Reconfigurable Target Recognition System

Gabor Szedo, Sandeep Neema, Jason Scott, Ted Bapty
Institute for Software Integrated Systems (ISIS), Vanderbilt University
Nashville, TN 37325, U.S.A.
gabor.szedo@vanderbilt.edu, {neemask|jscott|bapty}@vuse.vanderbilt.edu

Acknowledgements: This work was supported in part by DARPA/ITO under contract DABT63-97-C-0020 and with hardware & tools donations from Altera.

Abstract

This paper describes model-based development of a real-time, embedded Automated Target Recognition (ATR) system. ATR systems have extremely large computational requirements, on the order of 10 GOP/s, in some modes of operation. To implement a system capable of handling high sampling rate and large input images distributed processing must be used. The complexity of the applied Distance-classifier correlation filter (DCCF) algorithm used demands the use of DSP processors; however, some computational bottlenecks cannot be overcome without using dedicated hardware support. 2DFFT and 2DIFFT operations were therefore implemented in FPGA circuits.
From the specifications and test images provided, a heterogeneous, loosely coupled system was developed, which allows run-time algorithm mapping to resources (reconfiguration) from static configuration libraries, according to the changing stages of missile flight.

ATR Algorithm

ATR is an image processing system that should find and classify all objects in the input images, allowing later algorithm stages to guide a missile to its target. The current ATR system should be able to classify objects to 3 different classes, each class defined by a cluster of filters.
The basic task of ATR could be accomplished by correlating the incoming frames with all pictures representing each class, and from these correlations results picking the target image with the largest correlation peak value.
In spite of the extremely large number of correlations necessary to carry out these operations, generating potentially redundant information, results could be ambiguous. Measures of confidence other than the correlation peak value are necessary to select the target in the input image. Therefore the DCCF [1] algorithm was used, which not only classifies input images to arbitrary number of classes, but gives explicit 'confidence' information as well, namely the distance between the input image and class representation images. The block diagram of the algorithm is shown in Figure 1.
In DCCF input image is first normalized, then correlated with template filters from each class. Correlation is carried out in the spectral domain, to decrease the number of multiplications required. {1}

From $$\Theta(N) = N^4, \qquad \{1\}$$

to $$\Theta(N) = N^2(6\log(N)+1). \qquad \{2\}$$

Every template contains the frequency domain representation of an image aggregated from images of the target from different viewpoints.

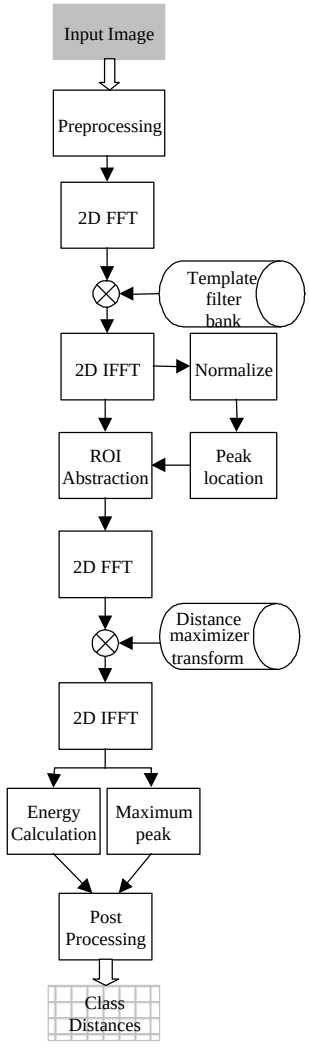


Figure 1. The DCCF Algorithm

Assuming x and h are N x N square matrices, correlation in the spectral domain can be defined as:

$$Y_{a,b} = \mathrm{corr2}(\underline{\underline{x}}, \underline{\underline{h}}) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} x_{i,j} \cdot h_{[a+i]_N [b+j]_N} = \quad \{3\}$$

$$= IFFT2(conj(FFT2(\underline{\underline{x}})) \circ FFT2(\underline{\underline{h}})) = \quad \{4\}$$

$$= IFFT2(conj(FFT2(\underline{\underline{x}})) \circ \underline{\underline{H}})$$

where $\circ$ denotes element by element multiplication, and $\underline{\underline{H}}$ is the template filter.
Correlation surfaces are scanned pixel-by-pixel for peaks. For each peak, the peak-to-sidelobe ratio (PSR) is calculated. An ordered list of peaks with PSR values greater than a threshold is generated. This list contains all potential locations of targets on the input image resembling the actual target category with which the input image was correlated.
Based on the peak location, Regions of Interest (ROIs) are established, which are tiles of the original input image, with the pixel corresponding to the correlation peak in the center. If the point is near the edges of the input image, the ROI image is truncated.
First, the ROIs are correlated with another, a more detailed image of the target. After the correlation the power of the new correlation surface (sum of squares) is calculated. On the same ROI distance, maximization transformations are applied. The cluster of these linear transformations is established beforehand. Each transformation is created from sample class images so as to maximize the distance between classes based on certain features. On the resulting matrices the highest imaginary value is found. In post-processing the actual distance of the ROI from each class is calculated from the measures described above. On the ratio of the distances from separate classes the system can decide, whether the actual point around which the ROI was originally established, is a target, or not.
As the final result of the ATR stage, the system generates a vector for each class, containing the coordinates of a possible target from that class (if it found any), together with distances from other classes, and confidence measures.
In our prototype application, incoming images are composed of only 128 x 128 pixels, but future versions of sensors might increase the available image data to 512 x 512 pixels with frame rates up to 300Hz. Our current target frame rate is 30 frame/s.
ROIs are composed of 32 x 32 pixels, and from each template filter a maximum of 10 target points (ROI centers) can be established.
In worst case, this means:

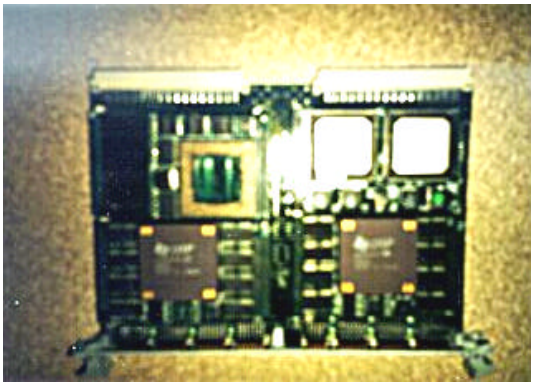
- 3, 128 x 128 2DFFT, Multiplication, 2DIFFT
- Exstraction of 30 ROIs,
- Apply 4*30 = 120, 32 x 32 2DFFT, Multiplication, 2DIFFT,

for each incoming frame.

Hardware Structure

An embedded ATR system poses huge computational requirements, as well as size and power limitations. These conditions can be met only if component utilization is maximized as much as possible for the selected hardware. Also, the selected hardware must have internal flexibility to allow design engineers to utilize the available resources.
A multiprocessor computing system was built from processor modules, each containing either a TMS320C40 DSP processor or an ALTERA FLEX10K100 FPGA,. Each module contains local memory and port interfaces, so the flexible system contains exactly the amount of resources necessary to accomplish the given functional specifications.
Some of the interconnection between the processor nodes is hard-wired on the motherboard (Figure 2.) but front panels connection of processor nodes permits variable topologies. A custom-tailored communication

network can be set up according to the mapping of processes to processors by using subset of the graph specified by the hardware connections.



CHANGE THIS PICTURE: ITS MISSING A PROCESSOR!!

Figure 2. Panel housing 4 processor modules

A heterogeneous, loosely coupled system was developed, which allows algorithm mapping to resources from static configuration libraries. This network allows changing the actual configuration of each processor node (with proper reconfiguration algorithm), according to the changing stages of missile flight.

At the current state of research, only the tightest bottlenecks, the frequency domain transformations, are addressed for hardware acceleration. Our prototype system uses 10-14 DSP and 2-6 FPGA modules for different versions of the same algorithm. In the DCCF algorithm, 2DFFTs / IFFTs are done with two different operand sizes. To speed up these operations a complete 2DFFT (or 2DIFFT) processor was implemented, using 2 FPGA modules. In our FFT processor, the size of the operand does not effect logical complexity of the FFT hardware, rather it affects the time necessary to carry out the operation. Also, the IFFT does not greatly differ from FFT, with extra resources, a flexible module can be generated. However, controlling communication between FPGAs and DSPs is another difficult issue. Managing the complexity of the system level design – process mappings, communication link setup – is difficult without proper tools.

System design tools

Designing such a system poses major challenges to the engineering process, demanding rigorous use of advanced design tools. The ACS [2] design environment developed at ISIS/Vanderbilt University offers such a design environment, runtime support and methodology. A Graphical Model Editor (GME) was used to construct a top-down structural description of the system. GME is a top-level graphical design entry tool, that runs on a Windows based PC. With the Signal Processing Paradigm, the user specifies the system by drawing a data-flow representation with boxes as functions and arrows as communication lines.

This approach made it particularly easy to describe, test, modify and optimize process-to-hardware assignment. Functions are specified as C source files for the DSPs, and as schematic, AHDL (Altera Hardware Description Language), or VHDL description of hardware components for the FPGAs. The graphical models are interpreted by automated tools that generate C source code for each DSP, together with top-level VHDL code for each FPGA and interfaces to establish intercommunication between hardware and software components. The interpreter also generates bootstrap trees, and communication graphs according to the hardware setup, which represents the physical setup of the actual system. During the design synthesis, the tool also verifies the validity of connections, tries to map the process model to the actual hardware architecture and synthesizes all sub-components to ensure correct operation [3] (package forwarders, splitters, etc.).

For all DSPs a common kernel is linked to the functions designated to run on the particular processor. During bootstrapping processors receive their configuration information and start running the kernel, which forwards the next configuration package to an adjacent processor according to the boot tree. Once the kernel is downloaded it schedules all processes running on the processor in a round-robin manner.

FPGAs also have a virtual kernel, which is responsible for booting, and communication across device boundary.

After the Interpreter generates all necessary source these files are compiled by the vendor-specific tools to generate executable code for the target devices. Besides the obvious instantiation of the same C function on different DSP platforms, the modeling paradigm also allows multiple realizations of the same task, even on different hardware platforms. This feature enables the tool to describe and grasp different kinds of reconfiguration: changing the system architecture while maintaining the same functionality, or reusing system resources after their task was completed, by means of altering the process – hardware assignment.

System Models

Top level

Creating a well-balanced processes map is crucial for the overall throughput of the system. Execution times of the different processes, data-dependence, physical location all must be concerned when assigning processes to processors. If processe ‘A’ on processor ‘a’ uses results of process ‘B’ on processor ‘b’, and ‘a’ is not directly linked to ‘b’, the system must to find an indirect connection (through other processors) in compilation time. A forwarder process is inserted in each subsequent processors process list to accomplish data routing. During execution, processor ‘a’ has to wait not just for processor ‘b’ but all other processors to allow these forwarder routines to run. Creating multiple data paths (as many as possible) again will not lead to the best possible communication efficiency, as dynamic selection of data-paths causes extra overhead.

Keeping these considerations in mind, the process dataflow architecture described in Figure 3. was selected.

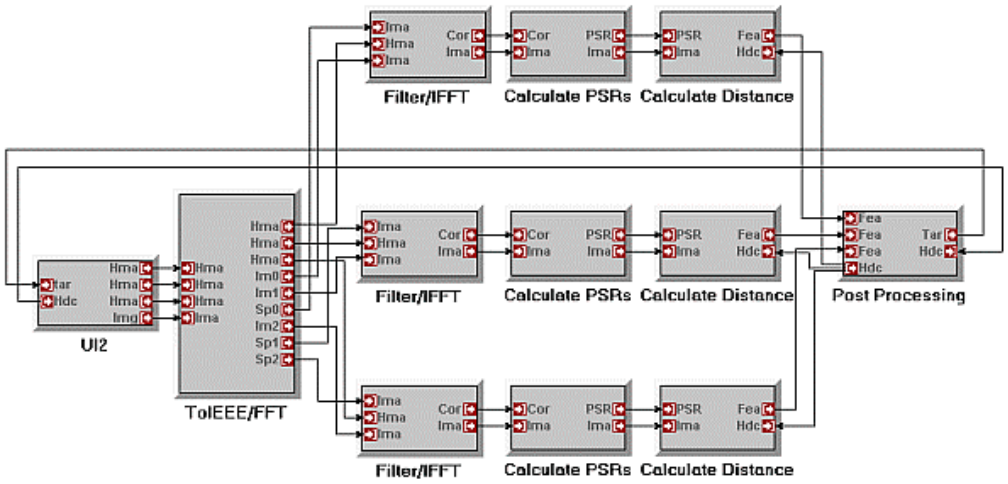


Figure 3. Top level model

In the prototype system the image sensor was simulated by a host PC that feeds previously recorded images to the system. Also, the host stores both filter banks, and allows receiving and transmitting different package types that is essential to debug the application. Host (UI: user interface), is the leftmost block. Floating point format of the PC (IEEE) is different from the DSPs (TI), therefore floating point operands must be converted. The next block (TOIEEE/FFT) carries out these conversions together with 2DFFT (128 x 128) calculation of

the incoming frame. The module also retains the incoming correlation filters, which are downloaded to the system only once for many images. (Correlation filters are altered only when new targets are necessary). This module splits the incoming images as well as their spectra to three different datapaths, which carry out all other processing. On Figure 4 a hardware node reference (on the right top, named N1) is also presented, which assigns all processes in this compound to run on node 1.

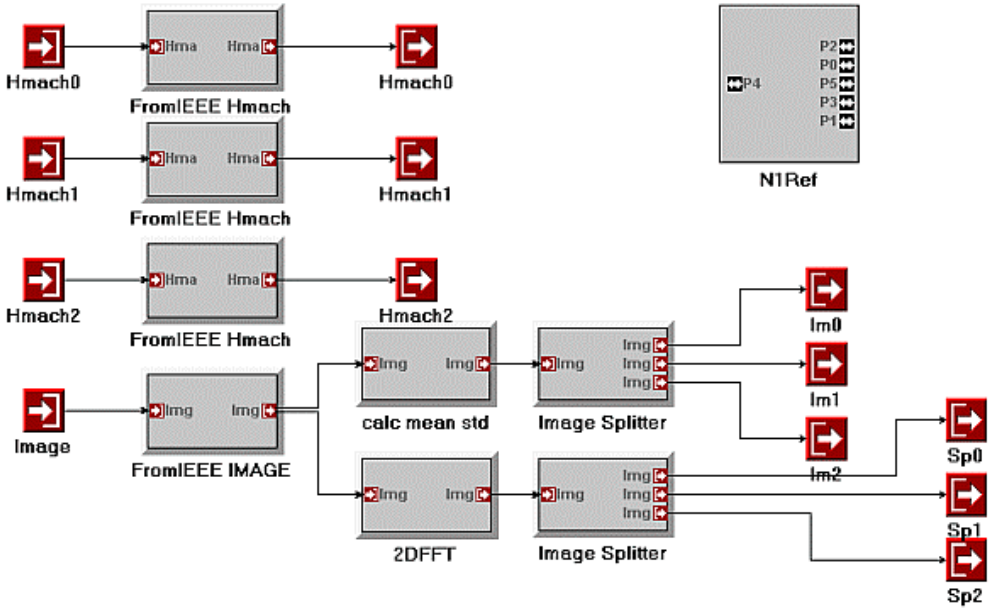


Figure 4. FromIEEE/FFT

In each of the three datapaths, the Filter/FFT creates the correlation surface. Further on, PSRs are calculated, from the PSR list and the original image ROIs are extracted according to the DCCF algorithm. From ROIs and the classification filters, class distances are calculated, post-processed and fed back to the User Interface.

FFT applications

2DFFTs/IFFTs can be carried out either on DSPs or FPGAs, with minor changes in the system model. Two realizations are supported for the FFT: one on DSP and one on FPGA. During the system synthesis process, the designer can select from different configurations, according to the targeted hardware setup.

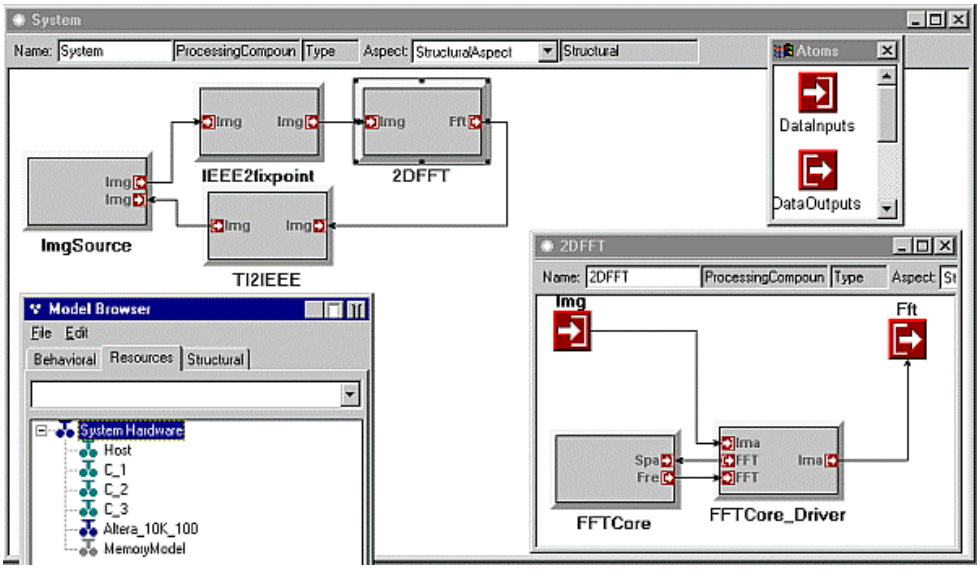


Figure 5. Hardware model of the 2DFFT block

The model above shows the testbench of the hardware (FPGA) version of the 2DFFT module. The Image source (ImgSource) runs in the host, format converters run on C1 (a DSP node, see the Hardware Resources window on the left bottom), 2DFFT is implemented in the Altera_10K_100.

Hardware realization of 2DFFTs

The hardware components necessary to carry out the operation did not allowed to squeeze all components to a single FPGA therefore two FPGAs were used. (Figure 5.) One module (slave) implements a 32/128 point 1DFFT, while the other (master) breaks down the 2DFFT to 1DFFT operations, supporting operands, and receiving results from the slave.

As the applied 1DFFT component uses 16 bit fix point number representation, which can handle a limited range of operands, operands must be transformed to this range. Having X as operand, a linear transformation is performed:

$$\underline{\underline{Y}} = \frac{2}{M-m}\underline{\underline{X}} - \frac{M+m}{M-m}, \qquad \{5\}$$

where M=max($\underline{\underline{X}}$), m=min($\underline{\underline{X}}$)

This operation is carried out on the DSP side, before the matrix is converted from floating point format to fixed point format and sent to the FPGAs. Consequently, M and m are retained until the matrix is processed, and after conversion from fix point format, the inverse transformation:

$$\underline{\underline{X}}' = \frac{M-m}{2}\left(\underline{\underline{Z}} + \frac{M+m}{M-m}\right), \qquad \{6\}$$

is applied, where $\underline{\underline{Z}}$ is the resultant matrix of the 2DFFT operation.

Slave FPGA

1D FFT is implemented using Altera's FFT megacore block. This is a scalable AHDL component, containing multiplier and adder sub-units which accomplish a butterfly operation, together with scheduler logic that addresses local (on chip) memory to store temporal results.

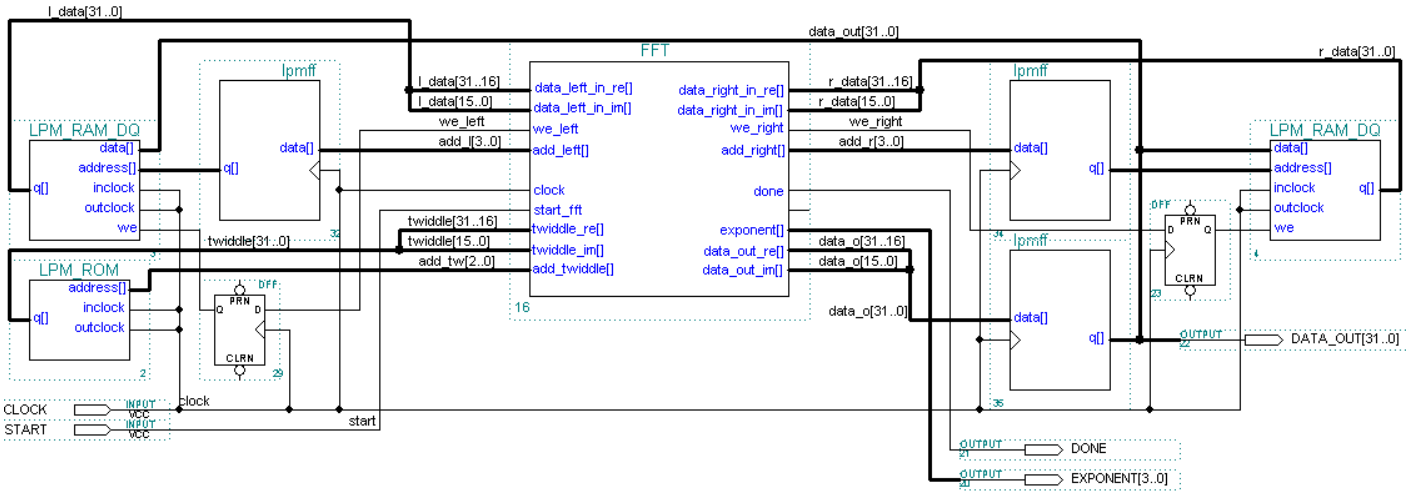
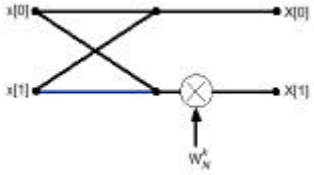


Figure 6. Schematic diagram of FFT core

The megacore block uses three different memories to maximize its throughput. Therefore, the 12 Embedded Array Blocks (EAB) the FLEX10K100 [4] contains are grouped into 3 memory parts: left and right RAMs, and a twiddle ROM. Each EAB can be configured to store 256 x 8 bit, so each memory partitions is capable of storing 256 x 32 words. Each word is splitted to 16bit real, and 16 bit imaginary parts. (Figure 6.) Before triggering the FFT megacore block to start the operation, all operands must be loaded to the right memory. Operands are in two's complement fractional notation, a fixed point number representation in the range of [-1,1[. Twiddle factors are stored in negated format, being able to represent the most often used twiddle factor, +1. Using this fixed-point format needs extra care to handle overflows. In worst case, two overflows might occur within a butterfly:



$$X[0] = x[0] + x[1]; \qquad \{7\}$$

$$X[1] = (x[0] - x[1]) \cdot w_N^k; \qquad \{8\}$$

If $x[0] = -1 - i,\ x[1] \approx 1 + i$ and $w_N^k = 0.707 + i0.707$,

overflow occurs during subtracting x[1] from x[0], then during the complex multiplication. As the FFT has no effect on its initial operands, these overflows are inevitable, unless the FFTcore automatically scales down data 2 digits at the first pass. If all results (the operands of the next pass) are smaller than 0.25 in absolute value in the first pass, no downscaling is necessary during the next pass. To keep track of scaling through passes, a block exponent is assigned to the current vector. This exponent is sent to the Master FPGA along with the results as the 2^N+1 vector element. In this sparsely used element, a vector ID is also encoded.

During the operation the FFT reads operands from the right memory and the twiddle memory simultaneously and, after a transient period while its queue is not full, starts accessing the left memory, to write temporal results. (Figure 7.) After all operands from right memory are processed, the first pass is over, the block starts addressing the left RAM for operands, and store results in right RAM. For 2^5 = 32 points, 5 passes are necessary, that means final results are going to be stored in the left RAM.

At each clock cycle, an operand is polled, and a result is stored. In case of N points, it means at least N clock cycles to complete a pass, aside from the overhead at the beginning (filling the queue) and at the end (emptying the queue) of the pass.

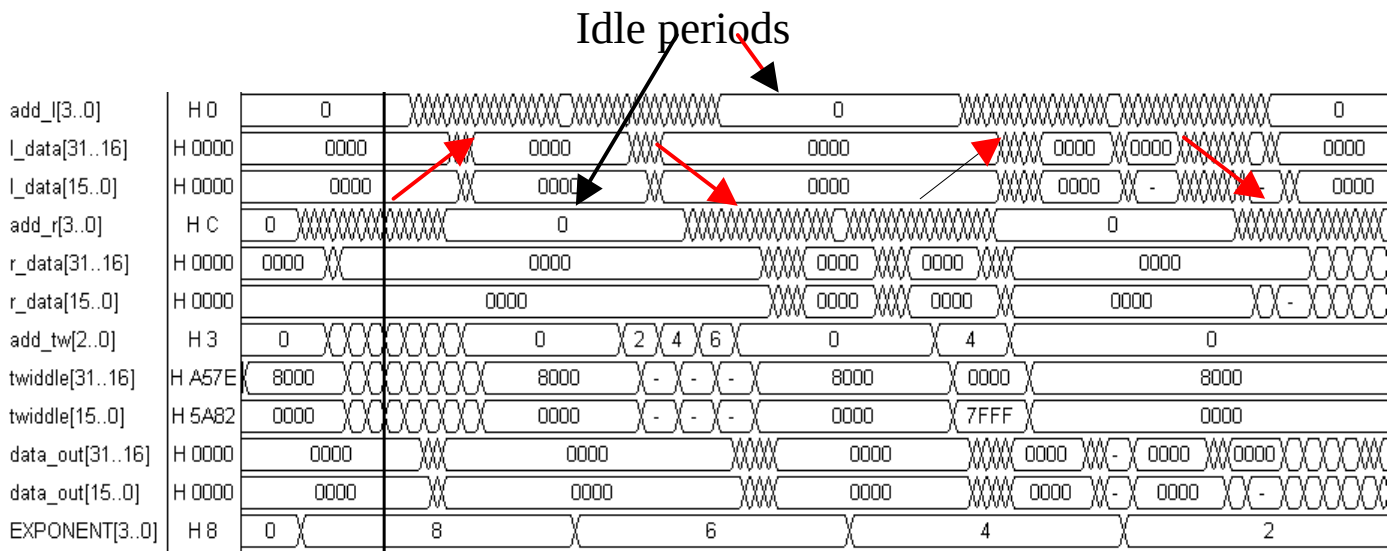


Figure 7. Functional simulation diagram of 16 point FFT.

As there are $\log_2 N$ passes, in case of a 32 point FFT, 282 clock cycles are necessary to carry out the whole FFT. At the current state of the research efforts are made to increase the maximal clock frequency from the actual 12.84 MHz over 20 MHz, which would allow us to do a 1DFFT in 14.1us, a 2DFFT in 0.9024 ms (1108 Frame/s).
Another obstacle to achieve this performance is the loading of new operands and the unloading of last results. To accomplish loading/unloading during the FFT operation, the memories were divided to two in depth: both the left and the right RAM contains two separate banks. The FFT core, together with the memories is extended with FIFOs, and a sophisticated scheduler. The scheduler ensures, that the FFT reads and writes using the same bank in both RAMs. When the scheduler detects, that address lines of any RAM are holding 0 for more than 1 clock cycle, the memory is not used meanwhile, it detaches that RAM, and let the corresponding FIFO to access the other bank of the particular RAM. During idle periods, the FIFOs upload/download data from the unused RAM banks. If the FIFOs were empty/full during the idle periods, therefore the FFT is not ready for the next cycle, the scheduler pauses until all data transfers are done. Of course, after the FFT core finished (its done signal goes high), both FIFOs attain full access to the memories to complete their operations.
To carry out inverse FFT operations, only the twiddle factors and the results should be conjugated, which are carried out by additional logic in this module.

Master FPGA

The first module controls the data flow of the 2D operation. It receives incoming matrices, put them into a frame buffer, from where it successively processes them through the slave. To carry out the 2DFFT operation, first the FFT of the rows of the matrix should be calculated, then the FFTs of the columns of the resulting matrix should be calculated.

Vectors from the operand matrix are read row by row and send to the slave, using a handshaking protocol. As the slave has input and output buffers, loading/unloading can be asynchronous, vectors are reconstructed by their sequence number and the last (2^N+1) element. Results of the row-by-row FFTs overwrite the original matrix, while their block exponents are stored in a different vector in local (on-chip) RAM.

After the first part is completed, the maximum block exponent is also calculated. By the column-by-column operation, all vector elements must have the same exponent. Therefore, each vector element must be scaled

(shifted to the right) by the calculated minimal exponent minus its particular (row-)block exponent. After the columns were processed resulting spectra vectors also have block exponents. These exponents also must be stored. After all the vectors are processed, the same maximum exponent must be calculated. When the output is returned to the DSP, each word must be scaled according to the maximum and the corresponding column exponent.

Reconfiguration

When the system is far from the target, in the target acquisition state, relatively few incoming frames must be processed, with many possible targets to scan. When the missile homes in on a particular target, all sensor frames must be processed, although there are fewer targets to focus on. (Figure 7.)
In the ATR algorithm there are two blocks using 2DFFT/IFFT operations:

1. applying template filters operates on 128x128 pixel matrices. In this case, the number of transformations necessary is proportional to the frame rate,
2. applying class separation filters on 32x32 ROIs, during class distance calculation. In this case, the number of transformations necessary is proportional to the frame rate and the regions of Interests (ROIs) found on the incoming images.

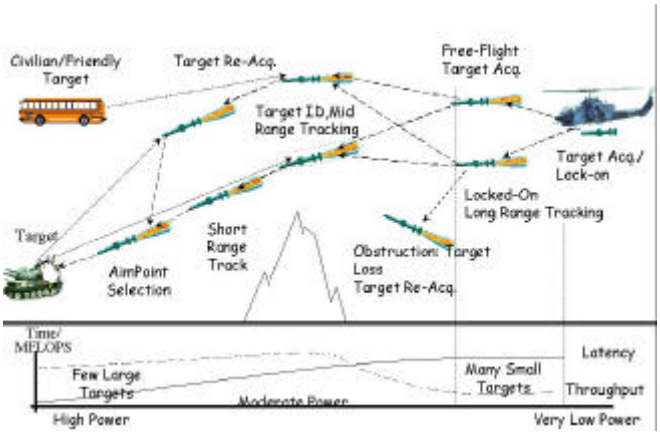


Figure 7. Operation modes

Therefore, in the target acquisition state, the second set of FFTs, while in the short range track mode, the first set need hardware support. 1DFFT hardware itself is easy to adjust to 128 point FFTs, apparently changing the number of points does not affect the size of the hardware. The memories are ‘allocated’ in 256 byte clusters, using 2*32 or 2*128 does not require allocating more EABs. Also, the size of the butterfly operation is determined by the accuracy, not the number of points to process. Only the address buses and connecting multiplexers, registers are affected by the mode change.
Similarly, the resources of the master FPGA are just slightly affected by changing operational mode.
Although both FPGAs could be easily reconfigured to different modes (retaining functionality, size, and pin assigmnets), transient management requires further work on seamless switch between modes.

Conclusion

FPGAs are enabling technology for a computing platform that is able to adapt to the changing requirements of the algorithm to be evaluated. FPGAs may be used as custom tailored, general purpose computing devices whose flexibility and speed fall between ASICs and CPUs. The algorithm of the embedded system required both 2D FFTs, IFFTs as well as image sizes of 32x32 and 128x128. A prototype system was built comprising 10-14 DSPs and 2-6 FPGAs.

Dynamic reconfiguration provides a better utilization of hardware over the different operational modes of the system. Hence the sizes, port mapping, and functionality of the different configurations are the same, partial reconfiguration can take place. On the other hand, reconfiguration – together with all transient management – must be real time to meet system deadlines. To find the optimal performance with a given set of hardware components, many possible configurations must be examined, from different aspects. To support handling of different configuration, a system design tool was developed that allows multiple physical implementations of a certain model.

References

[1] A. Mahalanobis, B.V.K. Vijaya Kumar, S. R. F. Sims “Distance-classifier correlation filters for multiclass target recognition “, Optical Society of America, 1996.

[2] Franke H., Sztipanovits J., Karsai G.: "Model-Integrated Computing", Proceedings of the 1997 Hawaii Systems Sciences Conference, (no page number available, CD-ROM publication), 1997.

[3] Scott J., Bapty T., **“**Runtime Environment for Dynamically Reconfigurable Embedded Systems”, ICSPAT-99, Florida, November 1999.

[4] Altera Corporation, “FLEX 10K Programmable Logic Family”, October, 1998, ver 3.13, A-DS-F10K-03.13

[5] Altera Corporation, “Fast Fourier Transform Data Sheet”, April 1997, ver 2, A-DS-FFT-02

Reference Sandeep’s Environment/Reconfiguration Paper, MAPLD99