Geosci. Instrum. Method. Data Syst. Discuss., 1, 1–29, 2011
www.geosci-instrum-method-data-syst-discuss.net/1/1/2011/
doi:10.5194/gid-1-1-2011

Discussions Geoscientific Instrumentation Methods and Data Systems

Martian dust devils detector over FPGA

E. de Lucas[1], **M. J. Miguel**[1], **D. Mozos**[1], **and L. Vázquez**[2]

[1]Universidad Complutense de Madrid, Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Madrid, Spain
[2]Universidad Complutense de Madrid, Departamento de Matemática Aplicada, Facultad de Informática, Madrid, Spain

Received: 1 November 2011 – Accepted: 3 November 2011 – Published: 1 December 2011

Correspondence to: E. de Lucas (enriquedelucas@pas.ucm.es)

Published by Copernicus Publications on behalf of the European Geosciences Union.

Abstract

Digital applications that must be on-board of space missions must accomplish a very restrictive set of requirements. These include energy efficiency, small volume and weight, robustness and high performance. Moreover these circuits can not be repaired in case of error, so they must be reliable or provide some way to recover from errors. These features make reconfigurable hardware (FPGAs, Field Programmable Gate Arrays) a very suitable technology to be used in space missions. This paper presents a Martian dust devil detector implemented on a FPGA. The results show that a hardware implementation of the algorithm present very good numbers in terms of performance compared with the software version. Moreover, as the amount of time needed to perform all the computations on the reconfigurable hardware is small, this hardware can be used more of the time to realize other applications.

1 Introduction

Digital systems for space applications have some special requirements not needed on normal systems. These requirements include energy/power efficiency, small volume, robustness and high performance as well as resistance under extreme conditions of pressure, temperature fluctuations, radiation and impacts. Traditionally, this has been got using embedded microprocessors and dedicated hardware peripherals. But many times the performance of these devices is not enough for the applications. Moreover, many applications have a high degree of internal parallelism that it is not used to improve the performance.

In general, hardware-based solutions provide better performance and less energy consumption than purely software solutions, since they eliminate the overhead due to instruction decoding and they include optimized hardware for the requested operations instead of carrying out those operations executing a sequence of predefined generic instructions. However, due to the exigent volume restrictions and the complex and expensive design process, it is not always feasible to implement all the needed

GID

1, 1–29, 2011

functionalities of an embedded system using only a hardware solution based on Application Specific Integrated Circuits (ASICs). Nevertheless, since embedded systems are targeting more and more complex applications, it is not likely that a software-based solution will achieve the requested performance. Hence, hardware accelerators for the most complex tasks are needed.

During the last 20 yr the hardware reconfigurable technology has evolved from small uniform devices, able to implement small circuits and statically reconfigurable, to very huge heterogeneous devices with capacity to be dynamically reconfigured. This evolution has produced FPGAs (the most popular type of reconfigurable hardware) which are used in a great number of applications, automotive circuits, digital image processing, video games, etc.

In the field of space applications one additional requirement is that all the hardware sent in space missions must be certified for space operation. This is because space-based systems must operate in an environment in which radiation effects have an adverse impact on integrated circuit operation Thomson (2005). Ionizing radiation can cause soft-errors in the static cells used to hold the configuration data. This will affect the circuit functionality and ultimately result in system failure. This requires special FPGAs that provide on-chip reconfiguration error-detection and/or correction circuitry. High-speed, radiation-hardened FPGA chips with million gate densities have recently emerged to support the high throughput requirements for space applications. In fact, radiation-hardened FPGAs are in great demand for military and space applications. For instance, companies such as Actel Corporation or Xilinx have been producing radiation-tolerant anti-fuse FPGAs for several years, intended for high-reliability space-flight systems. Actel FPGAs have been on-board more than 100 launches and Xilinx FPGAs have been used in more than 50 missions. For instance Xilinx (2003a,b) and Graham et al. (1999).

FPGAs are an intermediate solution between software developments (highly flexible, but power consuming and with performance problems) and ASICS (static applications, with good numbers in performance and power consumption) as we can see in Fig. 1.

GID
1, 1–29, 2011

FPGAs are now fully reconfigurable DeHon and Wawrzynek (2009); Hauck and DeHon (2008), a technological feature that allows a control station on Earh to adaptively select a data processing algorithm (out of a pool of available algorithms implemented on the FPGA) to be applied on-board. The idea is that FPGAs can be reconfigured on the fly. This approach is called run-time reconfiguration Resano et al. (2008). Basically the FPGA (or a region of the FPGA) executes a series of tasks one after another by reconfiguring itself between tasks. The reconfiguration process updates the functionality implemented in the FPGA, and a new task can then be executed. This time-multiplexing approach supports for the reduction of hardware components on-board since one single reconfigurable module can substitute several hardware peripherals carrying out different functions during different phases of the mission.

The flexibility provided by reconfigurable hardware can also be used to modify the functionality of the satellite instrumentation during the flight, or to automatically recover the system from malfunction. Moreover, the hardware design-cycle for FPGAs is much shorter than the one for custom integrated circuits, mainly because the design can be tested on the target platform since the first steps of the design process, thus avoiding a complex chip fabrication process.

In this paper we are going to present a space application that shows that the use of FPGAs is a good alternative in this kind of situations. The application is used to detect dust devils in the Mars surface. A dust devil is a hot whirlwind generated by a huge contrast between the atmospheric air and the surface in contact. This phenomenon has been studied in Earth and Mars Renno et al. (2000) and it is well known. We have to be able to detect a dust devil before it happens due to its destructive potential. This weather phenomenon can cause a great damage to instrumentation and human being. It is known that some big dust devils, about 1 km of height have been reported. In Mars some dust devils have been identified with a height over than 10 km of height. Apart from that, the friction caused by suspended particles brings a huge static load over dangerous levels.

GID
1, 1–29, 2011

The dust devils are detected as sudden changes in temperature and pressure. The need to have an automatic system which detects these phenomena is due to the distance between Earth and Mars. The key to propose a dust devils detector over a FPGA is that (1) an application like this is cheap in terms of space in a FPGA, (2) we can implement a lot of different applications in the FPGA at the same time, (3) we can reprogram it if we need and (4) there are FPGAs tested for space for other applications.

2 System description

To develop the design many ideas have been studied. We have studied several approaches to the STA/LTA algorithm and have implemented a software version in order to obtain a first approach to the solution. STA and LTA mean Short Time Average and Long Time Average respectively. The idea about have STA and LTA is because STA represents the recent changes and LTA represents the long time changes in some parameters. If we detect changes between the behavior of the temperature and pressure parameters in the short and large time we will declare an event that could be a dust devil. In order to achieve a better realism we used real data, from Mars Pathfinder mission, to test our design. Once the algorithm has been validated on software we have implemented a hardware design in a FPGA and a communication strategy between PC and FPGA.

2.1 Input data

In order to recreate appropriately the Martian environment we have used real data from Mars Pathfinder mission. In 1997 this mission which included a surface rover, the Sojourner, completed a successful landing on Mars, and for more than 3 months recovered data from the red planet.

Data from this mission can be accessed by anybody in NASA web page NASA (1997) where we can find Mars Pathfinder mission data in a tabulated text plain

GID
1, 1–29, 2011

format containing sensor lectures and date information. During the mission were detected certain number of dust devils Renno et al. (2000) that we use to test our system.

Every time Sojourner took a sample, it recovered data from its sensors. The period between two samples was very different, sometimes it was 4 s, and sometimes it was more than 1 h.

For our interest and with the purpose of dust devil detection, we needed to study data from temperature and atmospheric pressure, because these are the variables involved when a dust devil runs over surface. Specifically a dust devil produces a temperature rise and a pressure drop (see Fig. 2).

2.2 Algorithm

STA/LTA algorithms were developed for the first time by Lee and Stewart (1981) and therefore were improved for seism detection by Allen (1982). These algorithms are based on two sample sets, short time set (STA) and long time set (LTA).

With these two sample sets we can do different operations to achieve the current value (CV) and predicted value (PV) results of our algorithm, which are compared each other and if we get significant changes between them respect to a threshold value (THR) we will report an event (see Fig. 3).

2.3 Software approximation

As we said, firstly different possibilities has been studied on software about STA/LTA algorithms and their configuration parameters (number of STA and LTA samples and THR value) and once tested it, they have been implemented on a FPGA. To execute the variants of these algorithms are needed (1) a suitable storage structure for STA and LTA sets, (2) a specific implementation of primary characteristic function (PCF) and (3) the definition of THR value and its associated characteristic function.

The STA and LTA sets have a finite and static number of samples. We have decided to use a circular array to handle the refresh of the STA and LTA sets efficiently. That

GID
1, 1–29, 2011

circular array, which stores the samples as we see in Fig. 4, is a contiguous memory zone and two more variables to manage it, (a) an auxiliary pointer, that points to a position in memory where is stored the oldest element of the array, and (b) an integer, which represents the occupation level of the array.

The PCF represents the operations made with the samples. Usually the most used operation is the arithmetic average of each set but we can make more complex calculus. Other usual functions are root mean square and dispersion.

Finally we have the THR concept and its associated characteristic function (ACF). In our case we want to detect rise and descent events on temperature and pressure values. For that we have a rise THR (THR_R) and a descent THR (THR_D). To detect variations between STA and LTA sets we realize a division between STA average (STA_{AVG}) and LTA average (LTA_{AVG}) and then we compare its result with THR_U and THR_D as we show in Fig. 5. STA_{AVG} is equivalent to CV and LTA_{AVG} is equivalent to PV.

2.4 Hardware

Once tested different variants of STA/LTA algorithms we have implemented the hardware version. We have used a Virtex II pro to develop our designs and all the logic has been implemented with CLBs (Configurable Logic Blocks), see in Fig. 6 the board used to realize this implementation. Each CLB contains four slices Xilinx (2011).

2.4.1 Hardware Improvements

With a hardware design we can achieve a better performance than with a software design for several reasons.

A hardware design in a FPGA allows a high degree of parallelism. We can implement a system with a module for each magnitude without overhead on time execution. Also, with this alternative, we can have different modules with arithmetic average, root mean square, dispersion or any other functions as we want.

GID
1, 1–29, 2011

For all hardware algorithms we have taken the following design decisions.

1. We work with fixed point numbers. The reason is that the hardware required to execute integer division is not as expensive as float point division hardware. So we send data to the FPGA in fixed point format. Also the studied magnitudes such as pressure and temperature are presented in millibar and kelvin units respectively which allows us to work with unsigned integers.

2. We have defined a four byte wide data for each in-sample and return value.

3. In order to achieve a better performance we have implemented a circular FIFO RAM to access and compute data samples. This FIFO RAM has an integer occupation variable and a least recently used pointer to run.

4. As we said we have two sample sets, STA and LTA, being STA the set used to compute STA_{AVG} (STA average) and LTA used to compute LTA_{AVG} (LTA average). In order to calculate efficiently the arithmetic average and the root mean square we introduce an integer variable for each set which represents the sum of all values in each set. So when we introduce a new sample we do not have to sum all samples again, we only have to subtract the least recently used sample from this integer variable and add the new sample obtaining the sum of all samples at O(1) time.

$$\frac{a_1 + a_2 + \ldots + a_n}{n} \tag{1}$$

$$\frac{a_1 + a_2 + \ldots + a_n}{n} + \frac{a_{n+1} - a_1}{n} \tag{2}$$

$$\frac{a_2 + \ldots + a_n + a_{n+1}}{n} \tag{3}$$

5. Eventually we have studied the viability of change integer division by binary right shift. Each right shift of a integer represents a division by two, so if we can achieve

GID
1, 1–29, 2011

a number of elements power of two for STA and LTA sets we can make the division in a single clock cycle instead several. This produces an important reduction in area and execution time.

2.4.2 Hardware communication

The data to be analyzed has been sent from a PC to the FPGA. This communication has been performed through a serial port using RS232 protocol at 115 200 baud rate without hardware flow control, one stop bit and no parity. The reason to use this method is that we do not need a system which compute a huge amount of data in the minimum possible time, we need a system which compute a single data in the minimum number of cycles because the maximum frequency of new data is 1 Hz. So this strategy (see Fig. 7) is enough to our requirements. We have defined not only a sample four bytes frame to send data to the FPGA, but also for test purposes we have defined a four bytes return frame which contains execution information.

2.4.3 Arithmetic average version

This algorithm calculates STA_{AVG} and LTA_{AVG} (see Fig. 9), being the PCF the arithmetic average, and then execute the ACF (Associated Characteristic Function) which obtains the result of the algorithm, i.e. declares or does not declare an event. The core ACF is composed by a division between $STA_{AVG} \cdot 100$ and LTA_{AVG} (see Fig. 8). Then we compare this result with THR_R and THR_D and we declare or not the event as appropriate. The reason to multiply STA_{AVG} by hundred is that we are using fixed point arithmetic and we want two digits precision to compare with the THRs.

About cycles of execution we have to comment that since we receive a new sample until we finish the execution for that sample we consume 79 clock cycles. A big amount of that cycles are due to the two divisions of the algorithm (36 clock cycles per division). One division to calculate STA and LTA averages and other division to calculate STA average and LTA average quotient. So if we are able to eliminate these divisions we

GID
1, 1–29, 2011

would accelerate rather the whole algorithm. We cannot eliminate the second division because we cannot guarantee that LTA_{AVG} is a power of two, but the number of samples of STA and LTA sets can be power of two, it depends on if we are able to achieve a configuration parameters with this characteristic and which achieves our requirements.

Eventually we have to speak about FPGA utilization. The final report provided by Xilinx shows us an utilization of 29 % of slices with a number of samples of 64 for the STA set and 128 for the LTA set.

2.4.4 Arithmetic average optimized version

As we said if we can achieve a configuration with a number of samples power of two by each sample set we can replace the division by a right shift. This shift will be of x positions, being $x = \log_2(\text{number of samples})$. The algorithm is similar to the original algorithm being the only difference the substitution of the divider by a right shifter. Due to this right shift will be necessary a new initial stage to initialize the content of the FIFO RAMs before we start to divide. Only when we have fulled the FIFO RAMs we can start to divide otherwise we will obtain several incorrect initial values (up to number of LTA samples). With this variation we save 36 clock cycles. The final report provided by Xilinx shows us an utilization of 11 % (instead 29 %) of slices with a number of samples of 64 for the STA set and 128 for the LTA set.

2.4.5 Root mean square version

This variation of the algorithm calculates the root mean square of the samples. When we receive the new sample we calculate the square of it and this is what we send to the FIFO RAM of STA and LTA modules (see Fig. 10). The rest of the algorithm is the same as the arithmetic average algorithm. As we have to calculate the square of each sample we have introduced a multiplier in the design.

Each multiplier adds 5 clock cycles in parallel to execution time and due to that we need to store the new sample to calculate the square we add one more clock cycle.

GID
1, 1–29, 2011

Then we have an algorithm which can compute a new sample each $79 + 5 + 1 = 85$ clock cycles.

Regarding the FPGA occupation now we have two new 32-bit multipliers so the size of the design grow significantly. Each multiplier occupies 1088 LUTs of the FPGA. Due to the fact that we have got similar results with this algorithm and the first algorithm we have dismissed to implement it in hardware.

2.4.6 Dust devils detector version

This algorithm consist on two copies of the first algorithm (arithmetic average algorithm). One for each magnitude to analyze. In our case a module for pressure and other for temperature. The difference with the first algorithm is the final part of the ACF, it is, the condition to determine if we signal an event or not. In our study of the data with the software version of this algorithm we have obtained the configuration parameters contained on Table 1. These parameters are the number of samples for each set for pressure and temperature and the values for THR_R and THR_D.

As we can see we achieve a number of samples power of two for each sample set, so the dust devil detector will be implemented with the optimized version of the first algorithm. So we have the same number of clock cycles as the first algorithm, 79 for normal algorithm and 43 for optimized version. Regarding the FPGA occupation we have obtained a rate of 59 % of slices for normal version and 23 % for optimized version.

3 Results

In this section we have proceed to show the performance of the dust devil detector with the data provided and also we comment the pros of the hardware system versus a software system.

As we said the data provided are from Mars Pathfinder mission. In that mission were detected several dust devils which become this data set in a suitable data set.

GID
1, 1–29, 2011

Using these dust devils detected we have tunned our dust devils detector. The tunning process has been made with the software alternative and once achieved some suitable parameters we have proven these in the hardware system obtaining the results shown in tables three and four.

We can see that when we improve the pressure variations sensitivity we have an increment in the number of dust devil detections (Table 5), but we also increment the number of possible detections which are not reported in the original data as dust devils.

The software algorithm has been run over an Intel Q8200 Core2 Quad, with 2.33 Ghz frequency, 4 GB RAM at 1333 Mhz and Ubuntu 10.04LTS operative system. The FPGA clock used in this comparative has a frequency of 100 Mhz. The table two shows us the number of samples per second that the hardware and software algorithms can run. We observe that the performance of the hardware not optimized version is about 15 times higher than the software algorithm performance and the performance of the hardware optimized version is about 25 times higher. Also we obtain a decrease about 61 % in area with the hardware optimized version.

4 Conclusions

As we said, many applications have a high degree of internal parallelism which can not be used by traditional software solutions to improve the performance. A hardware design in a FPGA allows us to use that high degree of internal parallelism given that we can implement an algorithm with an independent module for each magnitude, even more, we can implement different algorithms and all without overhead on execution time. Other relevant aspect of use FPGAs is that the flexibility provided by reconfigurable hardware can be used to modify the functionality of the satellite instrumentation during the flight or to automatically recover the system from malfunction.

The results show that the hardware implementation of the algorithm present very good numbers in terms of performance compared with the software version (up to 25 times higher). Moreover, as the amount of time needed to perform all the computations

GID
1, 1–29, 2011

on the reconfigurable hardware is small, this hardware can be used more of the time to realize other applications. With these results and keeping in mind the difference between FPGA and PC clocks frequency we can conclude that the use of FPGAs to implement these algorithms is the best alternative.

Future work

In a real scenario the dust devils detector should be used to increment or decrease the measuring frequency of sensors. This would allows a system to do not lose events and to preserve power consumption. We would not need to have a high measuring frequency all the time to record all the events, but we would only need a minimum initial measuring frequency and capacity to modify that frequency in order to avoid the lost of events. Also, in order to minimize the false positives in the detection of events we could add to the system information about the normal values of the parameters involved in the detection. This will be possible as soon as we have an accurate characterization of the Martian surface layer Martínez et al. (2008) Meiga-Metnet (2000).

Acknowledgements. This research was supported by "Participación en la Misión a Marte Meiga-Metnet-Precursor AYA 2009-14212-C05-05/ESP", TIN2009-09806 and AYA2009-13300.

References

Allen, R.: Automatic phase pickers: their present use and future prospects, B. Seismol. Soc. Am., 72, 225–242, 1982. 6

DeHon, A. and Wawrzynek, J.: Reconfigurable computing: what, why, and implications for design automation, in: Proceedings of the IEEE/ACM Design Automation Conference, 610–615, 2009. 4

Graham, C., Petkovik, M., Russell, S., Seumahu, E. S., and Vesely, M.: The FedSat Microsatellite, http://www.itr.unisa.edu.au/projects/crcss/fedsat_microsatellite.pdf, last access: 12 October 1999. 3

GID

1, 1–29, 2011

Hauck, S. and DeHon, A. (Eds.): Reconfigurable computing: The theory and practice of FPGA-Based computation, Morgan Kaufmann, 2008. 4

Lee, W. and Stewart, S.: Principles and applications of microearthquake networks, Advances in Geophysics, Supplement 2, Academic Press, New York, USA, 293 pp., 1981. 6

Martinez, G., Valero, F., and Vazquez, L.: Characterization of the Martian Surface Layer, J. Atmos. Sci., 66, 187–198, 2008. 13

Meiga-Metnet: The next generation lander mission for martian atmosphere science, http://meiga-metnet.org, 2000. 13

NASA: The planetary Atmospheres Data Node: Mars Pathfinder Mission, http://pds-atmospheres.nmsu.edu/cgi-bin/getdir.pl?dir=document&volume=mpam0001, 1997. 5

Renno, N. O., Nash, A. A., Lunine, J., and Murphy, J.: Martian and terrestrial dust devils: Test of a scaling theory using Pathfinder data, J. Geophys. Res., 105, 1859–1865, 2000. 4, 6

Resano, J., Clemente, J. A., Gonzalez, C., Mozos, D., and Catthoor, F.: Efficiently scheduling runtime reconfigurations, ACM T. Des. Automat. El., 13, 58–69, 2008. 4

Thomson, J. T.: Rad Hard FPGAs, available at: http://esl.eng.ohio-state.edu/~rstheory/iip/RadHardFPGA.doc, 2005. 3

Xilinx: FPGAs aboard Mars 2003 Exploration Mission, http://www.xilinx.com/prs_rls/design_win/03104mars.htm, last access: 21 July 2003a. 3

Xilinx: Raytheon OPTUS space program, http://www.xilinx.com/prs_rls/design_win/03135optus.htm, last access: 25 September 2003b. 3

Xilinx: Virtex II Pro and Virtex II Pro X Platform FPGAs: Complete Data Sheet, http://www.xilinx.com/support/documentation/datasheets/ds083.pdf, last access: 30 June 2004. 7

GID

1, 1–29, 2011

Table 1. Algorithm parameters.

Parameter	Pressure	Temperature
No STA samples	8	4
No LTA samples	1024	32
THR_R	0.997	–
THR_D	–	1.01

GID
1, 1–29, 2011

Table 2. Hardware and Software comparative.

Arithmetic Average			
	Software	Hardware*	Optimized Hardware*
Samples freq. (Hz)	94 931.0070	1 327 311.9241	2 383 127.2727
Gain	1.0	13.9819	25.1037
Dust Devils Detector			
	Software	Hardware*	Optimized Hardware*
Samples freq. (Hz)	91 819.5936	1 327 311.9241	2 383 127.2727
Gain	1.0	14.4562	25.9543

* Results obtained supposing that we have a new data every time we need it.

GID
1, 1–29, 2011

Table 3. Test 1. Dust devils detected each sol.

Algorithm parameters Test 1		
Parameter	Pressure	Temperature
No STA samples	8	4
No LTA samples	1024	32
THR_R	0.997	–
THR_D	–	1.01
Results for Test 1		
Sol	Local Hour	Detected
25	13:10	Yes
25	13:53	Yes
34	09:52	No
34	11:32	Yes
34	11:38	Yes
38	12:32	Yes
39	11:31	No
39	13:47	Yes
49	11:02	Yes
52	12:03	No
55	14:19	Yes
60	10:09	No
62	12:31	No
62	12:34	Yes
62	14:06	Yes
68	11:42	No
68	13:29	Yes
69	12:54	Yes
70	14:25	Yes

GID
1, 1–29, 2011

Table 4. Test 2. Dust devils detected each sol.

Algorithm parameters Test 2		
Parameter	Pressure	Temperature
No STA samples	4	4
No LTA samples	1024	32
THR_R	0.998	–
THR_D	–	1.01
Results Test 2		
Sol	Local Hour	Detected
25	13:10	Yes
25	13:53	Yes
34	09:52	No
34	11:32	Yes
34	11:38	Yes
38	12:32	Yes
39	11:31	Yes
39	13:47	Yes
49	11:02	Yes
52	12:03	No
55	14:19	Yes
60	10:09	No
62	12:31	Yes
62	12:34	Yes
62	14:06	Yes
68	11:42	No
68	13:29	Yes
69	12:54	Yes
70	14:25	Yes

GID
1, 1–29, 2011

Table 5. Dust devils detected.

Test	Number of dust devils detected
Test 1	13 (of 19)
Test 2	15 (of 19)

GID
1, 1–29, 2011

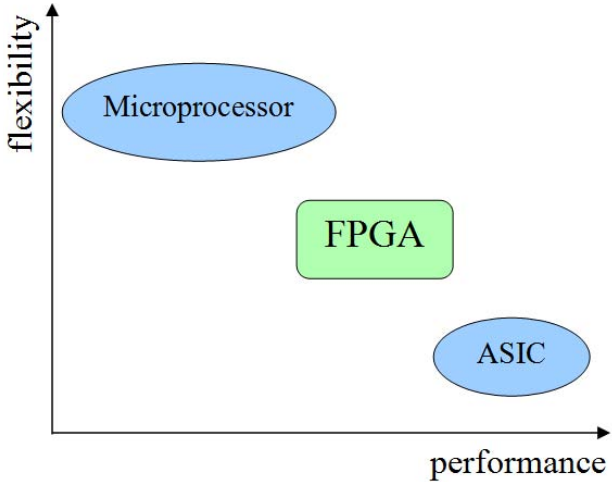


Fig. 1. Trade-off flexibility-performance for different technologies.

GID
1, 1–29, 2011

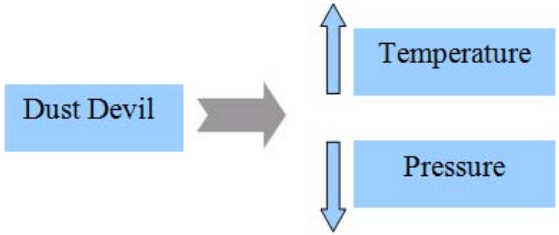


Fig. 2. A dust devil implies a temperature rise and a pressure drop.

GID
1, 1–29, 2011

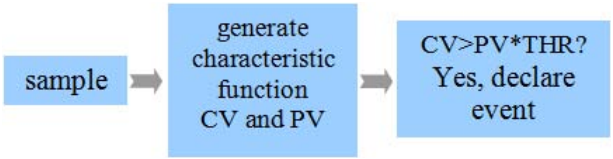


Fig. 3. Data flow diagram STA/LTA algorithm.

GID
1, 1–29, 2011

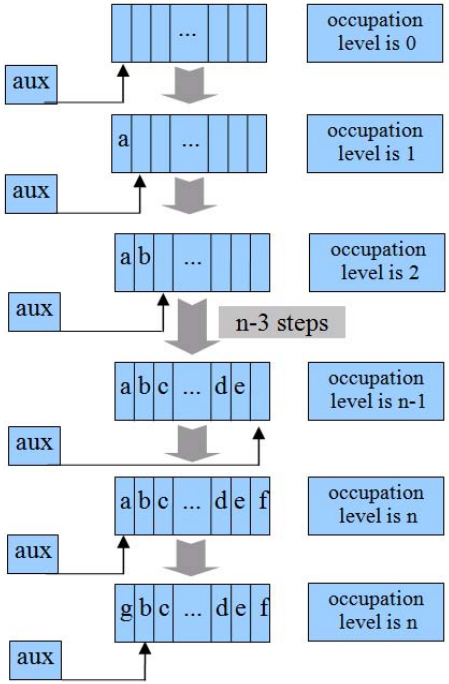


Fig. 4. Circular array occupation.

GID
1, 1–29, 2011

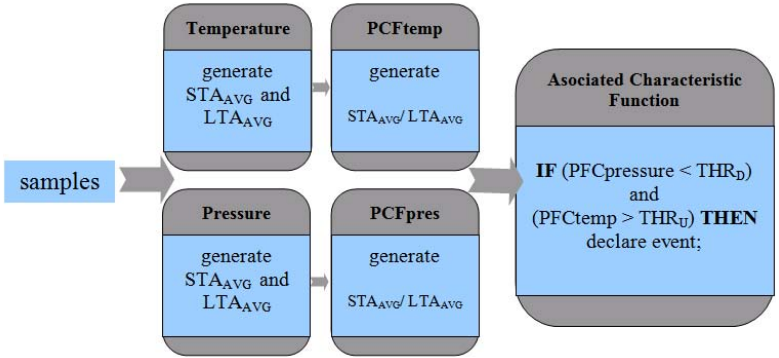


Fig. 5. Data flow diagram dust devils detector.

GID
1, 1–29, 2011

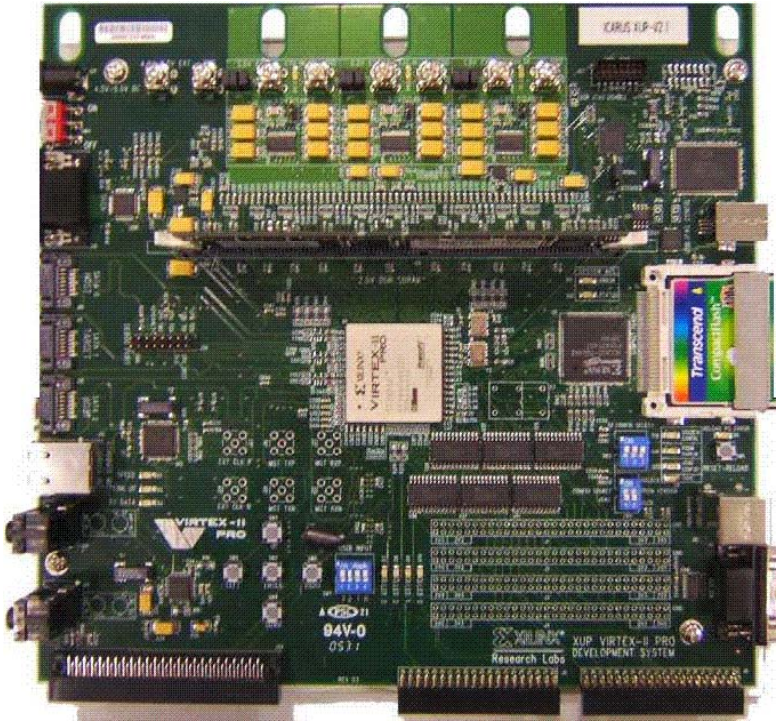


Fig. 6. XUPV2P30 Development System.

GID
1, 1–29, 2011

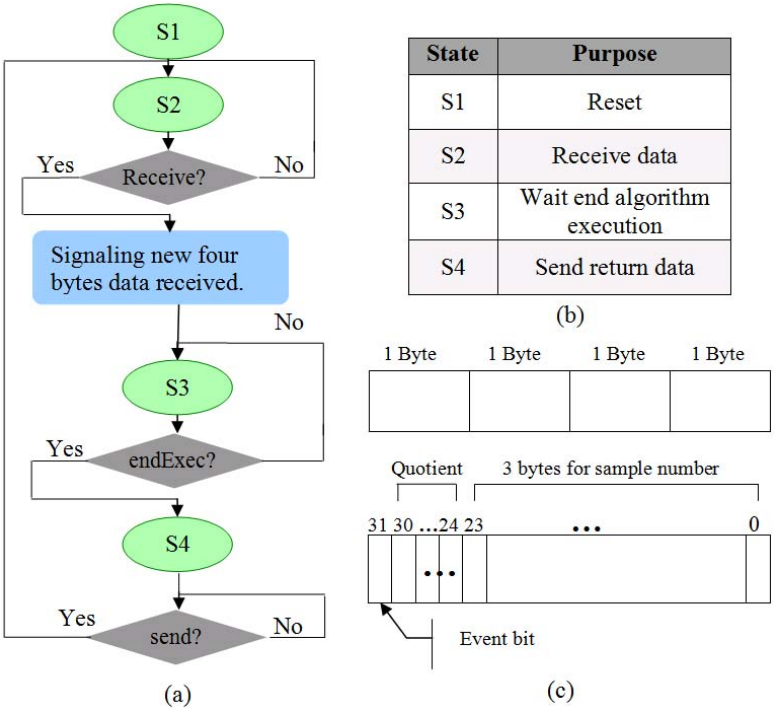
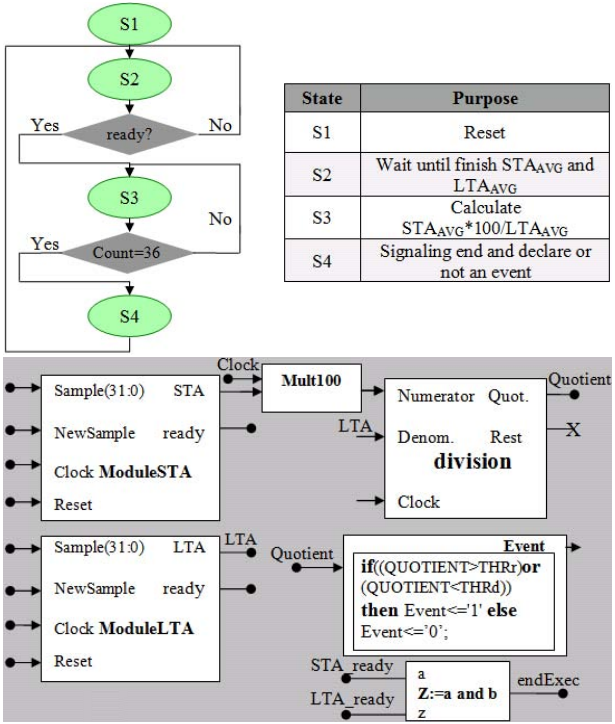


Fig. 7. (a) Data flow diagram of system's communication. **(b)** Table of states. **(c)** Frame sample and return value format.

GID
1, 1–29, 2011



State	Purpose
S1	Reset
S2	Wait until finish STA_{AVG} and LTA_{AVG}
S3	Calculate $STA_{AVG}*100/LTA_{AVG}$
S4	Signaling end and declare or not an event

Fig. 8. **(a)** Data flow diagram of arithmetic average version. **(b)** Table of states. **(c)** Interconnection.

GID
1, 1–29, 2011

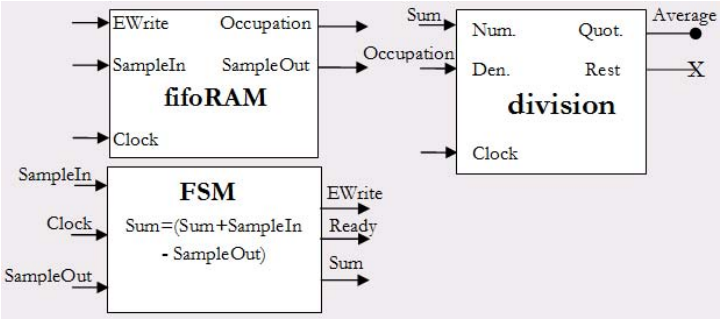


Fig. 9. Core Arithmetic Average version.

GID
1, 1–29, 2011

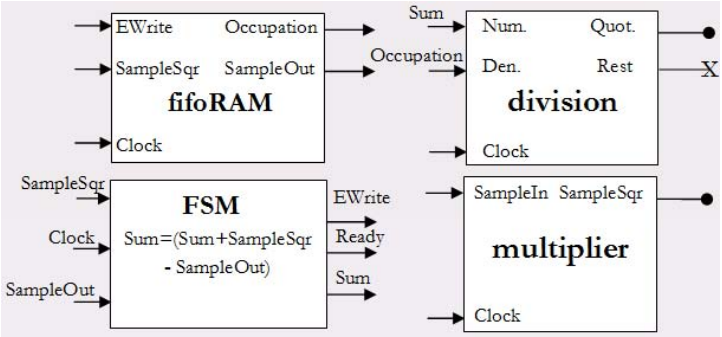


Fig. 10. Core Root Mean Square version.

GID
1, 1–29, 2011