DSP bus. Accumulation of the 180-degree phase switch data requires that the digitizer board removes the 180-degree phase switch (either by toggling the digitizer magnitude control, or by toggling the bits once the data is on the digitizer FPGAs), and that the cross-correlation FPGAs retain their correlation counter contents for 16 × 1ms phase switches. This option complicates the cross-correlation FPGAs in that they now need additional information regarding which 1ms phase switch the 16ms sequence is to start on. However, this modification is likely a necessity.
3 FFT benchmarking

Texas Instruments provides several assembler source files that can be used to perform FFTs. The COBRA wideband correlator processed the phase switched data as four separate phase bins. Since the data in each phase bin was real-valued, a real-valued FFT was used. In the CARMA correlator, phase switch demodulation to two phase bins will be performed, so two real-valued FFTs can be performed to process the 0-degree and 90-degree phase bin data, or the 0-degree and 90-degree data could be combined to produce a single complex data set, and then a complex valued FFT could be used to produce the upper (positive frequencies) and lower sideband (negative frequencies) data.

Table 2 shows the results of FFT benchmarking tests of the real-valued FFT written by Alex Tessarolo (version 2.0) (see fft_benchmarking/fft_bench.c in the DSP code area of the COBRA CVS repository). This is the FFT used in the COBRA wideband correlator system. The benchmarking code consisted of a C-coded source file, a MATLAB generated array of input data and a spectrum to check against, the FFT assembly file under test (two real-valued FFT routines were tested), and the memory map of the code text and data sections (so that the text and data could be placed in on-chip DSP SRAM RAM0 or RAM1, or in external SRAM). This benchmarking approach results in an absolute minimum of additional overhead (eg. there is no RTOS present). Benchmark times were obtained by using an oscilloscope to measure the pulse time of an LED. The benchmark test pulses the LED for the duration of the FFT call, and runs the FFT test routine inside an infinite loop (each time through the loop, the calculated FFT is compared to the MATLAB generated spectra). The benchmarking tests reflect the time taken to call the assembly-coded FFT routine from C, and does not include any data movement overhead (other than whatever the FFT routine is doing internally).

The efficiency of a complex FFT is $O(N \log_2 N)$, and for a real-valued FFT, $N$ can be replaced with $N/2$. For FFT sizes 32, 64, and 128, this corresponds to operation orders of 64, 160, and 384. The best FFT times in Table 2: i.e., 47$\mu$s, 77$\mu$s, 147$\mu$s, correspond to these operational orders multiplied by a scale factor of 0.3$\mu$s/operation and an overhead of 28$\mu$s. It is important to note from the benchmarks in Table 2 that there is a significant penalty paid for off-chip processing of the FFTs. The implementation for the correlator FFT processing should always ensure that data is moved on-chip before performing the FFT.

Table 1 shows that the number of lags for each band is not always a power of two. The FFT used by the DSP is a radix-2 based FFT, so the DSP will always pad lag data to the nearest power of two for processing. Since the DSP is performing phase rotations after FFT, Hanning weighting of the data cannot be applied before the 16ms FFTs. Hanning weighting of the data is performed at a later stage of the data processing. When the data is returned to the lag domain for Hanning weighting, the data padding can be removed, and a non-radix-2 FFT can be used to generate the spectra. Alternatively, the data can be left as-is, with Hanning weighting provided across only the valid lag channels. This means that the data processing chain will need to be capable of tracking both the number of samples in the spectra, and the number of valid lags in the lag domain. If support for this construct is not desired, then the Linux host can inverse FFT data at the 500ms rate, and re-FFT the data using a non-radix-2 FFT to produce the number of spectral channels indicated in Table 1. Regardless of the method chosen for the higher-level data, the DSP will always use a radix-2 FFT and pad data if required.

For the correlator boards processing 10 baselines, and at most 128-lags, the worst-case FFT total time is $147\mu$s×10 = 1.47ms, i.e., FFT processing of the 1ms phase switched data is not possible.
Table 2: COBRA DSP FFT benchmarks.

<table>
<thead>
<tr>
<th>Length</th>
<th>Time</th>
<th>FFT</th>
<th>twiddles</th>
<th>Setup</th>
<th>cache</th>
<th>optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>47us</td>
<td>RAM0</td>
<td>RAM0 or 1</td>
<td>RAM1  on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>57us</td>
<td>RAM0</td>
<td>RAM0</td>
<td>RAM0  on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>76us</td>
<td>SRAM</td>
<td>RAM0</td>
<td>RAM0  on</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>86us</td>
<td>SRAM</td>
<td>RAM0</td>
<td>RAM0  off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>102us</td>
<td>RAM0</td>
<td>RAM0 or 1</td>
<td>SRAM on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>149us</td>
<td>SRAM</td>
<td>SRAM</td>
<td>SRAM on or off</td>
<td>full</td>
<td></td>
</tr>
<tr>
<td></td>
<td>167us</td>
<td>SRAM</td>
<td>SRAM</td>
<td>SRAM on or off</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>77us</td>
<td>RAM0</td>
<td>RAM0 or 1</td>
<td>RAM1  on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>98us</td>
<td>RAM0</td>
<td>RAM0</td>
<td>RAM0  on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>110us</td>
<td>SRAM</td>
<td>RAM0</td>
<td>RAM0  on</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>145us</td>
<td>SRAM</td>
<td>RAM0</td>
<td>RAM0  off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>206us</td>
<td>RAM0</td>
<td>RAM0 or 1</td>
<td>SRAM on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>265us</td>
<td>SRAM</td>
<td>SRAM</td>
<td>SRAM on</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>330us</td>
<td>SRAM</td>
<td>SRAM</td>
<td>SRAM off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>147us</td>
<td>RAM0</td>
<td>RAM0 or 1</td>
<td>RAM1  on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>200us</td>
<td>RAM0</td>
<td>RAM0</td>
<td>RAM0  on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>185us</td>
<td>SRAM</td>
<td>RAM0</td>
<td>RAM0  on</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>284us</td>
<td>SRAM</td>
<td>RAM0</td>
<td>RAM0  off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>458us</td>
<td>RAM0</td>
<td>RAM0 or 1</td>
<td>SRAM on or off</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>541us</td>
<td>SRAM</td>
<td>SRAM</td>
<td>SRAM on</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>736us</td>
<td>SRAM</td>
<td>SRAM</td>
<td>SRAM off</td>
<td>any</td>
<td></td>
</tr>
</tbody>
</table>
4 Memory transfer benchmarking

Data movement is a significant component of the data processing overhead. If the DSP is to handle the 1ms data from the FPGAs, then there will be worst-case transfers of 1280-words (128 lags, 10 baselines) from the FPGAs required every 1ms. This data would need to be accumulated (integer format would be ok) for 16ms (requiring transfers on and off-chip), FFTed, delay/lobe corrected, and accumulated into 16ms packets. Then every 500ms transfers to SDRAM are needed. The 1ms data transfers can be removed from the DSP overhead by phase switch demodulating in the FPGAs, leaving the DSP with some 1ms FPGA housekeeping chores, 16ms data processing, and the 500ms data processing.

Table 3 provides benchmarking measurements of memory transfers (with the cache always on). The text and data sections of the compiled C source were placed into external SRAM, as this is representative of the COBRA DSP code. The main loop of the C coded memory transfer was 1:

```c
p = src_buffer;
q = dst_buffer;
for (i = 0; i < BUFFER_SIZE; i++) {
    *q++ = *p++;
}
```

The buffer transfers were performed between DSP internal RAM (RAM0 and RAM1), external SRAM, FPGA RAM, and SDRAM. Table 3 shows that the compiler settings can cause a significant difference in the measured timing. The fastest memory transfers occur when the assembler instruction `RPTB` (repeat block) is used (‘Inline options’ set to full; -x2 or ‘Optimization Level’ set to 2 or 3; -o2 or -o3). However, this instruction causes the DSP to ignore external interrupts, so has the same effect as disabling interrupts. This is an important point to consider if for example the memory move operation takes more than 1ms, and a critical FPGA interrupt is missed.

Table 3 can be used to estimate the time taken for the memory transfer component of processing. As an example, consider if the DSP was required to process the 1ms phase switch data. Worst-case DSP processing of 1ms phase switch data; 1280-samples of FPGA result, 1280-samples of accumulated results. At a minimum both have to be moved onto the DSP for accumulation and the sum moved back to SRAM. Hence there is a 1280-sample FPGA-to-DSP RAM move, a 1280-sample SRAM-to-DSP RAM move, and a 1280-sample DSP RAM-to-SRAM move, i.e., \((195\mu s + 100\mu s + 100\mu s) \times 1280 / 512 = 988\mu s\). That is, the DSP will be spending 98.8% of its time moving data! Clearly 1ms phase switching must be handled at the FPGAs.

---

1 A similarly coded while-loop gave slightly lower performance.
Table 3: COBRA DSP memory transfer benchmarks.

<table>
<thead>
<tr>
<th>Buffer size</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Optimization</th>
<th>Inline options</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>68µs</td>
<td>RAM0</td>
<td>RAM1</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>68µs</td>
<td>RAM0</td>
<td>RAM1</td>
<td>Level 2 or 3</td>
<td>disable</td>
<td></td>
</tr>
<tr>
<td>255µs</td>
<td>RAM0</td>
<td>RAM1</td>
<td>Level 0 or 1</td>
<td>disable</td>
<td></td>
</tr>
<tr>
<td>850µs</td>
<td>RAM0</td>
<td>RAM1</td>
<td>disable</td>
<td>disable</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>68µs</td>
<td>RAM0</td>
<td>RAM0</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>512</td>
<td>100µs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>100µs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>Level 2 or 3</td>
<td>disable</td>
<td></td>
</tr>
<tr>
<td>256µs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>Level 0 or 1</td>
<td>disable</td>
<td></td>
</tr>
<tr>
<td>945µs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>disable</td>
<td>disable</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>100µs</td>
<td>SRAM</td>
<td>RAM0</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>1024</td>
<td>195µs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>512</td>
<td>163µs</td>
<td>RAM0</td>
<td>FPGA RAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>195µs</td>
<td>FPGA RAM</td>
<td>RAM0</td>
<td>any</td>
<td>full</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>288µs</td>
<td>SRAM</td>
<td>FPGA RAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>318µs</td>
<td>FPGA RAM</td>
<td>SRAM</td>
<td>any</td>
<td>full</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>195µs</td>
<td>RAM0</td>
<td>SDRAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>286µs</td>
<td>SDRAM</td>
<td>RAM0</td>
<td>any</td>
<td>full</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>318µs</td>
<td>SRAM</td>
<td>SDRAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>410µs</td>
<td>SDRAM</td>
<td>SRAM</td>
<td>any</td>
<td>full</td>
<td></td>
</tr>
</tbody>
</table>
5 Data accumulation benchmarking

Table 4 shows the benchmark results for accumulation of real-valued floating-point or integer data sets (with the cache always on). The text and data sections of the compiled C source were placed into external SRAM, as this is representative of the COBRA DSP code. Accumulation of complex data is identical to two accumulations of real-valued data (one for the real-part one for the imaginary part). In Table 4 both the source and destination arrays are input data vectors, and the destination array is where data is accumulated. The main loop of the C coded memory transfer was:

```c
p = src_buffer;
q = dst_buffer;
for (i = 0; i < BUFFER_SIZE; i++) {
    *q++ += *p++;
}
```

The tests were performed using both integer and floating-point data types—identical benchmark results were obtained for both data type. The benchmarks that have the highest performance compile their loops to the RPTB assembly instruction.

The accumulation tests provide an answer to the following question; is it better to move data from the FPGAs onto DSP SRAM and then accumulate, or should the data be accumulated from the FPGA RAM directly? Assuming for the moment that we are reading and accumulating integer data, Table 3 shows that moving 512 samples from FPGA RAM to RAM0 takes 195\(\mu\)s, while Table 4 shows that the accumulation of two buffers in RAM0 takes 100\(\mu\)s, giving a total of 295\(\mu\)s. Table 4 also shows that accumulation of 512 samples from FPGA RAM into a RAM0 buffer takes 347\(\mu\)s. Hence it is better to move the data into DSP on-chip SRAM then add, rather than add directly from the FPGA memory.
Table 4: COBRA DSP integer or floating-point accumulation benchmarks.

<table>
<thead>
<tr>
<th>Buffer size</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Optimization</th>
<th>Inline options</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>100μs</td>
<td>RAM0</td>
<td>RAM1</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>100μs</td>
<td>RAM0</td>
<td>RAM1</td>
<td>Level 2 or 3</td>
<td>disable</td>
</tr>
<tr>
<td></td>
<td>255μs</td>
<td>RAM0</td>
<td>RAM1</td>
<td>Level 0 or 1</td>
<td>disable</td>
</tr>
<tr>
<td></td>
<td>786μs</td>
<td>RAM0</td>
<td>RAM1</td>
<td>disable</td>
<td>disable</td>
</tr>
<tr>
<td>512</td>
<td>100μs</td>
<td>RAM0</td>
<td>RAM0</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>512</td>
<td>256μs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>256μs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>Level 2 or 3</td>
<td>disable</td>
</tr>
<tr>
<td></td>
<td>317μs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>Level 0 or 1</td>
<td>disable</td>
</tr>
<tr>
<td></td>
<td>1000μs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>disable</td>
<td>disable</td>
</tr>
<tr>
<td>512</td>
<td>160μs</td>
<td>SRAM</td>
<td>RAM0</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>160μs</td>
<td>SRAM</td>
<td>RAM0</td>
<td>Level 2 or 3</td>
<td>disable</td>
</tr>
<tr>
<td></td>
<td>287μs</td>
<td>SRAM</td>
<td>RAM0</td>
<td>Level 0 or 1</td>
<td>disable</td>
</tr>
<tr>
<td></td>
<td>913μs</td>
<td>SRAM</td>
<td>RAM0</td>
<td>disable</td>
<td>disable</td>
</tr>
<tr>
<td>1024</td>
<td>500μs</td>
<td>RAM0</td>
<td>SRAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>318μs</td>
<td>SRAM</td>
<td>RAM0</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>512</td>
<td>412μs</td>
<td>RAM0</td>
<td>FPGA RAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>347μs</td>
<td>FPGA RAM</td>
<td>RAM0</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td>512</td>
<td>504μs</td>
<td>FPGA RAM</td>
<td>FGPA RAM</td>
<td>any</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>538μs</td>
<td>FPGA RAM</td>
<td>SRAM</td>
<td>any</td>
<td>full</td>
</tr>
</tbody>
</table>
6 Floating point conversion benchmarking

DSP integer-to-floating point conversion benchmark tests gave the same results as the data movement tests shown in Table 3. This means that the loop

```c
p = src_buffer;
q = dst_buffer;
for (i = 0; i < BUFFER_SIZE; i++) {
    *q++ = (float)*p++;
}
```

where the source buffer is integer, and the destination buffer is float (and they can be the same buffer) imposes no more additional overhead than a data movement (i.e., the floating point conversion adds no additional overhead).

DSP floating point-to-IEEE 754 floating-point conversion is implemented most efficiently using a C-callable assembly language routine. Benchmark tests were performed with the assembly routine text and data sections in on-chip SRAM, and the C code text and data in SRAM, full optimization, and the cache on. Conversion tests with buffers on-chip and in SRAM were performed, both took 1.85ms to convert 512-samples from TI format to IEEE format. The fact that the location of the buffers did not affect the total conversion time, means that the conversion process is dominated by DSP processing cycles. For conversion of the 500ms data, the worst case number of floating point samples is 10 baselines, 2 phase bins, and 128-lags per baseline (which FFTs to 65 complex channels), so the IEEE conversion will take $2600 \times 1.85ms/512 = 9.4ms$, i.e., about 2% of 500ms. However, this 9.4ms of conversion has to occur while the 16ms data processing is also going on. In comparison to the other benchmark operations, IEEE floating point conversion is an expensive operation that might be best performed by the host once it has received the 500ms data.
7 Multiplication benchmarking

The COBRA correlator FPGAs implement a 2-bit correlation with deleted inner product multiplication scheme. The data read from the correlator FPGAs is in integer format and must be converted to floating-point and then normalized before being passed to the FFT routine. After FFT, the new complex-valued data requires compensation for sub-1ns delays and lobe rotation. Normalization of the real-valued data requires multiplication, and delay compensation of the complex data requires complex-valued multiplication.

The following code provides an estimate of the time required for real-valued multiplications:

```c
p = src_buffer;
q = dst_buffer;
for (i = 0; i < BUFFER_SIZE; i++) {
    *q++ *= *p++;
}
```

With compiler optimization on full, floating point multiplication of 512 samples takes 102 µs (the loop compiles to a repeat-block, `rptb`, containing a floating-point multiply instruction, `mpyf3`). Integer multiplication takes a massive 723 µs! Examining the source code shows that the compiler generates a call to `MPY_I30`. Turning on the compiler code generation option to use the short multiply instruction replaces `MPY_I30` with the `mpyi3` instruction which causes the integer loop time to reduce to that of the floating-point loop. The short multiply operation assumes 24-bit signed integer types, while the `MPY_I30` call must implement a full 32-bit integer multiply.

The DSP delay compensation and lobe rotation compensation operations are applied in the frequency domain to complex valued, floating-point data sets. The complex multiplication of a sample \((a + jb)\) with another \((c + jd)\) is

\[
(a + jb) \times (c + jd) = (ac - bd) + j(ad + bc),
\]

i.e., the real and the imaginary part of the output complex sample require two multiplies and an add, for a total of four multiplies and two adds per complex sample. If the \(a\) and \(b\) data is also the destination location of the complex multiplication, then a temporary variable is required to store the original value of \(a\) so that it can be used in the calculation of the new value for \(b\), for example

```c
float tmp;
float *p = src_buffer;
float *q = dst_buffer;
for (i = 0; i < BUFFER_SIZE; i++) {
    tmp = q[0];
    /* Real */
    q[0] = tmp*p[0]+q[1]*p[1];
    /* Imag */
    q[1] = tmp*p[1]+q[1]*p[0];
    q += 2;
    p += 2;
}
```

where for each input vector, the real parts are at even indexes in the vector, and the imaginary parts are at odd indexes. For 512 complex samples, this loop with optimization on requires 384 µs. From the previous benchmarks, the multiplication of a single vector requires 100 µs (and the complex loop has four), and the accumulation operation also requires 100 µs (and the loop has two), indicating a potential total processing time of 600 µs. Examination of the assembler output shows that parallel instructions are not being used, so it is likely that the 384 µs benchmark can be reduced further.
8 Delay/lobe rotation calculation benchmarking

Propagation delay differences between antennas results in a phase slope across the cross-correlation estimates if left uncorrected. Propagation delay differences also occur in the IF path, eg. in the lengths of fiber used to route the signals to the correlator. The form of the delay component in the cross-correlation result is of the form:

\[ D(f; f_0, \tau) = \exp \{ j2\pi (f_0 + f)\tau \} = \cos \{ 2\pi (f_0 + f)\tau \} + j \sin \{ 2\pi (f_0 + f)\tau \} \] (2)

where \( f_0 \) is a frequency constant related to lobe rotation correction, and \( \tau \) is the required delay slope correction. The digitizer boards will implement delays quantized to 1ns (the digitizer sample period), so the correlation phase seen on the correlator boards after digitizer board digital delay correction by \( \tau_1 \) is

\[ D_{\text{corl}}(f; f_0, \tau, \tau_1) = \exp \{ j2\pi [(f_0 + f)\tau - f\tau_1] \} \]

\[ = \cos \{ 2\pi [(f_0 + f)\tau - f\tau_1] \} + j \sin \{ 2\pi [(f_0 + f)\tau - f\tau_1] \} \]

\[ = \cos \{ 2\pi [f_0\tau + f(\tau - \tau_1)] \} + j \sin \{ 2\pi [f_0\tau + f(\tau - \tau_1)] \} \] (3)

where the (constant) terms in \( f_0\tau \) need to be cancelled as part of the lobe rotation correction, and the terms in \( f\Delta\tau \) are the delay slope residual (sub-1ns) correction. The phase terms corrected for after correlation, i.e., \( \phi(f; f_0, \tau, \Delta\tau) \) require that the correlator boards know the frequency channel spacings \( f \), the frequency constant \( f_0 \), the geometric delay \( \tau \), and the delay residual \( \Delta\tau \).

The calculation time for the sine and cosine components are non-trivial, so a common technique for reducing this calculation overhead is to pre-calculate a quadrant of a sine (or cosine) and then use that as a lookup table. For an FFT routine, the quantization of this table is fixed by the number of samples in the FFT. For delay and lobe rotation correction the term \( \phi \) is essentially continuous, so some choice on acceptable quantization has to be made. Assuming we would like the finest delay correction to be 1/64th of the sample period, then for example, for the 500MHz band (1GHz sample clock, 1ns sample period), the delay correction can be quantized to 1ns/64 = 15.625ps.

The COBRA Wideband correlator produces 64-lags for the 500MHz band, so the frequency channel spacing is 15.625MHz, while for CARMA and SZA there are 32-lags and 31.25MHz channel spacing. The phase quantization is \( \Delta\phi = 2\pi \Delta f \Delta t \), so the number of phase samples required in a quadrant for COBRA Wideband would be \( 2\pi / (4\Delta\phi) = 1/(4\Delta f \Delta t) = 1/(4 \times 15.625MHz \times 15.625\text{ps}) = 1024 \) and 512 for CARMA/SZA. (Note that there is plenty of memory on the COBRA boards, so storage of the full sine lookup table is no problem).

Based on a 1/64th sampling period delay correction, any band with a clock period longer than 64ns (15.625MHz), i.e., the 8MHz and 2MHz bands, can have the delay correction applied using only digitizer delays. However, the lobe rotation correction still needs to be applied at the correlator board after cross-correlation (every 16ns).

---

\( ^2 \)This is not quite the correct equation for compensation in a double sideband system, but it is adequate for determining a benchmark test
Out of morbid curiosity, the following loop was executed for one 65 complex sample spectra;

```c
p = buffer;
for (i = 0; i < Nchannels; i++) {
    phi = dphi*i;
    /* Real */
    p[0] = cos(phi);
    /* Imag */
    p[1] = sin(phi);
    p += 2;
}
```

evaluation time was 1.13ms, i.e., 10 baselines would take 11.3ms to calculate!

By first generating a lookup table of quantized sinusoidal components, the loop can be reduced to an index generation and lookup operation. In the loop below, a full 360-degree table was generated, so the indexes in the table only need to be generated modulo the table length. If a single quadrant table is used, then the indexing scheme gets slightly more complex, and the sign of the phase read from the quadrant also has to be determined. A 360-degree lookup table was used so that a reasonable benchmark number could be determined. Re-coding this lookup using assembler and the specialized DSP addressing modes would likely offer improvement over this simple test. For a single correlation spectrum of 65 complex samples, the loop required 144µs, so 10 baselines requires 1.44ms (not much different than the assembly coded FFT!).

```c
p = buffer;
for (i = 0; i < Nchannels; i++) {
    /* Table index */
    sin_idx = ((int)(dindex*(float)i))%TABLE_LENGTH;
    cos_idx = (sin_idx+quarter_table)%TABLE_LENGTH;

    /* Real */
    p[0] = sine[cos_idx]; /* cos */
    /* Imag */
    p[1] = sine[sin_idx]; /* sine */
    p += 2;
}```
9 Data processing estimates

The memory benchmarking section shows that the DSP memory bandwidth is insufficient for dealing with the 1ms phase switched data. Assuming then that the 1ms phase switch demodulation is performed by the FPGAs, the DSP needs to process 16ms data. The estimate of processing steps and time are:

- Transfer of 1 phase bin, 10 baselines, 128-lags per baseline from FPGA memory to DSP memory; 1280-words is too large to fit into one of the DSP 1K RAM blocks, so the data will have to be transferred in smaller blocks, eg. 128-words per FPGA. There are two possible options for moving each data block; FPGA RAM-to-SRAM or FPGA RAM-to-DSP RAM then DSP-RAM to SRAM. Both methods appear to have comparable expense, so assuming the data is transferred from FPGA-to-SRAM, the time required is $1280 \times 318 \mu s / 512 = 795 \mu s$.

**NOTE:** The planned changes to the correlator FPGAs will leave the 1ms data accumulating in the internal lag counters, this means that the last 16ms integration will be valid in FPGA RAM for the full 16ms.

- DSP integer to floating point conversion of the latest 16ms integration; integer to floating point conversions are as costly as a data move operation, so the conversion can take place as part of the movement of data from the FPGAs to SRAM. However, this requires that the DSP perform the transfer, not the DMA controller. No cost.

  If the FPGA to SRAM transfers are implemented by a new system controller DMA controller, then the integer to floating point conversion can be performed later when the DSP moves the data from SRAM to on-chip RAM for processing. Again, since this move operation is required, there is no cost associated with the integer to float point conversion. If the DSP DMA controller is used to move the data into on-chip SRAM, then the DSP will have to perform a floating-point conversion before passing the data to the FFT (this reduces the usefulness of using the DMA controller for that particular transfer).

- Normalization of the 16ms data (assuming it is dominated by multiplication); $1280 \times 102 \mu s / 512 = 255 \mu s$.

- FFTing the latest 16ms data; $10 \times 147 \mu s = 1.47ms$.

  **Note:** 128-lags will FFT to 65-complex samples (the first and last samples are real-valued, but the processing chain will not account for that, hence the data load is 650-complex samples or 1300-words).

- Repacking the FFT output format into a sequence of (NLAGS/2+1) complex samples (this is treated as a DSP SRAM-to-DSP SRAM move); $1300 \times 68 \mu s / 512 = 173 \mu s$.

- Delay and lobe rotation vector generation; $10 \times 144 \mu s = 1.44ms$.

- Delay and lobe rotation correction; $650 \times 384 \mu s / 512 = 488 \mu s$.

- Accumulation of the current 16ms integration into the 500ms integration; 1300-words of the current integration have to be accumulated with the appropriate 1300-words of the 500ms data. Both of these data sets will need to be moved into the DSP SRAM from external SRAM, accumulated, and then moved back to external SRAM. The worst-case total time for this operation will be on the order of a SRAM-to-DSP RAM transfer, accumulation, and DSP RAM-to-SRAM transfer, i.e., $1300 \times (100 \mu s + 100 \mu s + 100 \mu s) / 512 = 762 \mu s$.

  Through careful choice of buffering in the two DSP SRAM blocks, some of the transfers of data blocks may be able to be overlapped using the DSP DMA controller to transfer data into DSP SRAM while the DSP is processing the current data block which is also in internal SRAM.
• Total processing time for the 16ms data;
  \[795\mu s + 255\mu s + 1.47ms + 173\mu s + 1.44ms + 488\mu s + 762\mu s = 5.38ms\ (34\% \ of \ 16ms).\]

• A benchmark routine was tested for a 128-lag data set that underwent the following sequence of operations; FPGA RAM-to-DSP RAM transfer, normalization, FFT, FFT output format repacking, delay vector generation, and then application, and finally accumulation with data in an external SRAM buffer\(^3\). The total processing time for the 128-lag data set was 540\(\mu s\), which implies 10 baselines should take 5.4ms (which nicely matches the preceding estimate!).

• Every 500ms, the two phase bins of data needs to be moved from SRAM over to SDRAM, which will take on the order of \(2 \times 1300 \times 318\mu s/512 = 1.6ms\).

• Optionally, the 500ms data to the host can be converted to the hosts IEEE 754 floating-point format. This conversion takes the DSP on the order of 9.4ms.

## 10 Performance improvement options

The DSP clearly cannot deal with the 1ms phase switched data, so the FPGAs must be modified to handle 180-degree phase switch demodulation, and accumulation of data for \(16 \times 1\ms = 16\ms\).

The DSP has only one external bus, so there are not too many options for improving memory bandwidth performance. One possible option for reducing DSP bus usage is related to the transfer of data from FPGA RAM to SRAM. Transfer of data from FPGA RAM to SRAM has a potential maximum transfer rate of 3 clock cycles per word (the three clock cycles would be needed by a state machine to generate a SRAM write-enable pulse). For example transfer of the 128-lags from a single FPGA for a 16ms integration would require \(128 \times 90\text{ns} = 12\mu s\ (33\text{MHz system clock})\). Table 3 benchmarked that memory transfer at \(318\mu s \times 256/512 = 80\mu s\), i.e., 7 times slower! This is due to the fact that the DSP must perform a read then a write operation for each sample transferred. A custom DMA controller implemented in the system controller FPGA could simultaneously perform the read from the FPGA, and while the data was valid on the bus, the write to SRAM. Transfer of all FPGA data to SRAM could be reduced to the order of \(115\mu s\) total, instead of the benchmark estimate of \(318\mu s \times 1280/512 = 795\mu s\). This \(680\mu s\) saving is \(4.3\%\) of the 16ms processing period.

A caveat regarding the FPGA-to-SRAM DMA controller option is that adding this logic requires modification to the COBRA board system controller FPGA. The system controller FPGA is already densely configured, and may have trouble meeting its timing requirements with this additional logic.

---

\(^3\)Accumulating into the SRAM buffer directly gave the same performance as moving the SRAM data on-chip, then accumulating, then writing the data back to SRAM