Chapter 8

Real Time and Embedded Implementation of Hybrid Algorithm

Since the complete implementation of the hybrid algorithm proposed here has a great computational complexity, it is necessary to test the possibility of implementing it in a real time and embedded environment. As the developed algorithm is at a primary research level, it is needed to perform tests on a flexible platform that allows the implementation of the non-optimized algorithms with a reasonable effort. The algorithm is first tried for real time implementation on PC using SIMULINK. For dedicated hardware implementation the DSP platform using 32-bit floating point processor TMS320C6713 from Texas Instruments is selected. DSK 6713 from Spectrum Digital Incorporation is used for implementing algorithm on the TMS320C6713 DSP. The Code Composer Studio Integrated Development Environment version 3.3 (CCS IDE V3.3) from Texas Instruments is used as compiler and debugger. This tool is invoked from MATLAB using RTW and Target Support Package TC6 toolboxes. Various profiling results are obtained and compared in this chapter.

8.1 Typical Setup for Developing Models

Figure 8.1 presents a block diagram of the typical setup for developing models, along with the input and output connected to the C6713 DSK [1].

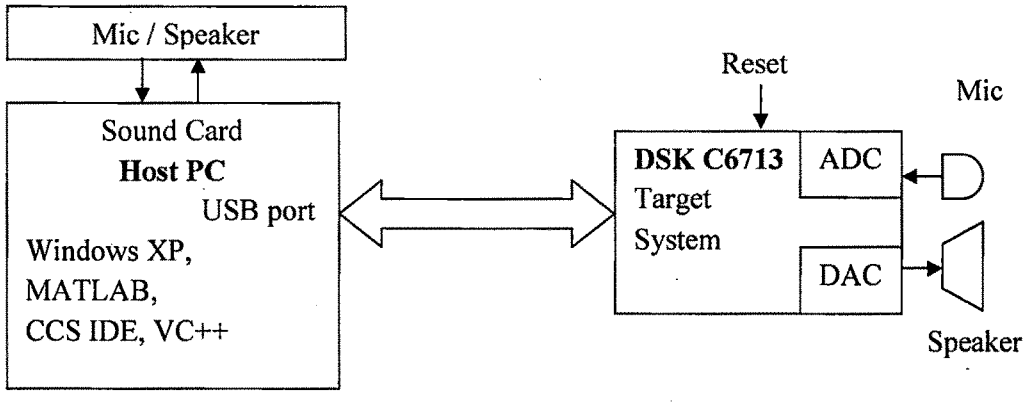

Fig. 8.1 Typical hardware and software setup for developing models

8.2 Real Time Implementation of Hybrid Approach on PC

Figure 8.2 presents a block diagram of real time PC implementation of hybrid algorithm. The data buffering and windowing, hybrid algorithm and overlap-add blocks represent sub-systems used to implement the overall speech enhancement system. In the set up the audio device (microphone) catch up the noisy speech signal from real environment. It digitizes the monophonic speech signal with 8 KHz sampling rate and 16 bits/sample resolution. The

buffering and windowing block frames the incoming data into a frame of 256 samples with 50% overlap and windowed using Hamming window. The RASTA algorithm is non-causal and requires future frames for filtering; which throws the challenge for real time implementation. So a sub-frame concept is used to overcome it. Here the framed data is divided into four sub-frames each consists of 64 samples. The matrix concatenate block is used to make a 64 x 4 data block from four 64 x 1 sub-frame. It is shown in figure 8.3. After proper framing the 256 point FFT is taken and from complex spectrum the magnitude is taken for further processing and phase is given for reconstruction with enhanced magnitude. The hybrid algorithm sub-system performs the speech enhancement operation using the combine RASTA and STSA approach described in chapter 6. Figure 8.4 shows the internal blocks of the sub-system. The entire hybrid algorithm is incorporated as an embedded MATLAB function. Finally from noisy phase and enhanced magnitude the enhanced complex spectrum is obtained for a frame. After 256 point IFFT operation, the overlap-add synthesis is performed to reconstruct the signal in time domain. Figure 8.5 shows the internal blocks to obtain overlap-add synthesis. The enhanced speech can be heard on speaker or headphone. The amplitude of output speech can be controlled by setting the gain value in the block before the wave device block (speaker/headphone).

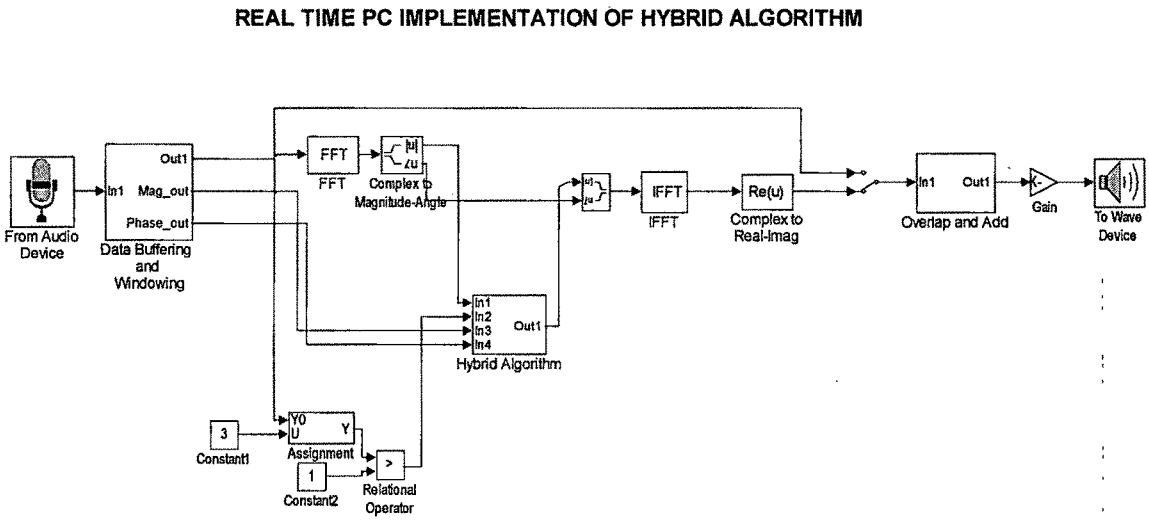

Fig. 8.2 SIMULINK block for real time implementation of hybrid algorithm on PC

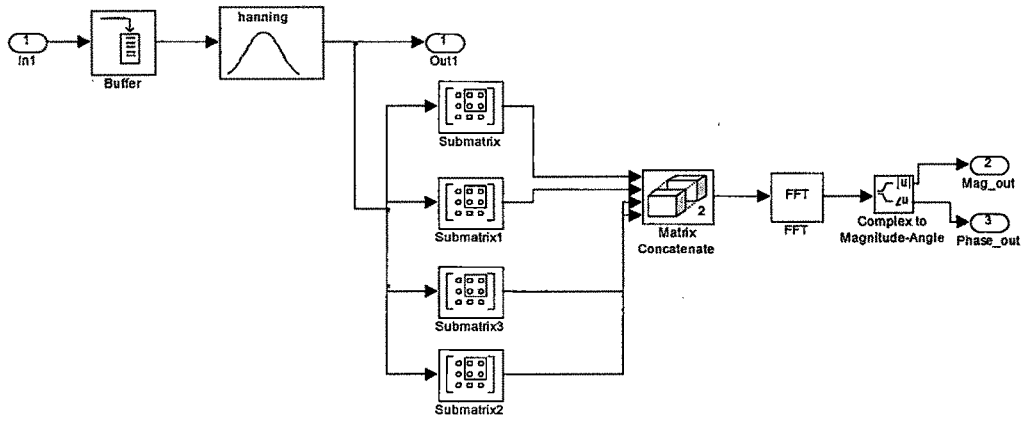

Fig. 8.3 Internals of sub-system data buffering and windowing

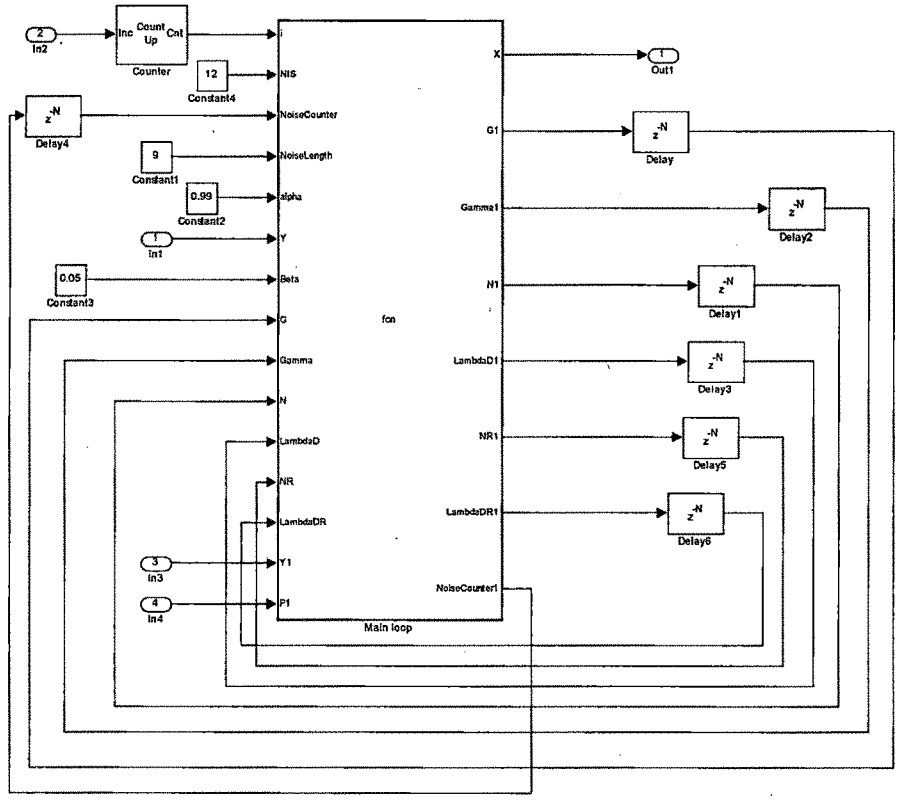

Fig. 8.4 Internals of sub-system hybrid algorithm

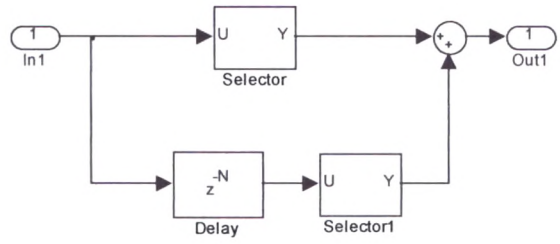

Fig. 8.5 Internals of sub-system overlap-add synthesis

Figure 8.6 shows the time domain waveform of clean speech, noisy speech corrupted by airport noise of 5dB and enhanced speech using the real time hybrid algorithm. It is self explanatory from this figure that the background noise is completely eliminated from the speech.

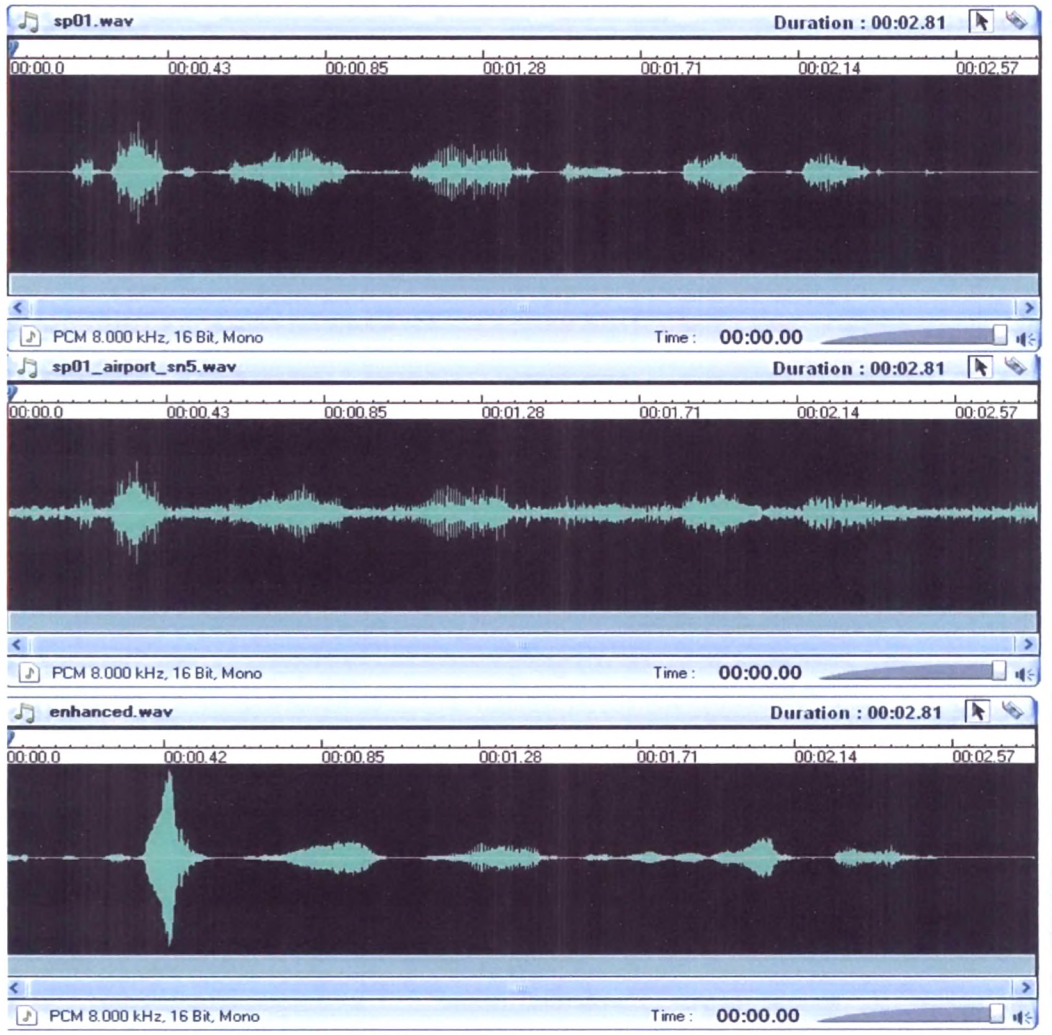

Fig. 8.6 Waveforms of clean, noisy and enhanced speech using real time hybrid algorithm

8.3 SIMULINK Profile Results

The Profiler allows running a program and then looking at how long each block took to execute. The profiler captures data while the model runs and identifies the parts of model requiring the most time to simulate. With this information one can concentrate on optimizing the sections of code that take up the most time. Figure 8.7 shows the SIMULINK profile results for the hybrid algorithm.

Simulink Profile Report: Summary

Report generated 20-Jun-2011 16:56:31

Total recorded time:	5.90 s
Number of Block Methods:	50
Number of Internal Methods:	0
Number of Nonvirtual Subsystem Methods:	3
Clock precision:	0.00000005 s
Clock Speed:	2166 MHz

To write this data as MMSE_RASTA_PCProfileData in the base workspace click here

Function List

Name	Time		Calls	Time/call	Self time		Location (must use MATLAB Web Browser to view)
sim	5.89683780	100.0%	1	5.89683780000	0.00000000	0.0%	MMSE_RASTA_PC
ModelInitialize	3.80642440	64.6%	1	3.80642440000	3.80642440	64.6%	MMSE_RASTA_PC
ModelTerminate	1.15440740	19.6%	1	1.15440740000	1.15440740	19.6%	MMSE_RASTA_PC
ModelExecute	0.93600600	15.9%	1	0.93600600000	0.04680030	0.8%	MMSE_RASTA_PC
MMSE_RASTA_PC (Output)	0.87360560	14.8%	189	0.00462225185	0.03120020	0.5%	MMSE_RASTA_PC
MajorOutputs	0.87360560	14.8%	189	0.00462225185	0.00000000	0.0%	MMSE_RASTA_PC
MMSE_RASTA_PC/Hybrid Algorithm/Main loop (Output)	0.73320470	12.4%	126	0.00581908492	0.01560010	0.3%	MMSE_RASTA_PC/Hybrid Algorithm/Main loop
MMSE_RASTA_PC/Hybrid Algorithm/Main loop/ SFunction (Output)	0.71760460	12.2%	126	0.00569527460	0.71760460	12.2%	MMSE_RASTA_PC/Hybrid Algorithm/Main loop/ SFunction
MMSE_RASTA_PC/Data Buffering and Windowing/Complex to Magnitude-Angle (Output)	0.04680030	0.8%	126	0.00037143095	0.04680030	0.8%	MMSE_RASTA_PC/Data Buffering and Windowing/Complex to Magnitude-Angle
MMSE_RASTA_PC/Noisy Audio Source/Random Source (Output)	0.01560010	0.3%	63	0.00024762063	0.01560010	0.3%	MMSE_RASTA_PC/Noisy Audio Source/Random Source
MajorUpdate	0.01560010	0.3%	189	0.00008254021	0.01560010	0.3%	MMSE_RASTA_PC
MMSE_RASTA_PC/Overlap and Add/Sum (Output)	0.01560010	0.3%	126	0.00012381032	0.01560010	0.3%	MMSE_RASTA_PC/Overlap and Add/Sum
MMSE_RASTA_PC/Magnitude-Angle to Complex (Output)	0.01560010	0.3%	126	0.00012381032	0.01560010	0.3%	MMSE_RASTA_PC/Magnitude Angle to Complex
MMSE_RASTA_PC/Manual Switch/SwitchControl (Output)	0.01560010	0.3%	126	0.00012381032	0.00000000	0.0%	MMSE_RASTA_PC/Manual Switch/SWitchControl
MMSE_RASTA_PC/Hybrid Algorithm/Counter (Output)	0.01560010	0.3%	126	0.00012381032	0.01560010	0.3%	MMSE_RASTA_PC/Hybrid Algorithm/Counter

Fig. 8.7 SIMULINK profile results of hybrid algorithm

The model initialize, terminate and execute times are not a major concerned for real time implementations. The Main loop of the hybrid algorithm occupies the majority of execution time as expected. Form figure 8.7 it is 12.4%. The other blocks require comparatively less time. Hence the main loop function of the hybrid algorithm is the major concern for embedded implementations. It can be concluded here that with the given complexity of the hybrid algorithm it is suitable for real time implementation on PC. It is interesting to see the same profiling results when the algorithm is downloaded on dedicated hardware.

8.4 Real Time Implementation of Hybrid Approach on DSK6713

Figure 8.8 presents a diagram for real time implementation of hybrid algorithm on DSK6713. It differs from the previous block only by I/O which is C6713 DSK ADC and DAC here. The link and procedure for downloading this model on the kit has been already described in chapter 7.

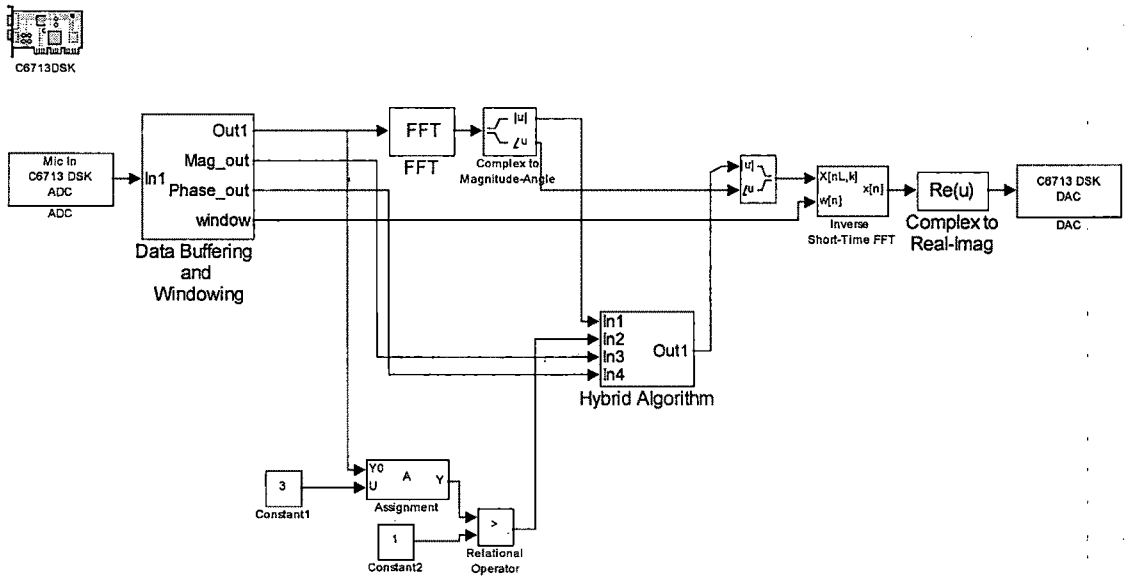

Fig. 8.8 SIMULINK block for real time implementation of hybrid algorithm on DSK6713

8.5 Profiling Results for DSK 6713 Implementation

Target Support Package TC6 software [2] supports DSP/BIOS features as options when code is generated for target and some ways it is possible to use the real-time operating system (RTOS) features of DSP/BIOS in the application. As a part of the Texas Instruments eXpressDSP™ technology, TI designed DSP/BIOS to include three components:

- DSP/BIOS Real-Time Analysis Tools — these tools and windows within Code Composer Studio IDE are used to view program as it executes on the target in real-time.
- DSP/BIOS Configuration Tool — enables to add and configure any and all DSP/BIOS objects that used to instrument the application. This tool is used to configure interrupt schedules and handlers, set thread priorities, and configure the memory layout on DSP.
- DSP/BIOS Application Program Interface (API) — lets to use C or assembly language functions to access and configure DSP/BIOS functions by calling any of over 150 API functions. Target Support Package TC6 software uses the API to access DSP/BIOS.

These components can be linked into application, directly or indirectly referencing only functions that need for the application to run efficiently and optimally. Only functions that specifically reference become part of the code base. Others are not included to avoid adding unused code to the project. In addition, after adding one or more functions from DSP/BIOS, the configuration tool helps to disable feature that do not need later, letting to optimize the program for speed and size.

While generating code that includes the DSP/BIOS options DSP/BIOS objects become part of the generated code. With these in place the profiling option in Target Support Package TC6 software can be used to check the performance of application running on target, gauge performance and find bottlenecks. To generate code that includes DSP/BIOS options, the Target Preferences block must select DSP/BIOS from the Operating system list on the Board Info pane. By selecting profile real-time task execution in the RTW software options, it inserts statistics (STS) object instrumentation at the beginning and end of the code for each atomic subsystem in the model. After the code has been running for a few seconds on target, the profiling results from target can be retrieved and it displays the information in a custom HTML report. Code profiling works only on atomic subsystems in the model. By designating subsystems of the model as atomic, each subsystem is forced to execute only when all of its inputs are available. Waiting for all the subsystem inputs to be available before running the subsystem allows the subsystem code to be profiled as a contiguous segment. Nested subsystems are profiled as part of their parent systems—the execution time reported for the parent subsystem includes the time spent in any profiled child subsystems. When the model is configured to use single-tasking mode, all atomic subsystems in the model are profiled and appear in the report. However, all systems and

subsystems do run once before the program terminates. This allows obtaining profiling results for all systems. The following tasks compose the process of profiling the code generated.

1. Enable DSP/BIOS for the code.
2. Enable profiling in the Real-Time Workshop software.
3. Create atomic subsystems to profile in the model.
4. Build, download, and run the model.
5. Use profile to view the MATLAB profile report.

The report shows the amount of time spent computing each subsystem, including outputs and updates of code segments, and provides links that open the corresponding subsystem in the SIMULINK model. Following are the definitions of report entries.

- **System name**

  Provides the name of the profiled model.

- **Number of iterations counted**

  The number of interrupts that occurred between the start of model execution and the moment the statistics was obtained.

- **CPU clock speed**

  The instruction cycle speed of the digital signal processor.

- **Maximum time spent in this subsystem per interrupt**

  The amount of time spent in the code segment corresponding to the indicated subsystem in the worst case. Over all the iterations measured, the maximum time that occurs is reported here. Since the profiler only supports single-tasking solver mode, no calculation can be preempted by a new interrupt. All calculations for all subsystems must complete within one interrupt cycle, even for subsystems that execute less often than the fastest rate.

- **Maximum percent of base interval**

  The worst-case execution time of the indicated subsystem, reported as a percentage of the time between interrupts.

- **STS objects**

  Profiling uses STS objects to measure the execution time of each atomic subsystem. One STS object can be used to profile exactly one segment of code. Depending on how RTW generates code for each subsystem, there may be one or two segments of code for the

subsystem; the computation of outputs and the updating of states can be combined or separate.

Using the above mentioned settings the report obtained for DSK6713 implementation of the model is shown in figure 8.9.

Profile Report

Simulink model: MMSE_RASTA_DSK_BIOS.mdl

Target: C6713DSK

Report of profile data from Code Composer Studio (tm)

04-Aug-2011 13:25:37

Timing constants

Base sample time	16 ms
CPU clock speed[1]	225 MHz

Profiled Simulink Subsystems

System name	MMSE_RASTA_DSK_BIOS
STS object	stsSys8_OutputUpdate
Maximum time spent in this subsystem	301.5 ms (1884% of base interval)
Average time spent in this subsystem	60.68 ms (379% of base interval)
Number of iterations counted	498

Fig. 8.9 Profile report of real time implementation of hybrid algorithm on DSK6713

System name	MMSE_RASTA_DSK_BIOS/Hybrid Algorithm
STS object	stsSys5_OutputUpdate
Maximum time spent in this subsystem	286.7 ms (1791% of base interval)
Average time spent in this subsystem	45.95 ms (287% of base interval)
Number of iterations counted	498

System name	MMSE_RASTA_DSK_BIOS/Hybrid Algorithm/Main loop
STS object	stsSys4_OutputUpdate
Maximum time spent in this subsystem	286.2 ms (1788% of base interval)
Average time spent in this subsystem	45.48 ms (284% of base interval)
Number of iterations counted	498

System name	MMSE_RASTA_DSK_BIOS/ADC
STS object	stsSys0_OutputUpdate
Maximum time spent in this subsystem	15.9 ms (99% of base interval)
Average time spent in this subsystem	124.4 µs (0.777% of base interval)
Number of iterations counted	250

Fig. 8.9 Profile report of real time implementation of hybrid algorithm on DSK6713 (cont.)

System name	MMSE_RASTA_DSK_BIOS/Data Buffering and Windowing
STS objects	stsSys2_Output, stsSys2_Update
Maximum time spent in this subsystem	11.14 ms (69% of base interval)
Average time spent in this subsystem	10.83 ms (67% of base interval)
Number of iterations counted	499

System name	MMSE_RASTA_DSK_BIOS/STFT
STS object	stsSys7_OutputUpdate
Maximum time spent in this subsystem	2.774 ms (17.3% of base interval)
Average time spent in this subsystem	2.663 ms (16.6% of base interval)
Number of iterations counted	499

System name	MMSE_RASTA_DSK_BIOS/ISTFT
STS objects	stsSys6_Output, stsSys6_Update
Maximum time spent in this subsystem	1.2 ms (7.5% of base interval)
Average time spent in this subsystem	1.126 ms (7.04% of base interval)
Number of iterations counted	498

Fig. 8.9 Profile report of real time implementation of hybrid algorithm on DSK6713 (cont.)

System name	MMSE_RASTA_DSK_BIOS/DAC
STS object	stsSys1_OutputUpdate
Maximum time spent in this subsystem	69.51 µs (0.434% of base interval)
Average time spent in this subsystem	54.19 µs (0.339% of base interval)
Number of iterations counted	249

System name	MMSE_RASTA_DSK_BIOS/FRAME INDEXING
STS object	stsSys3_OutputUpdate
Maximum time spent in this subsystem	30.93 µs (0.193% of base interval)
Average time spent in this subsystem	14.79 µs (0.0924% of base interval)
Number of iterations counted	499

Notes

1. The CPU clock speed is assumed to be 225 MHz. If your board uses a different clock speed, then you must specify the correct CPU clock speed in the Target Preferences Block.
2. STS timing objects associated with subsystem profiling are configured for a host-side operation of 4*x, reflecting the numerical relationship between CPU clock cycles and high-resolution timer clicks. Therefore, STS Max, Total, and Average measurements are correctly reported in units of "instructions" or "CPU clock cycles".
3. This page is best viewed with the MATLAB Help Browser, which allows the system names to link to the corresponding subsystems in the Simulink model.

Fig. 8.9 Profile report of real time implementation of hybrid algorithm on DSK6713 (cont.)

Looking at the report the hybrid algorithm block occupies 284% average time of the base sample time. That is the constraint for the DSK6713 implementation of the same model which has no problem at all when runs on PC. The output speech obtained is obviously no longer as per the requirements. The algorithm needs some optimizations before its implementation on DSK6713. The comparison of both these implementations is shown in table 8.1.

Function/Block	**PC Implementation**	**DSP Implementation**
CPU clock speed	2166MHz	225MHz
	Average Execution Time	
Input	0.3%	0.78%
Data buffering/windowing	0.8%	6.7%
Hybrid algorithm (Main loop)	12.4%	284%
Overlap-add	0.3%	7.04%
Output	0.3%	0.34%

Table 8.1 Profile results comparison

8.6 CCS Profiling Results for DSK 6713 Implementation

To create an efficient application, it is needed to focus on performance, power, code size, or cost depending upon goals. Application code analysis is the process of gathering and interpreting data about factors that influence an application's efficiency. CCS IDE provides profile tool to help in analyzing the code [3]. These profiling are incorporated for use with a simulator (C6713 device cycle accurate simulator with little endian is used in the application), and will not function properly with a DSK hardware configuration. This activity measures the total cycles consumed by entire application and calculates the total code size of application. The settings for the same are described in [3]. Using this summary of the profiling of the program loaded in simulator is obtained and described in table 8.2. To optimize performance it is required to decrease stall cycles and increase hit ratio of various cache memories. However it requires a complex tuning process.

Event	Count	Percentage
Total Cycles	423782	
NOP cycles	26392	49.71
Stall Cycles	370688	87.47
L1P Stall Cycles	201339	47.51
L1D Stall Cycles	218253	51.50
Instructions decoded	45214	
Instructions executed	40015	88.50
Instructions conditioned false	5199	11.50
Execute Packets	32699	
Branches taken	6924	
Total Loads	2697	
Total Stores	6643	
Instruction cache references	19370	
Instruction cache hits	15979	82.49
Instruction cache misses	3391	17.51
Data cache references	9340	
Data cache reads	2697	28.88
Data cache writes	6643	71.12
Data cache hits	552	5.91
Data cache read hits	293	10.86
Data cache write hits	259	3.90
Data cache misses	8788	94.09
Data cache read misses	2404	89.14
Data cache write misses	6384	96.10
L2 cache references	14	
L2 cache data reads	0	0.00
L2 cache data writes	0	0.00
L2 cache instruction reads	14	100.00
L2 cache hits	1	7.14
L2 cache data read hits	0	0.00
L2 cache data write hits	0	0.00
L2 cache instruction hits	1	7.14
L2 cache misses	13	92.86
L2 cache data read misses	0	0.00
L2 cache data write misses	0	0.00
L2 cache instruction misses	13	92.86
L2 SRAM references	4497	
L2 SRAM data reads	14	0.31
L2 SRAM data writes	4376	97.31
L2 SRAM instruction reads	107	2.38

Table 8.2 CCS profile summary of hybrid algorithm

8.7 Summary

This chapter has described the real time implementation of hybrid algorithm on PC as well as DSK6713 through SIMULINK. The profiling results are obtained and described. For PC implementation the algorithm works fine and gives the real time enhanced speech output. But for DSK6713 implementation it is not the case. The enormous resources available on PC are responsible for the better performance. For DSK implementation as already indicated the main loop requires optimization as the execution can't be completed within base sample time. More powerful platform like media processor DM6437 may provide the desired result. Further optimization of the code can be done through the algorithm tuning process[1].

[1] A paper entitled "Real Time and Embedded Implementation of Hybrid Algorithm for Speech Enhancement" is accepted for presentation in IEEE World Congress on Information and Communication Technologies (WICT 2011) Co-organized by Machine Intelligence Research Labs (MIR Labs) and University of Mumbai, Mumbai.