Rapid Application Development (RAD) and code optimization technique

M. Ahmadian *, N Nakhaee †, Andrew Nesterov †
*School of Engineering & Electronics , University of Edinburgh, Edinburgh, UK
†Sundance Digital Signal processing Inc. 4790 Caughlin Parkway #233 Reno NV 89509-0907 U.S.A.

***Index Terms*— rapid application development, RAD, embedded system, Matlab ®, Simulink ® ,Real-Time Workshop ®, code optimization, DSP, SMT6050**

***Abstract*— In this paper the efficiency of code generated by a particular RAD systems will be examined and contrasted with other available systems. The functionality and performance of SMT6050, the new software module from Sundance, helping RAD systems target DSP will be fully discussed. The RAD used as the base system, in this case, is Simulink and Real Time Workshop (RTW) from MathWorks . These two products work in conjunction to simulate (Simulink) and then to generate code (Real-Time Workshop) for a real time system model. This combination represents an effective RAD for designing embedded systems, and with the addition of Sundance SMT6050 and GDD toolbox for Simulink component, prototypes can be developed in record time and with impressive performance. This paper investigates Real-Time Workshop code generation techniques and performance of the resulting code. The structure of the generated code and the interaction of various components including SMT6050, Simulink and Real-Time Workshop are presented in details The paper also explains the parameters that a designer should take into consideration when designing a model in Simulink so the generated code could be as optimised as possible. The performance gain resulting from using these tools are also measured and presented.**

I. Introduction

Rapid Application Development has opened its doors to embedded system designers for some time now [1] [2] [3]. There are several products already in the market [4], and some are in research phase [5], for automatically generating embeddable codes. As the performance of an embedded system is always a very important parameter, there is a vital question to be considered:

How optimised is the RAD generated code?

The SMT6050 [4] software was designed to assist system developers in using MathWorks [6] well established RAD system to target Texas Instrument's DSPs [7] . SMT6050 assists Real-Time Workshop [8] to generate appropriate efficient codes for Sundance [9] DSP boards, which are based on Texas Instrument's DSP processors series TMS320C6000. SMT6050 supplies platform information to Real-Time Workshop and controls it to generate DSP target code from a Simulink [10] model. SMT6050 then generates a make file to compile and link all the created source codes to building an application that can be downloaded into the specified underlying DSP board. SMT6050 uses Code Composer Studio (CCS) for compilation and linking. CCS can be used for downloading the created application into the DSP hardware. SMT6050 also generates some other non-C files that are needed for building the generated application. For example, SMT6050 interrogates the installed DSP board to determine how much memory it has and generates a suitable memory map and linker command file to be used during linking phase in CCS. SMt6050 could use highly optimised GDD libraries [11], depending on whether the designer has used the GDD block set in the Simulink model or not. GDD libraries are hand optimised assembly codes for TMS320C6000 processors. The library functions have been optimised algorithmically at the assembly level and hand coded in assembly to obtain the maximum possible performance, and at the same time to provide for the maximum accuracy achievable in single precision arithmetic (32-bits IEEE-754 floating-point numbers). Therefore, users have the benefit of improved performance, which in some cases could be as much as 10 times faster than the corresponding C functions. Usually the use of the library functions would give 30% through 300% performance gain. In short, the steps for developing a working software application in this system can be summarised as: create software model in Simulink using GDD block sets, simulate the model, generate code using Real-Time Workshop and SMT6050, compile and link using CCS and the run and test resulting solution.

II. Simulation and code generation technique

Simulink uses a model-based approach for the rapid application development. In a model based system design, a system is a model that is constructed from other interconnected models. Simulink is shipped with several primary models; other models can be bought from MathWorks or third-party suppliers. To design a system using Simulink, the designer begins with developing a model for the desired system in Simulink and debugs it using Simulink simulator. The design phase is finished when the developer has obtained satisfactory results of the simulation. For the code generation phase the system designer would use SMT6050 to generate code for the developed model. Based on this approach to system design, designing a system has three steps:

1) Model construction.
2) Model simulation.
3) Model code generation.

A. Model construction

During this phase, a model is constructed by interconnecting available models. Simulink comes with several primary models and other models can be bought from MathWorks or third parties. SMT6050 provides with the models for hardware input/output devices on the Sundance boards and ADC and DAC modules. For the purpose of this paper, we are using "demodspcopdsp.mdl" demo model that is shipped with SMT6050 (a copy of this model can be found in the Matlabroot/sundance/demos/dsp_cop subdirectory). This model that is shown in fig.1, demonstrating how to design a DSP coprocessor for Matlab. The idea behind a DSP coprocessor is that a DSP board can act as a coprocessor for Matlab to offload the heavy duty DSP processing from a PC to a DSP processor.

The model consists of five blocks. From left to right, they are:

1) IComport3: Gets data from the PC.
2) Data type conversion 1: Convert input data to float.
3) Magnitude FFT: Calculate the power spectrum of input.
4) Data type conversion 2: Convert input to int32.
5) Ocomport3: Send data to PC.

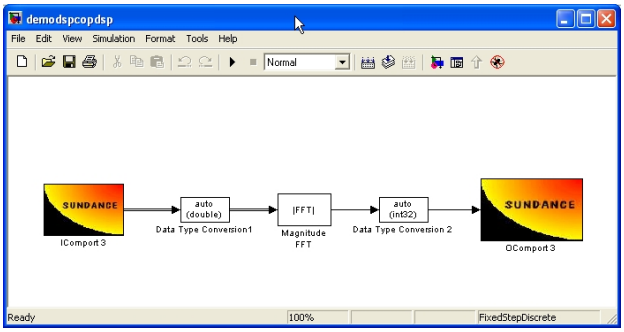

Fig. 1. DSP Coprocessor model

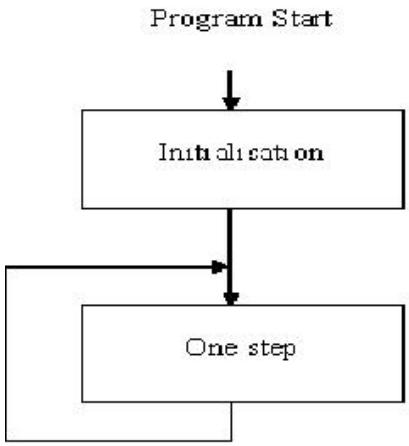

Fig. 2. Flowchart of the generated code

B. Simulation

During this phase, the system response is simulated and errors could be detected and fixed. Simulink call each block at the appropriate time to simulate the system. Simulink also checks that the data flows between blocks are correctly handled. For example, if a block expects an integer data as its input but is fed with floats, Simulink would generate an error and inform the developer that there is a data flow error. The designer can change the model if the system response is not in the acceptable region and retest the simulation to find a new response. The iterations continue until the system response become acceptable.

C. Code generation

When the designer has satisfied with the system response, he can use SMT6050 to generate code. Since Simulink tested the developed model during simulation and designer checked the system response, the code generation normally would pass without errors. SMT6050 generates the source code and necessarily files needed for compiling, linking and building the executive binary file for uploading to the DSP board. For example, it generates a linker command file and a make file to use make utility for compiling and linking the generated code and a batch file to set the required environment variables needed by make file and TI compiler and linker. During code generation, Matlab run the generated batch file to produce the output file ready to be uploaded into a Sundance DSP board.

III. Code generation method

SMT6050 analyse the model and generates code that contain two main parts: Initialisation and one-step as shown in fig 2. In the initialisation subroutine, code is generated to initialise each block in the model. Initialisation is called only once at the beginning of program run. The one step is responsible to generate the required response of the system. In the one step subroutine, SMT6050 generates code that reproduces the functionality of each block in the model. The generated code for the selected model would follow the structure of the graph shown fig 3.

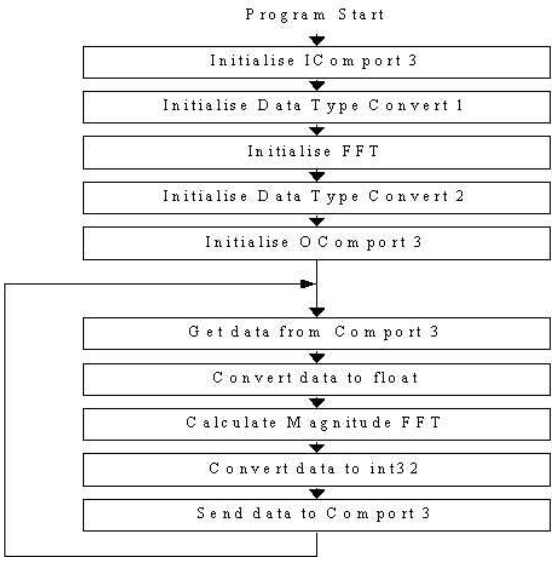

Fig. 3. Detail flowchart of the generated code

IV. Optimisation technique

Since initialisation codes runs only one time but the one-step subroutine calls repeatedly, optimisation effort focused on one step subroutine.

SMT6050 generates code for one-step using information that is passed to it via a TLC file for each block. Each block in the model should have a corresponding TLC file. The TLC file instructs the SMT6050 on how to generate the code for a block. TLC files for the blocks shipped with Simulink have directives to generate code compatible with ANSI C compilers. The generated codes are very effective but since they generated based on a TLC that is targeted for general ANSI C compilers they do not use the internal parallel structure of TI DSP processors. To solve this problem, we developed a new set of blocks for Simulink. The TLC for these blocks instructs SMT6050 to generate calls to GDD DSP libraries. GDD libraries are a set of hand optimised assembly subroutines for TI DSP processors. The library functions intensively use the internal parallel structure of TI DSP processors to effectively process the input data.

A. GDD Library specifications

The GDD libraries were designed to implement the most frequently required DSP vector operations and a set of routines for solving systems of linear equations and matrix eigenvalues computations.

Structure of the DSP library (GDD300)[12]:

- Transforms (FFT, Hartley)
- Filters, convolutions, windows
- Real and complex vector operations
- Complex scalar math
- Data conversions
- Random number generation

Structure of the Eigenvalue library (GDD8000)[12]:

- Real and complex generalized eigensystems
- Real and complex general matrix eigensystems
- Symmetric and hermitian generalized eigensystems
- Symmetric and hermitian eigensystems
- Tridiagonal symmetric eigensystems
- SVD and least squares solutions

Structure of the Linear Equations library (GDD7000)[12]:

- Real and complex general matrix linear equations
- Real and complex symmetric matrix linear equations
- Real symmetric positive definite matrix linear equations
- Hermitian matrix linear equations
- Hermitian positive definite matrix linear equations
- Real and complex band matrix linear equations
- Real positive definite symmetric band matrix linear equations
- Positive definite hermitian band matrix linear equations
- Tridiagonal matrix linear equations
- Tridiagonal symmetric matrix linear equations
- Real and complex Toeplitz linear equations
- Cholesky decomposition
- QR matrix decomposition
- SVD and least squares solutions

The functions in these libraries conforms to the C calling conventions and are C callable, however coding in C applied a number of restrictions that potentially reduce the effectiveness of the code, thus internally these functions were coded in assembly, that helped to overcome restrictions of the C language and use internal parallel structure of TI TMS320C6000 processors that are capable to execute up to eight independent instructions on each CPU cycle.

The main features of functions in the libraries are optimal instruction flow, interruptibility and reentrability (thread safe). There are two separate modules to work in little and big endian memory mode.

The architecture of TMS320C6000 allows for performing of different types of an assembly code optimisation. The most important of them are instruction partitioning, scheduling and software pipelining [13] [14] [15]. The other types of optimisation performed were data dependencies disambiguation, memory banks access conflicts removal, split-joint path resolving and loop unrolling.

Two special kinds of optimisation were specifically excluded and have not been used; these are loop iteration intervals of less than 6 cycles and multiple register assignments. Although they may increase performance of the code, they also restrict interruptibility of the final code. Instead, the preference has been given to the ability of the code to be interrupted on every CPU cycle, and the performance degradation connected with large iteration intervals has been worked around by loop unrolling.

V. Experiment and results

SMT6050 was used to generate code for the selected model, compile and link for the SMT376 Sundance TIM board with TI TMS320C6711 microprocessor running at 150 MHz clock rate and 256 MB of SDRAM [16]. The generated code was profiled to estimate the efficiency of the generated code.

The time that each block takes to run in one call to one-step subroutine is estimated based on DSP clock tick. The results are shown in table I.

The performance of the data input/output blocks (IComport3 and OComport3) of the Simulink model completely depends on the performance of the underlying hardware. The communication ports of the test bed platform (SMT376) are implemented on a Xilinx FPGA device. To start/stop data communication via the comports, the control program has to set up a number of registers in the FPGA memory space. After the control registers has been properly initialised, the communication process is being performed by the hardware. Thus, the data communication blocks IComport3 and OComport3 need not be software optimised.

Block name	DSP Clock Tick
IComport3	—-
Data Conversion 1	3115
FFT	348316
Data Conversion 2	13469
Ocomport3	—-

TABLE I
PROFILE RESULT FOR DEMO MODEL CONSTRUCTED WITH SIMULINK BLOCKS

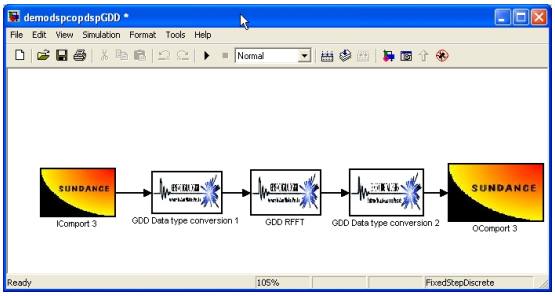

Fig. 4. DSP co-processor model constructed using GDD block set.

To generate optimised code, a new model that uses GDD block set was constructed. The new model has similar functionality compared to the previous model before but it uses GDD blocks instead of Simulink blocks. The model was changed as shown in fig.4.

SMT6050 generates code for this model based on GDD library calls and hence it is optimised for TI DSP processors.

The generated code for this model was profiled and the results are shown in the table II.

The time that each step takes in the optimised and non-optimised code is shown in the table III.

VI. Conclusion

Code generation using SMT6050 and MathWorks Real-Time Workshop was discussed and a technique based on using hand optimised assembly code such as GDD was presented. The structure of generated code was examined. It was shown that the generated code from a model that was constructed from GDD block sets was

Block name	DSP Clock Tick
IComport3	—-
Data Conversion 1	1750
GDD RFFT	30997
Data Conversion 2	3606
Ocomport3	—-

TABLE II
PROFILE RESULT FOR DEMO MODEL CONSTRUCTED WITH GDD BLOCK SET

Model	DSP Clock Tick
Non optimised	364597
Optimised	36353

TABLE III
COMPARISON BETWEEN THE PROFILE RESULT FOR THE TWO CONSTRUCTED MODEL

10 time faster compared to a similar model that was constructed from Simulink native block sets. The gain in speed comes from the fact that using GDD block sets force SMT6050 and Real-Time Workshop to generate codes that calls into an hand optimized assembly code. The other advantage of using this technique is that since GDD block sets provides with a simple and generic API, these codes can be easily ported to other platforms by linking against the respectively targeted GDD libraries.

REFERENCES

[1] W. Wolf, "CAD techniques for embedded systems-on-silicon," *16th International Conference on Computer Design*, pp. 24–29, Oct 1999.

[2] W.Wolf, "Household hints for embedded systems designers," *Computer (IEEE)*, vol. 35, no. 5, pp. 106–108, May 2002.

[3] M. Mrva, "Reuse factors in embedded systems design," *Computer (IEEE)*, vol. 30, no. 8, pp. 93–95, aug 1997.

[4] http://www.sundance.com/edge/files/productpage.asp?STRFilter=SMT6050.

[5] R. Obermaisser and P. Peti, "A framework for rapid application development of distributed embedded real-time systems," *EUROCON 2003*, vol. 1, pp. 80 – 84, Sept 2003.

[6] http://www.MathWorks.com.

[7] http://www.ti.com.

[8] *Real-Time Workshop user's manual (version 5)*, The MathWorks, Inc., 2002.

[9] http://www.sundance.com.

[10] *Simulink user's manual (version 5.1)*, The MathWorks, Inc., 2003.

[11] "http://www.dspshop.com/MiddleRoots.asp?Code=1."

[12] *GDD user's manual*, Sundance Multiprocessor Technology Ltd, 2002.

[13] "TMS320C6000 CPU and Instruction Set Reference Guide, rev. F," Texas Instrument, Tech. Rep. SPRU189F, October 2000.

[14] "TMS320C6000 Optimizing Compiler User's Guide, rev K," Texas Instrument, Tech. Rep. SPRU187K, October 2002.

[15] "TMS320C6000 Programmer's Guide, rev F," Texas Instrument, Tech. Rep. SPRU198F, February 2001.

[16] Sundance, "SMT376 DSP hardware user guide."