ISSN 1392 – 1215 ELECTRONICS AND ELECTRICAL ENGINEERING 2007. No. 3(75)
ELEKTRONIKA IR ELEKTROTECHNIKA

T121 SIGNAL TECHNOLOGY
SIGNALŲ TECHNOLOGIJA

Embedded Microcontrollers Benchmarking using Sliding Window Algorithm

Ž. Nakutis
Department of Electronic and Measurement Systems, Kaunas University of Technology,
Studentų st. 50, LT-51368, Lithuania, tel.: +370 37 300898; e-mail: nakutis@ktu.lt

Introduction

Fast-to-market feature is a key for product or solution success in today‘s highly changing market. Design engineer or technical project manager is under constant stress to make a fast decision for key components and technologies selection. Microcontroller (MC) and its firmware development tolls are of the first importance in embedded systems design since they influence both hardware and especially time consuming firmware development. Skills and experience of designers together with price are the issues considered first of all. Technical requirements of the task are important as well. Number of inputs/outputs, integrated peripherals, memory, etc. can be acquired from datasheets. In real-time applications performance of MC is a decisive parameter. Specification of classical MIPS are seldom sufficient to answer the selection questions. Benchmarking is widely used in the practice of digital signal and general purpose processors to compare their performance. There are organizations and companies like Berkeley Design Technology, Inc. [1], Embedded Microprocessor Benchmarking Consortium [2] that produce paid benchmarking reports to assist designers in processor selection. Availability of large number of vendors and integration of DSP enabled features pushed towards benchmarking initiatives in the embedded MC sector also [3-5]. Benchmarking using standard algorithms is very convenient to compare different MC but is not always enough to answer the question if the particular MC will meet real-time requirements of a considered task. Thus, starter kits, evaluation boards and simulators are convenient to preliminary check the capability of MC to handle the task. Prototyping firmware portions can be coded in high level language (C most usually) and timing parameters roughly estimated.

In this paper I present case study of very common 8, 16 and 32 bit MC benchmarking using sliding window algorithm. Sliding window algorithm is often used for surface visual quality inspection data processing. The algorithm involves intensive RAM and ROM memory access, bit manipulation instructions and software cycles. Kernel of the algorithm is described bellow. The benchmark in this case is sampling frequency that can be processed in real time.

Sliding window algorithm

The algorithm accepts binary pixels (1 represents fault pixel, 0 – fault free pixel) at the input. The goal of the algorithm is to check if total sum of fault (black) pixels in the rectangular window composed of K columns and L lines exceeds preset threshold level W_{th}. The window then constantly shifts forward taking the new line into its scope and forgetting the oldest one. See Fig.1 to assist further explanation and notations. Physically window slide can be implemented by transporting the investigated object by the line scan camera. Sum of fault pixels

$$S_W = \sum_{i=1}^{L}\sum_{j=1}^{K} p_{ij} \tag{1}$$

must be updated after acquiring the new line of the object surface image. In the equation (1) p_{ij} is the pixel obtained after filtering of raw pixels generated by the line scan camera

$$p_{ij} = 1\text{, if } \sum_{n=1}^{N} s_{ni} > S_{th},\ p_{ij} = 0 \text{ otherwise;} \tag{2}$$

here s_{ni} is the n-th raw pixel in the i-th column, N is the size of segment in lines, S_{th} - segment threshold level.

Estimation of the pixels has to be done in every column *i*. Fig. 2 presents sequence of necessary calculations that must be performed by the embedded MC upon reception of a new raw line. Critical data path is marked with the solid line. It is evident that the bottleneck of the algorithm will be experienced upon reception of the N-th raw line. Indeed at this moment it has to be:

1. Updated sum $\sum s_{ni}$ in every column;
2. Comparison performed to assign p_{ij} value 1 or 0;

3. Calculated sum of pixels of recent line (sum of averaged pixels of all columns in the recent line);
4. Updated sum S_W, which includes:
   a. subtracting sum of the oldest line,
   b. adding sum of the recent line;
5. Comparing new S_W over the threshold level.

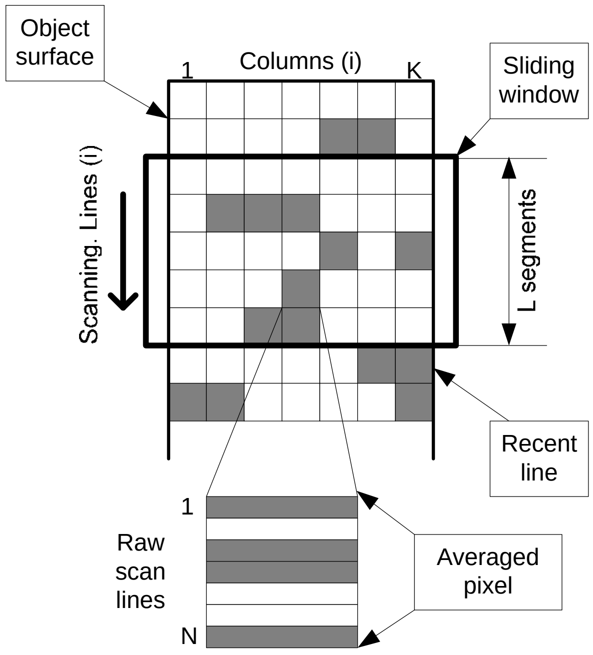

Fig. 1. Sliding window algorithm explanation

Benchmark of the sliding window algorithm in my investigation is the possible raw line scanning frequency which could be handled by the embedded MC in real time. Thus, frequency will be defined by the time necessary to perform all critical data path calculations by the embedded MC.

Sliding window algorithm implementation

Sliding window algorithm was implemented in high level programming language C. Simplified flow diagram of the algorithm is presented in the Fig. 1.

Raw data line of K bit length is the input to the algorithm (annotation 1). Packing block (ann. 2) is explained later. Blocks 3 and 4 are executed in cycle for every column from the first to K-th. Line index (ann.5) is used to track if an entire segment (N raw lines) was already acquired (ann. 6). Blocks 7 to 9 implement updating of sum *WndSum* of all pixels in the sliding window after the acquisition of new segment. Block 10 simply compares the sum value over threshold level and block 11 sets alarm bit for the further actions to be taken by the application. Block 12 clears temporary variables suck as column counters that will be used for the next segment processing.

Despite the common algorithm flow diagram there were implemented five versions of the algorithm (see Table 1). Even though producing the same output results given the same inputs, versions differ in their particular coding techniques. It was assumed that each data line is fed to the algorithm through 8 bit input port of an embedded MC. The length of line was set to K=16 (bit). Therefore, two 8 bit data units (in software development literature called variables) were seen at the input of the algorithm. Algorithms A1, A2 and A5 performed calculations manipulating these two separate 8 bit data units, while algorithms A3 and A4 packed (ann.2) two 8 bit variables to one 16 bit unit at the beginning and then operated with the single representation of the line. It was expected that embedded MC having internal architecture of 16 or 32 bit could exhibit better performance if the program data is manually optimized to meet CPU bit size.

Manual unrolling noted in the 3rd column of the Table 1 notifies the style of program cycles implementation. "No" means that cycles were coded using *for* cycle operator of the C language, while "Yes" means that *for* cycle operator was substituted by necessary number of code lines. For instance, cycle describing 8 assignments was manually changed to 8 consecutive assignments.

The difference between versions A2 and A5 is in that A5 uses special Mask array to check particular bit positions while A2 uses constants directly inserted in source lines.

To understand all algorithm implementation peculiarities source code can be freely obtained from the author.

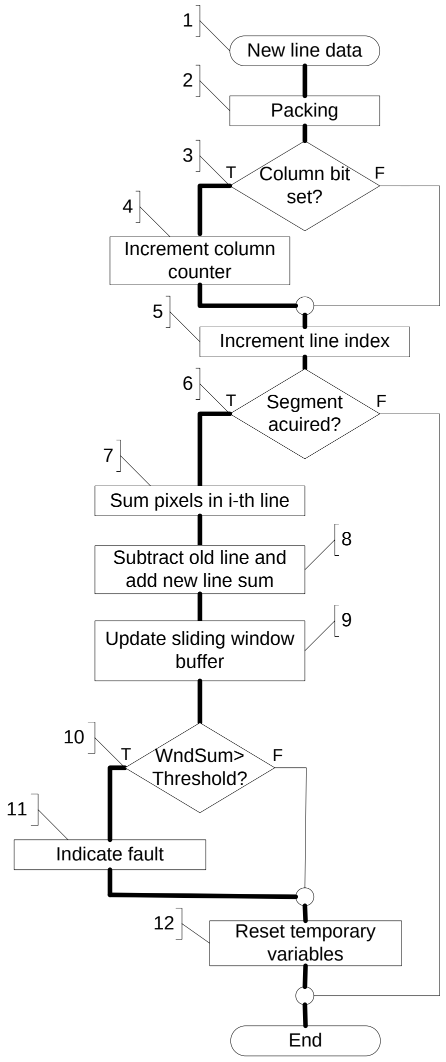

Fig. 2. Sliding window algorithm implementation flow diagram

Duration of the algorithm execution depends upon data. The worst case scenario takes place when all new line data bits are equal to 1. In this case all conditional statements in the algorithm result in true condition and MC needs to execute all operations. Otherwise, if i-th column data bit is equal to 0, then MC will skip increment of the column particular counter (see blocks 3 and 4 in Fig.2), etc. Therefore, internal variables of the algorithm were set in the manner to always force MC to follow critical path of the algorithm.

Table 1. Sliding Window algorithm implementations

Version	Bit	Manual unrolling
A1	8	No
A2	8	Yes
A3	16	No
A4	16	Yes
A5	8	Yes (but masks in array)

Experimental setups and tools

Three typical embedded MC with 8, 16 and 32 architectures were investigated in this work. The hardware targets and development tools used are listed in the Table 2.

Table 2. Embedded MC and tools

MC	Vendor	CPU bits	Hardware target & tools	Development tools
Atmega16	Atmel	8	STK500+ JTAGICE mkII	WinAVR release 20050214 and AVR studio 4
MSP430 F449	Texas Instr.	16	MSP-FET430 P440	IAR EW KickStart for MSP430 v.3.20
STR712F	ST Micro-electronics	32	STR712-SK + J-Link	IAR EW KickStart ARM v.4.40A

Algorithm execution duration was measured using oscilloscope Tektronix TDS2002 by probing discrete output which is toggled each time target MC accomplishes the analyzed algorithm version.

Benchmarking results

It is not only architecture and programming style define speed of the algorithm execution. Compiler settings and program memory type are also of significant importance. It is very common that compiler has options to enable optimization of C code while translating to assembler. Optimization feature usually has several levels and can target program size or speed. Optimization enables utilization of MC specific addressing modes as well as other tips like loop unrolling, function inlining, etc. Optimization features are available in both IAR and WinAVR compilers. In this research it was evaluated algorithm execution speed without compiler optimization and with turned on highest optimization level for speed (algorithm implementation version marked with asterisk * in result tables and figures).

All analyzed MC execute their programs from internal memory. Program of STR712F MC based on the ARM7 core can be loaded and executed either from RAM or Flash memory. Both options were investigated (see Table 3 and Fig. 4). Algorithm code execution from RAM was 1.3 to 1.5 times faster compared to the execution from Flash memory (evident from the Table 3).

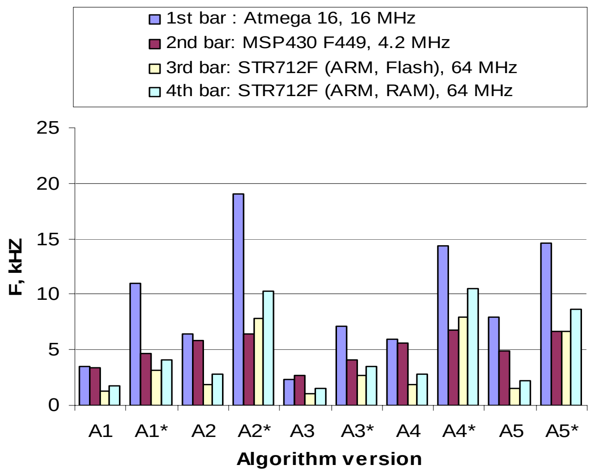

Fig. 3. MC real time frequency F (CPU clock freq. 4 MHz)

Table 3. MC real time frequency, kHz

Alg. Ver.	Atmega 16	MSP430 F449	STR712F (ARM, Flash)	STR712F (ARM, RAM	STR712F (Thumb, RAM)
Clock freq., MHz	16	4,2	64	64	64
A1	13	3	20	28	27
A1*	41	5	50	66	62
A2	24	6	30	45	44
A2*	71	6	125	165	121
A3	9	3	17	24	24
A3*	27	4	43	57	53
A4	22	6	30	45	44
A4*	54	7	126	168	117
A5	30	5	24	35	35
A5*	55	7	107	139	90

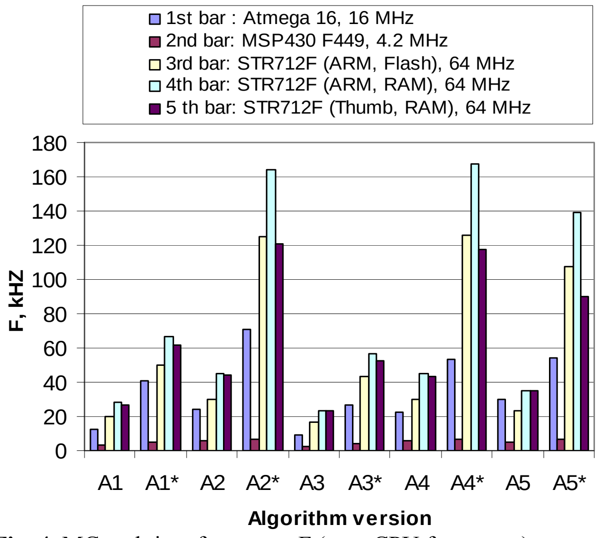

Fig. 4. MC real time frequency F (max CPU frequency)

In addition STR712F can operate in ARM mode (32 bit processor) or so called Thumb mode (16 bit processor).

Influence of these two possibilities was also considered and results presented.

Today's embedded MC internal clock frequency can be set by the software. To compare operation of considered MC I present results when all MC were set to operate from the equal clock frequency 4 MHz (Fig. 3). On the other hand, full processing power of the MC is utilized when it runs with the highest possible clock frequency. Table 3 and Fig. 4 give results when each MC is set to its upper clock frequency limit.

Conclusions

Given fixed 4 MHz CPU clock frequency 8 bit MC Atmega16 from Atmel Corp. outperformed 16 bit MC MSP430F449 from Texas Instruments Inc. and 32 bit ARM7 core based STR712F from ST Microelectronics by means of described sliding window algorithm execution time. The gain is around twice in case of most favorable algorithm implementation. However, STR712F features the highest throughput if every MC is set to operate at its top CPU clock frequency. The highest processable real time sampling frequency was up to 160 kHz.

The sliding window algorithm includes mostly bit manipulation instructions, conditional branching, intensive data and program memory access, but not high precision arithmetic. Due to this, architecture of MC (8, 16 or 32 bit) does not influence benchmark significantly.

Performed embedded MC benchmarking in combination with the corresponding development tools can be used to select the best algorithm implementation in order to optimize execution speed. Considering presented sliding window algorithm it can be seen that implementation versions A2*, A4* and A5* offer the highest speed despite the MC used (see Fig. 4).

References

1. **Insight, Analysis, and Advice on Signal Processing Technology.** Web address: http://www.bdti.com/index.html (viewed 2006-12-10).
2. **EEMBC** Embedded Microprocessor Benchmark Consortium. Web address: http://www.eembc.org/ /About/index.asp (viewed 2006-12-10).
3. **Embedded System Performance Comparisons.** Web address: http://www.freertos.org/PC/index.html (viewed 2006-12-10).
4. **A Review of Performance Analysis** (Benchmarking) Approaches for Embedded Microprocessors and Microcontrollers, Report by C. R. Powers, 1998. – Web address: http://www.ece.utexas.edu/~bevans/students/ms/ /chuck_powers/ms.pdf (viewed 2006-12-10).
5. **MiBench:** A free, commercially representative embedded benchmark suite. Web address: http://www.eecs.umich.edu/ /mibench/ /Publications/MiBench.pdf (viewed 2006-12-10).

Submitted for publication 2006 12 25

Ž. Nakutis. Embedded Microcontrollers Benchmarking using Sliding Window Algorithm // Electronics and Electrical Engineering. – Kaunas: Technologija, 2007. – No. 3(75). – P. 53–56.

Benchmarking using standard signal processing algorithms is very convenient to compare different embedded microcontrollers performance but is not always enough to answer the question if the particular microcontrollers will meet real-time requirements of the considered task. Sliding window algorithm used in materials surface quality visual inspection is described. Speed of this algorithm execution was tested on three common 8, 16 and 32 bit microcontrollers (Atmega16, MSP430F449 and ARM7 core based STR712F). Influence of different implementation versions of the algorithm in C language as well as compiler optimization settings upon execution speed were investigated. Applied approach enables to select the best algorithm implementation in order to optimize execution speed. Atmega16 microcontroller exhibited the fastest operation set 4 MHz clock frequency for all microcontrollers. However, STR712F featured the highest throughput if every microcontroller is set to operate at its top CPU clock frequency. Ill. 4, bibl. 5 (in English; summaries in English, Russian and Lithuanian).

Ж. Накутис. Оценка производительности встраиваемых микроконтроллеров используя алгоритм скользящего окна // Электроника и электротехника. – Каунас: Технология, 2007. – № 3(75). – С. 53–56.

Оценка производительности, используя стандартные алгоритмы обработки сигналов является полезным методом сравнивания встраиваемых микроконтроллеров, но не всегда достаточным для ответа на вопрос достаточна ли для выполнения обработки в реальном времени сигналов конкретной задачи. Описывается алгоритм скользящего окна часто используемый в задачах оптического контроля качества поверхностей. Скорость выполнения этого алгоритма была проверена с тремя обычными микроконтроллерами 8, 16 и 32 бит (Atmega16, MSP430F449 и STR712F на основе ARM7). Была исследована влияние реализации версии алгоритма на C языке и настройки опций оптимизации компилятора. Представленная методика может быть использована для выбора лучшей реализации алгоритма с целью оптимизации скорости алгоритма. Микроконтроллер Atmega16 превзошел остальных, когда все конкуренты работали на фиксированной тактовой частоте 4 МГз. Однако микроконтроллер STR712F быстрее всех выполнял алгоритм, когда все контролеры работали на своих допустимых тактовых частотах. Ил. 4, библ. 5 (на английском языке; рефераты на английском, русском и литовском яз.).

Ž. Nakutis. Įterptinių valdiklių našumo vertinimas naudojant slenkančio lango algoritmą // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2007. – Nr. 3(75). – P. 53–56.

Našumui vertinti naudoti standartinius signalų apdorojimo algoritmus yra patogu, kai reikia palyginti skirtingus įterptinius mikrovaldiklius, bet to ne visada pakanka atsakyti į klausimą, ar konkretus mikrovaldiklis tenkins realaus apdorojimo reikalavimus nagrinėjamam uždaviniui spręsti. Aprašytas slenkančiojo lango algoritmas, naudojamas paviršių optinės kokybės kontrolės uždaviniuose. Algoritmo vykdymo greitis buvo patikrintas tiriant tris įprastus 8, 16 ir 32 bitų mikrovaldiklius (Atmega16, MSP430F449 ir STR712F su ARM7 šerdimi). Buvo tirta skirtingų algoritmo realizavimo C kalba versijų ir kompiliatoriaus optimizavimo nustatymų įtaka algoritmo vykdymo greičiui. Pateikta metodika leidžia parinkti geriausią algoritmo realizavimą jo vykdymo greičio požiūriu. Atmega16 mikrovaldiklis greičiausiai vykdė slenkančiojo lango algoritmą, kai visų lyginamų mikrovaldiklių taktinis dažnis buvo lygus 4 MHz. Tačiau kiekvienam mikrovaldikliui veikiant maksimaliu leistinu taktiniu dažniu, STR712F mikrovaldiklis pasižymėjo didžiausiu našumu. Il. 4, bibl. 5 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).