Echolocation Methods

High Performance Embedded Computing (HPEC)

Iwona Kochańska
One of the major goals of HPEC: to deliver ever greater levels of functionality to embedded signal and image processing (SIP) applications.

High performance SIP algorithms demand throughputs ranging from hundreds of millions of operations per second (MOPS) to trillions of OPS (TOPS).
HPEC challenges

• HPEC is particularly challenging:
  o high throughout requirements,
  o real-time deadlines
  o form-factor constraints.

• HPEC is a juggling act that must deal with all three challenges at once.
HPEC latencies

- HPEC latencies:
  - milliseconds for high pulse repetition frequency (PRF) tracking radars
  - a few hundred milliseconds in surveillance radars
  - minutes for sonar systems

- The best designs will satisfy both latency and throughput requirements while minimizing:
  - hardware resources,
  - software complexity
  - form factor (HPEC systems must fit into spaces ranging from less than a cubic foot to a few tens of cubic feet, and must operate on power budgets of a few watts to a few kilowatts)

- With these size and power constraints, achievable computational power efficiency, measured in operations per second per unit power (OPS/watt), and computational density, measured in operations per second per unit volume (OPS/cubic foot), determine the overall technology choice.
HPC latencies (2)

• The front-end and back-end latency allowances are determined by the overall latency requirement and must be traded off against each other.

• For example, tracking radars may have to close the tracking loop in a few milliseconds.
  o The amount of time the radar has to detect the target, update its track, generate a new position prediction, and then direct its antenna to point in a new direction (to keep the target in the radar beam) must not exceed a few milliseconds.
  o This update rate is driven by the dynamics of the target being tracked. For a slow-moving target, a few hundred milliseconds or even a few seconds may be appropriate. For a highly maneuverable target such as a fighter aircraft, firecontrol radars may need to operate with millisecond latencies.
HPEC design process

• The design process requires the architect to first decompose the candidate algorithms into constituent stages, exposing:
  o computational parallelism
  o communication patterns
  o key computational kernels

• After decomposition, the algorithm components are mapped to processing hardware, which may be a combination of application-specific circuitry and more general-purpose programmable components.

• The design proceeds iteratively. The algorithm may be modified to:
  o reduce computational complexity
  o increase parallelism
  o accommodate hardware and software options (e.g., a specialized processing chip or an optimized library routine).
HPEC design process (2)

• Application performance must then be reassessed in light of the modified algorithm, so that a balance between performance and complexity can be reached.

• Quite often, different but equivalent algorithm variants—for example, time domain versus frequency domain filter implementations—are possible. Each variant affects:
  o computational complexity
  o communication patterns
  o word length
  o control flow

hence, each has an influence on the computer architecture or, conversely, is more or less suited to a particular architecture.
HPEC canonical architecture

HPEC processing stages

• **front-end** signal and image processing stage - extract information from a large volume of input data (performs stream-based signal and image processing).
  - removal of noise and interference from signals and images
  - detection of targets
  - extraction of feature information from signals and images

• **back-end** data processing stage - further refine the information so that an operator, the system itself, or another system can then act on the information to accomplish a system-level goal (knowledge-based processing).
  - parameter estimation
  - target tracking
  - fusion of multiple features into objects
  - object classification and identification
  - other knowledge-based processing tasks
  - display processing
  - interfacing with other systems
HPEC technology

• The technology choices for front-end processors:
  o full-custom very-large-scale integration (VLSI)
  o application specific integrated circuits (ASICs)
  o field-programmable gate array (FPGAs)
  o programmable digital signal processors (DSPs)
  o microprocessor units (MPUs).
  o hybrid designs that incorporate a combination of these technologies

• The technology chosen for the back-end data processing:
  o programmable multicomputer composed of DSPs or MPUs
  o shared memory multiprocessor

• Typically, front-end processing requires significantly greater computational throughput, whereas back-end processing has greater program complexity.

• Throughput is usually measured in terms of OPS.
  o front-end algorithms can require anywhere from a few billion OPS to as many as a few trillion OPS
  o back-end algorithms tend to require an order of magnitude or two fewer OPS
HPEC data rates

- Front-end computations perform operations that transform raw data into information.

- The analog-to-digital converters (ADCs) are themselves highly sophisticated components that set limits on the precision and bandwidth of the signals that can be digitally processed.

- SIP algorithms remove noise and interference, and extract higher level information, such as target detections or communication symbols, from a complex environment being sensed by a multidimensional signal, image, or communication sensor.

In phased-array radars, for example, signals with data rates in the 100s of millions of samples per second (MSPS) arrive at the front-end of the digital signal processor from tens of receiver channels. This results in an aggregate sample rate of billions of samples per seconds (GSPS)
Data reduction in front-end stages.

ECCM (electronic counter-countermeasures): beamforming operation in which the channels are combined to form a small number of beams.

Detection stage - algorithm rejects noise and identifies the range gates that contain targets. The number of targets is significantly less than the total number of range gates.
Front-end output/input data rate is typically less than $\frac{1}{2} \%$

1. The input is from a 48-channel radar phased array sampled at 480 Msps with 12-bit real data per sample.
2. The detector assumes a maximum of 20,000 targets, with each target report being 256 bytes.

Data types in front-end processing

- sequences of real or complex integer or floating-point numbers
- data aggregated into vectors or arrays
- vectors and arrays have simple indexing computations:
  
a vector reference \( v(i) \) can be computed by simply multiplying the basic element size by \( (i - 1) \) and adding the result to the initial location
Data types in back-end processing

• Complex, composite data structures with numerous fields

• Fields themselves may be vectors, matrices, other composite data items, or pointers to any of these structures.

• In more complex structures, the entries may be pointers to other entries or to other objects outside of the data item.

Example: a track-file object may link each target and track estimate and may also link the time-sequence of targets
Data types in back-end processing

• High-level languages provide powerful referencing semantics and syntax for iterating through both simple stream data items as well as complex data objects. However, if the references are dynamic, meaning that they may depend on data values or program state generated at runtime, the compiler will not be able to precompute a constant offset into the data object, and complex assembly-level code will be required.

Trade off between statically and dynamically allocated data!

• Object accesses in back-end processing may be significantly more time-consuming than data accesses in front-end processing.
Front-end computations

• Most of the computations performed in the front-end require regular and repetitive data-indexing patterns.

• The locations of the data in memory can be determined beforehand and do not depend on the values of the data. This is important for custom hardware implementations that rely on regular, predictable memory accesses to optimize control logic and data flow.

• The operation mix applied at the front-end strongly favors arithmetic operations, such as multiplication, addition, and division; low percentage of logic and data manipulation

• Most high performance front-end algorithms consist of kernel operations from matrix or vector mathematics, with some special computational constructs, such as butterflies that support fast Fourier transform (FFT) computations.
## Front-end computations (2)

<table>
<thead>
<tr>
<th>FE Kernel</th>
<th>Computation Elements</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time domain FIR and IIR filters</td>
<td>( \mathbf{a} = \mathbf{a} + \mathbf{bc} ), also called a multiply-and-accumulate or MAC operation: load registers ( \mathbf{b} ) and ( \mathbf{c} ), perform the multiplication, add the product to the contents of register ( \mathbf{a} ), and accumulate the result back in ( \mathbf{a} )</td>
<td>Time domain convolution, correlation, time domain frequency filtering (high, low, bandpass)</td>
</tr>
<tr>
<td>Fourier transforms (FFT &amp; IFFT)</td>
<td>butterfly computations</td>
<td>Frequency domain filters and spectral analysis; matched filtration</td>
</tr>
<tr>
<td>Vector and matrix multiplications, inner products;</td>
<td>Operation ( \text{axpy: } \mathbf{y} = \alpha \mathbf{x} + \mathbf{y}, \mathbf{x} - \text{vectors, } \alpha - \text{scalar. Dot product } \mathbf{c} = \mathbf{y}^H \mathbf{x} . \text{Can be accomplished with (C)MAC operations. Vector machines can load } \mathbf{y} \text{ and } \mathbf{x} \text{ into vector registers, each with a single instruction} )</td>
<td>Numerous signal and image processing stages; in particular, beamforming for radar and sonar</td>
</tr>
<tr>
<td>Matrix QR or LQ decomposition</td>
<td>succession of matrix-matrix multiplications. The Householder and Givens algorithms compute reflections and rotation operation matrices. Inner products and divisions are needed. The Givens rotation requires a square root computation</td>
<td>many adaptive weight calculations, in which they are used to factor a sample matrix as part of the Weiner-Hopf optimization equation</td>
</tr>
</tbody>
</table>

Back-end computations

• Processing predominantly manipulates higher-level, composite objects

• Parameter estimation algorithms (such as maximum-likelihood estimation) - vector or matrix mathematics over a grid of estimation points

• Automatic target recognition, discrimination, and object identification:
  o correlations and convolutions
  o threshold computations
  o normalizations and other scaling operations
  o distance computations
Back-end computations (2)

- Back-end operations use dense matrix kernels (as in the front-end). Tracking techniques, such as Kalman filtering, utilize state-space representations that lend themselves to dense matrix mathematics.

- Matrix kernels are applied to many fewer objects than in the case of front-end processing. Thus, the computational load in the back-end is not usually dominated by dense matrix mathematics.

- Instead, computations are characterized by:
  - a large percentage of logic and data manipulation (versus arithmetic) operations,
  - highly irregular memory accesses,
  - complex program flow-of-control

- Examples of tasks dominated by these computational elements:
  - scheduling
  - planning
  - user interfaces
  - networked communications
  - databases
  - decision support tasks
Back-end computations (3)

- Many of the back-end tasks employ graph-based algorithms:
  - Markov decision processes
  - Associative databases
  - Decision-tree techniques

- Graph-based algorithms can also be expressed as sparse-matrix algorithms (matrices in which most of the entries are zeros.)

- Front-end processing - dense matrix tasks
- Back-end processing - sparse matrix tasks
Caching the data

• Front-end streams transient data, meaning that they are accessed only a few times before being discarded
  o typically, the stream data are transformed to new data items, and the original data are no longer needed
  o each data item is only reused a few times and, hence, there is little advantage in storing it in cache memory.

• Back-end data objects tend to be persistent, meaning that they last for many program iterations and have internal state that evolves over time.
  Example: a track can exist for seconds or hours. Hence, a back-end program that operates on a set of tracks can benefit greatly from data caching.
Performance of PowerPC Microprocessor Unit (MPU) on the convolution (time domain FIR filter).

### Computational Complexity for Common Signal Processing Kernels

<table>
<thead>
<tr>
<th>Signal Processing Kernel</th>
<th>Computational Complexity Real Input</th>
<th>Computational Complexity Complex Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix-matrix multiplication</td>
<td>$2mnp$</td>
<td>$8mnp$</td>
</tr>
<tr>
<td>Fast Fourier transform</td>
<td>$\frac{5}{2} n \log_2 n$</td>
<td>$5n \log_2 n$</td>
</tr>
<tr>
<td>Householder QR decomposition</td>
<td>$2n^2 \left( m - \frac{n}{3} \right)$</td>
<td>$8n^2 \left( m - \frac{n}{3} \right)$</td>
</tr>
<tr>
<td>Forward or back substitution</td>
<td>$n^2$</td>
<td>$4n^2$</td>
</tr>
<tr>
<td>Eigenvalue decomposition: eigenvalues only</td>
<td>$\frac{4}{3} n^3$</td>
<td>$\frac{16}{3} n^3$</td>
</tr>
<tr>
<td>Eigenvalue decomposition: eigenvalues and eigenvectors</td>
<td>$9n^3$</td>
<td>$23n^3$</td>
</tr>
<tr>
<td>Singular-value decomposition: singular values only</td>
<td>$4mn^2 - \frac{4}{3} n^3$</td>
<td>$16mn^2 - \frac{16}{3} n^3$</td>
</tr>
</tbody>
</table>

Parallelism

- High performance embedded computing algorithms contain computations that can take place concurrently. Computer architectures attempt to exploit this concurrency with hardware execution units that operate in parallel.

- Parallelism is exploited at all levels of granularity, from bit-level parallelism within basic operations to entire program replication.

- The basic trade-off is the high degree of computational efficiency and power density that custom implementations can deliver.

- Detailed understanding of algorithm structure and parallelism is required to develop effective implementations.

- **Data parallelism** - object is decomposed into subobjects, each of which is similarly operated on by a computation. An algorithm contains data parallelism if the same operation can be applied concurrently over a set of data objects.
Levels of parallelism

- **Level 5**: Jobs or Programs (Coarse Grain)
- **Level 4**: Subprograms, Job Steps, or Related Parts of a Program (Medium Grain)
- **Level 3**: Procedures, Subroutines, Tasks, or Co-routines
- **Level 2**: Nonrecursive Loops or Unfolded Iterations (Fine Grain)
- **Level 1**: Instructions or Statements

Matrix addition
- „embarrassingly” parallel

- For example: the matrix operation $A = B + C$;

- The degree of parallelism (DoP) is $3 \times 4 = 12$
- In this example, the operation can be applied independently to the individual data elements
  Embarrassingly parallel!
- Other partitionings possible, as shown below

- DoP for column vector partitioning is $1 \times 4 = 4$

Matrix multiplication

Parallel, matrix multiplication is an example of a data parallel algorithm that requires *synchronization* and *communication* between the parallel units. The partitioning of the algorithm and mapping of the data attempt to balance processor load with communication overhead.

Array partitioning

- There are a few standard ways to partition arrays and distribute them over compute nodes to accommodate data-parallel algorithms.

- Partitioning the data into blocks and to assign each block to a separate compute node. This works well when the amount of work to be performed is proportional to the amount of data on the node.

- Distributing the data cyclically, so that every nth column is distributed to the same node. A cyclic partitioning more evenly distributes the work for operations such as parallel matrix factorizations (for example, QR, and LU) in which the amount of work depends on the amount of data on the node and the location of the data within the matrix.

- General block-cyclic arrangement of data: blocks of size m are distributed cyclically so that every nth block is assigned to the same node.
  - Sometimes it is important to share data “at the edges” of each partition, in which case a block (or block-cyclic) partitioning with data overlap is used.
Common partition strategies

Data reorganization

• As the overall computation proceeds it may be necessary to redistribute the data to achieve the best parallelism for the next computation.

• Important data reorganizations in HPEC parallel processors:
  o **broadcast** operation - the same data are sent to all processors
  o **gather** operation - data from a set of processors are gathered onto a node
  o **scatter** operation - a set of data are distributed onto a set of processors. Computations may be carried out in concert with communication
  o **reduction** operation - takes a set of data, combines the data according to a defined operation (for example, addition), and stores the result on a processor
  o **spread** (or expansion) operation performs the reduction, takes the result of the reduction, and sends it to a set of processors.
Important data reorganizations in HPEC parallel processors

Round-robin parallelism

Throughput = 4 × throughput of single stage
Latency = 4 × dataset interval

- Increased throughput:
  \[ R_p = N \cdot R_0 \]
- Increased latency:
  \[ T_p = N \cdot T_D \]

- Once pipeline is full, a processed dataset is produced each dataset interval
- Each stage must be prepared to accept data at \( 1/N \) rate of input, where \( N \) is the depth of the round-robin

Pipeline scheduling

- Throughput of pipeline = 4 × throughput of single stage
- Latency of pipeline = 4 × latency of single stage

- A processed dataset is produced each dataset input interval (once pipeline is full)

- Increased throughput:
  \[ R_p = \text{SUM}(R_j), \quad j = 1, \ldots, N \]

- Increased latency:
  \[ T_p = \text{SUM}(T_j), \quad j = 1, \ldots, N \]

Parallelism metrics

- **Speedup** for a parallel algorithm distributed over N processors is

  \[ S_N = \frac{T_S}{T_P} , \]

  \( T_S \) - execution time of the serial algorithm

  \( T_P \) - execution time of the parallel algorithm.

  Ideally, the speedup achieved is equal to the number of processors.

- **Parallel efficiency** for an algorithm distributed over N processors is defined as

  \[ E_N = \frac{S_N}{N} \]

  The ideal efficiency is 100% and the ideal speedup is N.

- Speedup is usually less since the parallel components need to spend time to communicate data and synchronize between themselves. This time is **overhead** that cannot be used to do computation. As the algorithm is distributed over more and more nodes, the overhead may exceed the benefit of using the additional computing power and the speedup may begin to decrease.
Speedup and efficiency

-效率随节点数增加而下降
-最终的超线程速度限制

Mapping of the radar signal processing chain onto the 1000-processor massively parallel signal processor.

Surface moving-target indication (SMTI) survivallence radar example

- SMTI radars are used to detect and track targets moving on the earth’s surface

- Radar signal consisting of a series of pulses from a coherent processing interval (CPI) is transmitted

- The pulse repetition interval (PRI) determines the time interval between transmitted pulses. Multiple pulses are transmitted to permit moving-target detection

- The pulsed signals reflect off targets, the earth’s surface (water and land), and man-made structures such as buildings, bridges, etc.; a fraction of reflected energy is received by the radar antenna

- The goal of the SMTI radar is to process the received signals to detect targets (and estimate their positions, range rates, and other parameters) while rejecting clutter returns and noise. The radar must also mitigate interference from unintentional sources such as RF systems transmitting in the same band and from jammers that may be intentionally trying to mask targets
(SMTI) surveillance radar

SMTI analog processing

- The radar antenna typically consists of a two-dimensional array of elements (1000s of elements)

- The signals from these elements are combined in a set of analog beamformers to produce subarray receive channels, thereby reducing the number of signals that need to be converted to the digital domain for subsequent processing. For example:
  - 20 vertical subarrays are created that span the horizontal axis of the antenna system.
  - Employed in an airborne platform, the elevation dimension is covered by the subarray analog beamformers, and the azimuthal dimension is covered by digital beamformers

- The channel signals subsequently proceed through a set of analog receivers that perform downconversion and band-pass filtering

- The signals are then digitized by analog-to-digital converters (ADCs) and input to the high performance digital front-end
SMTI digital processing

- Channelizer process divides the wideband signal into narrower frequency subbands
- Filtering and beamformer front-end mitigates jamming and clutter interference, and localizes return signals into range, Doppler, and azimuth bins
- Constant-false-alarm-rate (CFAR) detector (after the subbands have been recombined)
- Post-processing stage that performs such tasks as target tracking and classification
SMTI radar

• SMTI radars can require over one trillion operations per second (TOPS) of computation for wideband systems

• The adaptive beamforming performed in SMTI radars is one of the major computational complexity drivers

• The principal challenge in the airborne front-end processors is to provide extremely high performance that can fit into a highly constrained space, operate using low power, and be air-vehicle qualified

• The ADCs must operate at a sufficiently fast sampling rate to preserve the range resolution provided by the waveform
SMTI radar example

• Example: for achieving about one-meter range resolution on transmit, a 180 MHz linear FM waveform is used.
• The ADCs sample at 480 Msps, which amounts to oversampling of the signal by a factor of 4/3 over the Nyquist rate.

Key Operational Parameters for the SMTI Radar

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nch</td>
<td>20</td>
</tr>
<tr>
<td>f_samp</td>
<td>480,000,000</td>
</tr>
<tr>
<td>dec_ratio</td>
<td>96</td>
</tr>
<tr>
<td>Nsubband</td>
<td>48</td>
</tr>
<tr>
<td>Nppf</td>
<td>128</td>
</tr>
<tr>
<td>Nppf_taps_dn</td>
<td>12</td>
</tr>
<tr>
<td>PRF</td>
<td>2,000</td>
</tr>
<tr>
<td>Npri</td>
<td>200</td>
</tr>
</tbody>
</table>

SMTI Radar Parameters

Number of channels
Sampling frequency (Hz)
Decimation ratio
Number of subbands
Number of polyphase filters
Number of polyphase filter taps (analysis)
PRF (Hz)
Number of PRIs per CPI

SMTI processing chain

• The full SMTI processing chain consists of nine stages:
  1. subband analysis
  2. time delay and equalization
  3. adaptive beamforming
  4. pulse compression
  5. Doppler filtering
  6. space-time adaptive processing (STAP)
  7. subband synthesis (recombining)
  8. detection
  9. estimation
### SMTI throughput requirements

<table>
<thead>
<tr>
<th>Stage</th>
<th>Fixed Point (GOPS)</th>
<th>Floating Point (GFLOPS)</th>
<th>Aggregate (GOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subband Analysis</td>
<td>478</td>
<td></td>
<td>478</td>
</tr>
<tr>
<td>Time Delay &amp; EQ</td>
<td>[up to 1000]</td>
<td></td>
<td>478</td>
</tr>
<tr>
<td>Adaptive Beamforming</td>
<td>139</td>
<td>0.20</td>
<td>617</td>
</tr>
<tr>
<td>Pulse Compression</td>
<td>198</td>
<td></td>
<td>816</td>
</tr>
<tr>
<td>Doppler Filtering</td>
<td>66</td>
<td></td>
<td>881</td>
</tr>
<tr>
<td>STAP</td>
<td>41</td>
<td>2.36</td>
<td>925</td>
</tr>
<tr>
<td>Subband Synthesis</td>
<td>76</td>
<td></td>
<td>1,001</td>
</tr>
<tr>
<td>Detection</td>
<td>0.00</td>
<td>5.37</td>
<td>1,007</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>999</strong></td>
<td><strong>7.93</strong></td>
<td><strong>1,007</strong></td>
</tr>
</tbody>
</table>

## SMTI memory requirements

<table>
<thead>
<tr>
<th>SMTI Memory Requirements (Mbytes)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Subband Analysis</td>
<td>0</td>
</tr>
<tr>
<td>Adaptive Beamforming</td>
<td>5,218</td>
</tr>
<tr>
<td>Pulse Compression</td>
<td>346</td>
</tr>
<tr>
<td>Doppler Filtering</td>
<td>346</td>
</tr>
<tr>
<td>STAP</td>
<td>889</td>
</tr>
<tr>
<td>Subband Synthesis</td>
<td>0</td>
</tr>
<tr>
<td>Detection</td>
<td>240</td>
</tr>
<tr>
<td>Total</td>
<td>7,038</td>
</tr>
</tbody>
</table>

SMTI communication requirements

<table>
<thead>
<tr>
<th>Stage</th>
<th>Input (Gbytes/s)</th>
<th>Input (Gbits/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subband Analysis</td>
<td>12.96</td>
<td>103.7</td>
</tr>
<tr>
<td>Adaptive Beamforming</td>
<td>17.28</td>
<td>138.2</td>
</tr>
<tr>
<td>Pulse Compression</td>
<td>3.46</td>
<td>27.6</td>
</tr>
<tr>
<td>Doppler Filtering</td>
<td>3.46</td>
<td>27.6</td>
</tr>
<tr>
<td>STAP</td>
<td>6.88</td>
<td>55.0</td>
</tr>
<tr>
<td>Subband Synthesis</td>
<td>2.58</td>
<td>20.6</td>
</tr>
<tr>
<td>Detection</td>
<td>1.93</td>
<td>15.5</td>
</tr>
<tr>
<td>Downlink</td>
<td>0.05</td>
<td>0.4</td>
</tr>
</tbody>
</table>

## Power requirements

<table>
<thead>
<tr>
<th>Stage</th>
<th>Throughput (GOPS)</th>
<th>Units</th>
<th>Total (GOPS)</th>
<th>Power (Watts)</th>
<th>Memory (Mbytes)</th>
<th>Memory Power (Watts)</th>
<th>Total Power (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis-PPF</td>
<td>28</td>
<td>10</td>
<td>280</td>
<td>6</td>
<td>–</td>
<td>–</td>
<td>6</td>
</tr>
<tr>
<td>Analysis-FFT</td>
<td>20</td>
<td>10</td>
<td>200</td>
<td>5</td>
<td>–</td>
<td>–</td>
<td>5</td>
</tr>
<tr>
<td>ABF</td>
<td>7</td>
<td>20</td>
<td>140</td>
<td>4</td>
<td>5248</td>
<td>82</td>
<td>86</td>
</tr>
<tr>
<td>ABF-SMI*</td>
<td>2.7</td>
<td>1</td>
<td>2.7</td>
<td>5</td>
<td>–</td>
<td>–</td>
<td>5</td>
</tr>
<tr>
<td>PC</td>
<td>4.2</td>
<td>48</td>
<td>202</td>
<td>5</td>
<td>384</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>DF</td>
<td>1.4</td>
<td>48</td>
<td>67</td>
<td>2</td>
<td>364</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>STAP</td>
<td>4.2</td>
<td>10</td>
<td>42</td>
<td>1</td>
<td>880</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>STAP-SMI*</td>
<td>2.7</td>
<td>4</td>
<td>10.8</td>
<td>20</td>
<td>–</td>
<td>–</td>
<td>20</td>
</tr>
<tr>
<td>Synthesis-PPF</td>
<td>2.3</td>
<td>24</td>
<td>55</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>2</td>
</tr>
<tr>
<td>Synthesis-FFT</td>
<td>1.1</td>
<td>24</td>
<td>26</td>
<td>2</td>
<td>–</td>
<td>–</td>
<td>2</td>
</tr>
<tr>
<td>DET*</td>
<td>2.7</td>
<td>10</td>
<td>27</td>
<td>50</td>
<td>512</td>
<td>8</td>
<td>58</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1053</strong></td>
<td><strong>102</strong></td>
<td><strong>102</strong></td>
<td></td>
<td></td>
<td><strong>116</strong></td>
<td><strong>218</strong></td>
</tr>
</tbody>
</table>

*denotes DSP; all others are custom VLSI