A HOUGH TRANSFORM RAPID PROTOTYPING SYSTEM USING THE MATLAB EMBEDDED TARGET FOR THE TI TMS320DM642 EVM

Cheyne Gaw Ho
MIG Development. Email: cheyneho@netscape.net.

ABSTRACT

The Embedded Target for TI TMS320C6000 DSP is a useful development tool, which can be used to generate executable code for a range of targets, from a set of Simulink model blocks. In this paper we outline the process of code generation for the TMS320DM642 Evaluation Module (EVM) using a Hough transform algorithm. This was computed using the forward Radon transform, evaluated using the central slice theorem, and involves a 2-D Discrete Fourier Transform (DFT), a rectangular to polar coordinate transformation, and a 1-D DFT. This was implemented using Simulink model blocks together with custom blocks, which were written using the S-Function Builder. The Embedded Target generates all the video capture and display routines required, so that the algorithm can be developed in Matlab and Simulink, before being tested on the EVM. Problems and limitations with the Embedded Target are highlighted, and solutions and code profiling optimizations are proposed.

Index Terms— Digital signal processors, Hough transforms, Image processing, Radon transforms, Software prototyping.

1. INTRODUCTION

The Embedded Target for TI TMS320C6000 DSP [1] allows the development of signal and image processing algorithms on standard TI DSP hardware, such as the fixed-point TMS320DM642 Evaluation Module (EVM) from Spectrum Digital [2]. The Real-Time Workshop Embedded Coder [3] uses the Embedded Target to generate code from Simulink models, and creates a TI Code Composer Studio DSP/BIOS project to compile, execute and run the generated code on the target processor.

The Simulink models are composed of several blocks from the Simulink library, the C6000lib libraries, and from the Signal Processing and the Video and Image Processing (VIP) blocksets. The C6000lib libraries form part of the Embedded Target for TI C6000 DSP, and contain the DM642 EVM video capture and display routines. The Signal Processing and VIP blocksets provide block libraries for many signal and image processing operations. However some of these functions are very basic, such as the Hough transform block from the VIP blockset, which only supports a Hough transform [4] on a binary image, using the parametric expression: $r = x \cos\theta + y \sin\theta$.

More complex algorithms, which are not available in the Signal Processing and VIP blocksets, can be composed from several other blocks. This produces very inefficient code, since the Real-Time Workshop Embedded Coder does not combine blocks very efficiently, and instead generates code segments for each block used, which are then linked together.

Figure 1. DM642 Embedded Development System.

Another way to implement complex algorithms is to write custom Simulink C S-Functions using the S-Function Builder. This enables more efficient code to be produced, which can be easily profiled and optimized in the Code Composer Studio Integrated Development Environment (IDE).

The algorithm can be tested in Simulink with inputs from an image or media file, with the results output to the workspace or screen for analysis. Once the algorithm has been refined sufficiently, DM642 video capture and display blocks, or Real-Time Data Exchange (RTDX) blocks, can be added to the model. For RTDX transfers the Link for Code Composer Studio [5] can be used to communicate with the target processor from within the Matlab environment.

We have designed and constructed an Embedded Development System using the DM642 (figure 1). This consists of a CCD camera, TFT display, and DM642 EVM, together with a Lithium Ion battery and suitable power conversion circuitry. The EVM contains a 720MHz TMS320DM642 DSP [6], which has 16KB L1P and L1D program and data cache memories, 256KB configurable L2 cache/mapped memory, 32MB of external SDRAM and 4MB of non-volatile flash memory.

In this paper we outline the process of code generation using Simulink model blocks and custom S-Function blocks, which were written using the S-Function Builder. A Hough transform algorithm was implemented using the forward Radon transform [7], evaluated using the central slice theorem [8]. This involves a two dimensional Discrete Fourier transform (DFT), a rectangular to polar coordinate transformation, and a 1-D DFT, to produce a sinogram in the Radon Space.

The algorithm implemented [9] was originally developed using Matlab 4.2c.1, and redeveloped in Simulink using Matlab R2006a (7.2), with Code Composer Studio v3.1 SP1. Due to bugs which were found in the Matlab provided Lcc C Compiler, C Mex files were compiled using Microsoft Visual C++ v6.0.

2. RADON TRANSFORM SIMULINK MODEL

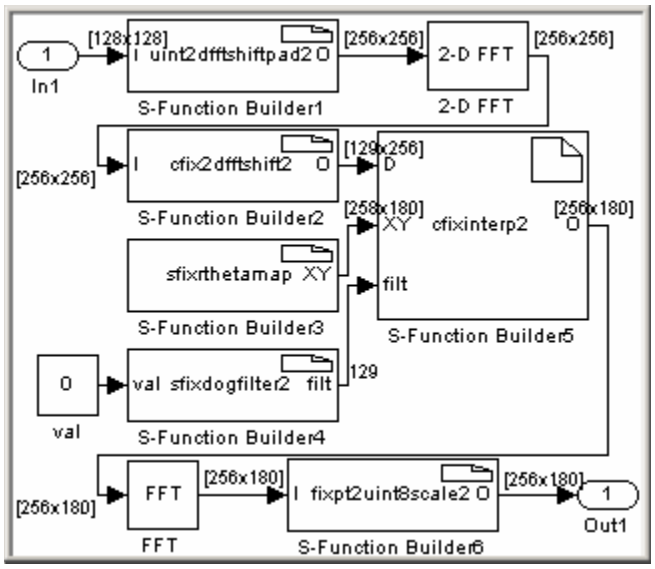


Figure 2. Radon Transform Simulink model.

Figure 2 shows the Radon Transform Simulink model. Initially an 8-bit grayscale image of size 128x128 was used with a matrix viewer block to display results to the screen. For code generation this was replaced with the DM642 capture and display blocks from the C6000lib DM642 Board Support library. 180 angular slices were taken between 0 and π to produce a sinogram of size 256x180.

The size of each block was made tunable, so that their sizes were determined by the parameters in the model, which were set up in the Simulink Model Explorer. Reusable functions were created wherever possible, by storing code for several different operations within the block.

The image was first zero padded and 2-D fftshifted, by swapping the first and third, and the second and fourth quadrants of the image, in S-Function Builder 1. Zero padding the image improves the resolution of the resulting 2-D Fourier spectrum, and reduces the aliasing effects which arise from the rectangular to polar resampling used in S-Function Builder 5.

A 2-D FFT was then taken to produce complex data, which was stored in a structure containing interleaved 16-bit real and imaginary components in Q.15 format, (sfix16_En15 in Matlab fixed-point notation). The bottom half of the result was 2-D fftshifted again in S-Function Builder 2, so that the zero-frequency component was moved to the centre of the 2-D Fourier Spectrum.

Since the original image is real its 2-D Fourier spectrum is symmetric about its origin, (diametric Hermitian symmetry), so it was only necessary to sample the top half of the 2-D Fourier spectrum. This was resampled from rectangular to polar coordinates in S-Function Builder 5, using bilinear interpolation with precomputed look-up table data from S-Function Builder 3. The rest of the data was obtained by conjugate reflection of the resampled data about the zeroth frequency.

The look-up table data was stored in Q16.15 format, (sfix32_En15 in Matlab fixed-point notation), and contains the Cartesian location of each polar sample point in the mapping. The sine and cosine values used were computed from linear interpolation of successive points in a fixed-point sine table.

After resampling from Cartesian to polar coordinates, the resulting 1-D Fourier spectrum was filtered with a 1-D difference of Gaussian (DOG) filter [9-10], which was computed in S-Function Builder 4, to edge enhance and improve the peak structure of the sinogram. A 1-D FFT was then taken of the result to produce the sinogram. This was 1-D fftshifted, by swapping the left and right halves of the image, and quantized to 8-bit resolution, in S-Function Builder 6.

The look-up tables in S-Functions 4 and 5 were computed at each time step when the model was run, since there was no option to run the block only once at start up. This can be fixed in the Code Composer Studio IDE after the model has been built.

3. S-FUNCTION BUILDER

The S-Function Builder consists of a dialog box with several tabs for setting up the parameters, the output and input ports, the location of any include files, and the algorithm, which is contained in a C wrapper function. The S-Function Builder generates a Simstruct S-Function code file and a Real-Time Workshop Target Language Compiler (TLC) file, from the parameters and block properties which have been specified. The S-Function Builder then uses the Matlab Mex command to compile and link the Simstruct S-Function code file and the C wrapper function into a dynamically loadable Matlab executable (MEX) file, for use in Simulink. The Real-Time Workshop Embedded Coder also uses the C wrapper function with the TLC file to generate code for the DM642 EVM.

The input and output port sizes, data types and complexity, were set up in the data properties tab, together with the parameters that are passed to the function. However, the generated Simstruct S-Function code for the block needed to be modified to provide correct support for complex fixed-point data types, and to enable the size of the ports to be specified in terms of the parameters.

The algorithm for the block was entered in a box in the outputs tab as a C wrapper function. Pointers to the input and output ports, and the parameters, which have been specified in the data properties tab, are passed to the function, together with the width of the arrays containing the S-Function's inputs and outputs. The S-Function Builder inserts a call to the C wrapper function in the mdlOutputs callback method in the generated Simstruct S-Function code for the block. This is called by Simulink at each simulation time step to compute the S-Function's output.

The #IFDEF directive was used to distinguish between code for the Simulink model, (MATLAB_MEX_FILE), and code specific to the DM642. When using the TI C64x DSP Library (DSPLIB) [11] and the TI C64x Image/Video Processing Library (IMGLIB) [12], the Programmer's References provide functional C code, equivalent to the assembly code used in the functions, which can be used to model the functions in Simulink.

For moving contiguous blocks of memory, the memove command was used for the Simulink model, and the DAT_copy2d command was used to transfer the data in blocks to the L2 SRAM using the enhanced direct memory access (EDMA) controller on the DM642 [13]. The UNROLL and MUST_ITERATE pragma directives were also used, to allow the compiler to optimize loop unrolling, and to maximize the use of the data bus.

4. CODE COMPOSER STUDIO TUNING

When the model is built the Real-Time Workshop Embedded Coder uses the Embedded Target to create a Code Composer Studio DSP/BIOS project, and generate code from all the blocks in the model. This is compiled together with the C wrapper files and any external files, which have been specified in the Real-Time Workshop configuration options, to create a single executable common object file format (COFF) file, which is then downloaded and run on the EVM.

The Real-Time Workshop Embedded Coder generates all memory statically, so that memory is allocated for each block in the model. Since the Embedded Target does not produce very efficient code, a lot of memory is allocated for intermediate variables, so the available memory can quickly run out. The size of each block was also fixed, so the model needs to be rebuilt if the parameters are changed.

All memory arrays which were generated by the Embedded Target were automatically aligned to 8-byte alignment boundaries, to achieve single instruction multiple data (SIMD) optimization, so that multiple operations can be combined together into a single instruction. Other memory arrays can be manually aligned using the DATA_ALIGN and DATA_SECTION pragma directives.

To analyze the performance of the algorithm, STS objects were initialized with the DSP/BIOS Configuration Tool [14], and STS_set and STS_delta calls were inserted between each section of code. Table 1 shows the average processing time of each of the processing stages. This takes about 237ms to process one 128x128 8-bit grayscale image. This is mainly due to the FFT blocks from the Signal Processing and VIP blocksets, which take about 222ms and are not optimized for the DM642.

STS	Task	Time/ ms
1	S-Function Builder 1 (uint2dfftshiftpad2.c)	6.39
2	2-D FFT block (256x256)	152.82
3	S-Function Builder 2 (cfix2dfftshift2.c)	2.05
4	S-Function Builder 5 (cfixinterp2.c)	5.06
5	1-D FFT block (256x180)	69.28
6	S-Function Builder 6 (fixpt2uint8scale2.c)	1.66
0	Total	237.27

Table 1. DSP/BIOS timing data for a 128x128 image.

The assembly language code produced by the Code Composer Studio compiler contains software pipeline information about how efficiently loops are pipelined, and how many of the processor's instruction units are operating simultaneously. By studying the assembly language code, inefficient processes can be identified and optimized by modifying the C code, or changing the compiler options to allow it to produce more efficient assembly language code.

The code can be improved further by writing linear assembly code, or inlining intrinsic operators and assembly language code into the C code. There are many C callable assembly optimized routines available in the TI C64x DSPLIB, which is available in the C6000lib C64x DSP library, and also in the TI C64x IMGLIB. Further code profiling and optimization steps are documented in the TMS320C6000 Programmer's Guide [15].

5. DM642 EVM OUTPUT

Figure 3 shows the Radon transform of a 128x128 8-bit grayscale image, which was captured with a Sony CCD camera using the DM642 Embedded Development System shown in figure 1. Since the video encoder on the EVM only supports 8-bit data, the output was quantized to 8-bit resolution in S-Function Builder 6, for display at 640x480.

The image was first edged detected using a spatial domain Sobel filter. Each pixel in the image maps to a sinusoid in Radon space, which intersect at points corresponding to the parametrisation of the lines in the image, and produces butterfly dispersions around each maximum point [16]. To detect these maxima points, the 1-D Fourier spectrum can be bandpass filtered to attenuate both the low and high frequency components. This produces a peak enhanced sinogram which can be thresholded to locate the maxima points in the sinogram.

Figure 4 (a) shows the Radon transform of the original 8 bit grayscale image from figure 3. This was filtered with a 1-D DOG Filter [9-10], to produce the sinogram shown in figure 4 (b). The edges of the image were detected causing spurious maxima points, which were reduced by subtracting the Radon transform of the average intensity of the image.

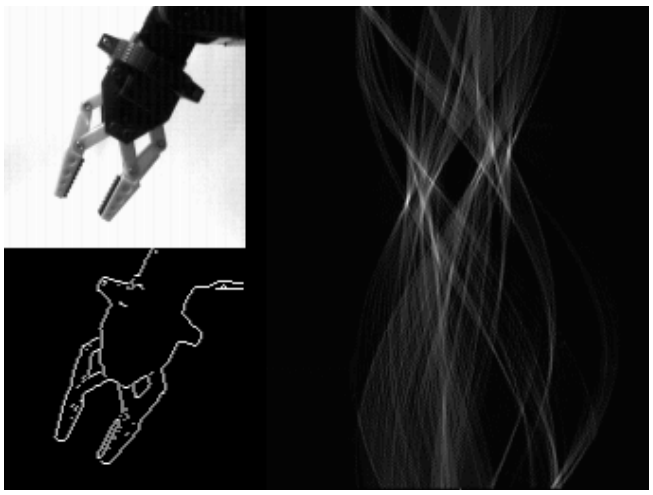
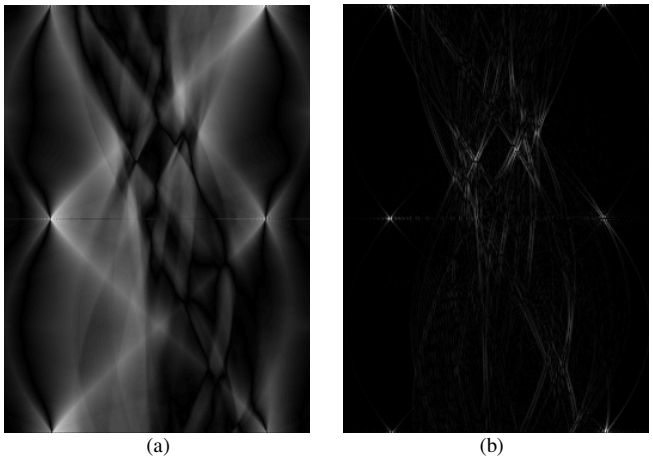


Figure 3. Radon transform of a Sobel edge detected 128x128 8-bit grayscale image.



(a) (b)

Figure 4. Effect of the filtering the Radon Transform of the original image from figure 3.

6. CONCLUSIONS

In this paper we have outlined the process of code generation for the TMS320DM642 EVM using the Embedded Target for TI C6000 DSP. We show how the Embedded Target can be used for rapid prototyping using a Hough transform algorithm. This was implemented using the central slice theorem and involves a 2-D DFT, a rectangular to polar coordinate transformation, and a 1-D DFT.

The algorithm was implemented using blocks from the Simulink library, the C6000lib libraries, and the Signal Processing and VIP blocksets. The C6000lib libraries contain the DM642 EVM video capture and display routines, and other board support routines, together with optimized assembly language code in a C64x DSP library.

Since the Real-Time Workshop Embedded Coder does not combine blocks very efficiently, complex algorithms were implemented using custom Simulink C S-Functions, which were written using the S-Function Builder. Reusable functions were written wherever possible, by storing code for several different operations within the block.

The generated code was profiled in the Code Composer Studio IDE using the DSP/BIOS tools to identify inefficient areas for optimization. These optimizations can be implemented in the Code Composer Studio IDE, or in the S-Function Builder, using the #IFDEF directive to distinguish between code for the Simulink model, and code specific to the DM642.

7. REFERENCES

[1] The Mathworks, *The Embedded Target for TI TMS320C6000 DSP Platform User's Guide.* Version 3.0. The Mathworks, Natick, Massachusetts, March 2006.

[2] Spectrum Digital, *TMS320DM642 Evaluation Module Technical Reference.* Spectrum Digital, Stafford, Texas, August 2003.

[3] The Mathworks, *Real-Time Workshop Embedded Coder User's Guide.* Version 4.4. The Mathworks, Natick, Massachusetts, March 2006.

[4] Hough, P.V.C., *Method and means for recognising complex patterns.* U.S. Patent No. 3,069,654, 1962.

[5] The Mathworks, *Link for Code Composer Studio Development Tool's User's Guide.* Version 2.0. The Mathworks, Natick, Massachusetts, April 2006.

[6] Texas Instruments, *TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor Data Manual.* Tech. Rep. SPRS200B, Texas Instruments, Dallas, Texas, May 2003.

[7] J. Radon, "Über die bestimmung von funcktionen durch ihre integralwerte längs gewisser mannigfaltigkeiten." *Ber. Sächs. Akad. Wiss. Leipzig.*, vol. 69, pp.262-278, 1917.

[8] R.M. Mersereau, and A.V. Oppenheim, "Digital reconstruction of multidimensional signals from their projections." *Proc. IEEE*, vol. 62, no. 10, pp. 1319-1338, 1974.

[9] C.G. Ho, R.C.D. Young, C.D. Bradfield, and C.R. Chatwin, "A Fast Hough Transform for the Parametrisation of Straight Lines using Fourier Methods," *Real-Time Imaging*, vol. 2, pp. 113-127, 2000.

[10] D. Marr, and E. Hidreth, "Theory of edge detection." *Proc. R. Soc. Lond. B.*, vol. 207, pp. 187-217, 1980.

[11] Texas Instruments, *TMS320C64x DSP Library Programmer's Reference.* Tech. Rep. SPRU565A, Texas Instruments, Dallas, Texas, April 2002.

[12] Texas Instruments, *TMS320C64x Image/Video Processing Library Programmer's Reference.* Tech. Rep. SPRU023A, Texas Instruments, Dallas, Texas, April 2002.

[13] Texas Instruments, *TMS320C6000 DSP Enhanced Direct Memory Access (EDMA) Controller Reference Guide.* Tech. Rep. SPRU234, Texas Instruments, Dallas, Texas, July 2003.

[14] Texas Instruments, *DSP/BIOS User's Guide.* Tech. Rep. SPRU423D, Texas Instruments, Dallas, Texas, April 2004.

[15] Texas Instruments, *TMS320C6000 Programmer's Guide.* Tech. Rep. SPRU198G, Texas Instruments, Dallas, Texas, August 2002.

[16] V.F. Leavers, and J. Boyce, "The Radon transform and its application to shape parametrisation in machine vision." *Image and Vision Computing*, vol. 5, no. 2, pp.161-166, 1987.