FPGA Based Hardware Acceleration of Sensor Matrix

Abdul Mutaal Ahmad
German Research Center for Artificial Intelligence (DFKI)
Trippstadter Str. 122
67663 Kaiserslautern, Germany
abdul_mutaal.ahmad@dfki.de

Paul Lukowicz
German Research Center for Artificial Intelligence (DFKI)
Trippstadter Str. 122
67663 Kaiserslautern, Germany
paul.lukowicz@dfki.de

Jingyuan Cheng
Wearable Computing Lab
Technische Universität Braunschweig
Mühlenpfordtstraße 23
38106 Braunschweig, Germany
j.cheng@tu-braunschweig.de

ACM 978-1-4503-4462-3/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2968219.2968289

Abstract

This paper describes the hardware acceleration of various feature calculation functions used in activity recognition. In this work we have used a large scale sensing matrix which recognizes and counts gym exercises. Human activity is played on pressure matrix and the sensor data is sent to computer using a wired protocol for further processing. The recorded data from matrix is huge making it impractical to process on a smart phone. We propose a FPGA (Field Programmable Gate Array) based processing methodology which not only accelerates sensing data processing but also reduces the size of 2D sensor data matrix to 10 features. The resultant feature set can be transferred using wireless medium to a smart phone or other processing unit where the classification can be done. Our system takes a matrix of arbitrary size and output a 'features' set for each matrix frame. We used HLS (High Level Synthesis), an approach to write algorithm for FPGA using SystemC/C/C++ instead of traditional VHDL/Verilog. Results show promising improvement in processing time as compared to Matlab. Since the size of data is reduced, wireless medium can be use to transmit data. Additionally, the development time for FPGA designs is greatly reduced due to the usage of an abstracted high level synthesis approach. This system is currently developed for pressure sensing system but

this strategy can be applied to other sensing application like temperature sensor grid.

Author Keywords
Hardware acceleration; FPGA; high level synthesis; sensing; activity recognition

ACM Classification Keywords
I.2.m [Artificial Intelligence]: Miscellaneous

Introduction
Most sensing application consists of sensors attached to microprocessor. This system works well for reasonable number of sensors but as number of sensor increases, the high demand of processing or size of sensor data also increases. This created a bottleneck for large scale sensor matrix with high channel number to be processed in real-time. Also the wireless transmission is not feasible due to increase of size/frame. This forced the offline processing of sensor data.

Pressure sensing is one technique used to recognize human activities [12]. In this work we used a Smart-Mat which recognizes gym exercises and counts the number of exercises. The hardware (to record these exercises) consists of an 80 cm x 80 cm pressure sensor matrix. To analyze and process the data, it is sent to PC via wired serial port. The processing and analysis of these frames is done using standard feature extraction algorithms within Matlab. The details of the sensing hardware are explained in [15]. The size of single sample is quiet high, for example 80 x 80 matrix where each element is sampled with 24-bit ADC with a matrix sampling rate of 40Hz; makes it impractical to process using a smart phone in real time. Each frame consists of 2D array of 80 x 80 – 24-bit values. This created a barrier to connect large scale matrices to mobile phone or handheld device wirelessly where power and processing is limited. We considered external accelerators to speed the processing of sensor matrix frames and reduce the size of matrix into a reasonable size that can be sent via wireless protocol. There are 2 big issues in offline approach. First, the data produced by matrix is huge considering the size of matrix (for example 80 x 80). Second, the time consumed by sequential software to generate the required features is high. In this paper we suggest to use FPGA for this purpose. FPGA's are suitable for real time parallel processing of stream of data [3] [8]. FPGA contains abundant on-chip reconfigurable resources. These resources can be used depending on the level of parallelism required by the application. For example for image processing, each pixel of frame can be processed in parallel fashion to achieve high degree of speed-up as compared to traditional instruction based CPU execution. However complex and lengthy process of FPGA design restricted software developers and algorithm developers to use it for implementations. In this work we used 'high level synthesis' approach to move our algorithms to FPGA. We compared our results with standard Matlab running on server.
Overall this paper provides the following contributions.

1. *An approach that accelerates large scale sensing data processing:* Using this approach, the feature extraction and processing of various sensing application can be accelerated.
2. *A way that can be extended to use handheld device for large sensing matrix or grid*: In this approach we reduce the size of large matrix which is in thousands bytes to precisely 40 bytes/frame that only keeps the features. This

will enable the use of wireless medium to transmit this data to handheld device.

3. *A new design approach that can be integrated to sensing applications*: We proposed a high level synthesis to program FPGA. Using this approach the development time will be reduced and scientists, without having exact hardware knowledge, can use this approach to accelerate algorithms used in sensing applications.

Related Work

Existing work on sensing and activity recognition generally consists of sensing hardware which sample the data and then store it for processing [12]. The feature extraction algorithms are compute intensive and size of single frame is high. This issue restricted researchers to use only limited number of sensors. The work of [10] which consider several design options using sensors for wearable computing, suggests that the design space for complete system is a multidimensional problem. They propose 3-4 sensors attached to the system. Since the number of sensors is low, microprocessor is the most feasible choice for processing. The problem aggravates when number of sensors grow and the algorithm becomes complex. Same work in this area suggest to use GPU, as discussed in GravitySpace [1] where a high resolution pressure-sensitive floor is used to track and recognize activities. However for processing, SIFT [2] runs on a GPU, which provides a lot of programming flexibility but at the cost of power. GPU as general-purpose processor has very little acceptance in embedded world due to power reasons. Also the real-time processing cannot be kept intact and recorded data has to be stored to process it offline.

FPGA is already used in acceleration for feature based applications, as suggested in [5]. In this work, the author merged distributed feature detector with a rotational invariant feature descriptor and implemented on FPGA, achieving 15 x speed-ups over CPU. This is more evident in the work of dynamic vision sensors [7] where FPGA-based framework for event-based processing is used that allows uncorrelated-event noise removal and real-time tracking of multiple objects. High level synthesis, HLS, is already used in image processing as it provides flexibility in developing quickly for FPGA. For example the work in [9], in which a separate hardware library is developed for image processing function based on HLS.

Smart-Mat: The Pressure Sensing Matrix

Smart-Mat is resistive based pressure sensing matrix developed to recognizing and counting Gym exercises [15]. It consists of a hardware platform which has sensing matrix of thin layer of conductive polymer fiber sheet. This sheet is placed between 80 parallel stripes of conductive foil on each side making it a 80 x 80 matrix spaced at 1cm in direction [12].The sensing matrix is attached to three 24-bit ADC based FPGA system to sample and send data to computer over serial port at every 1/40 sec. The processing of these frames is done within Matlab. The processing system is divided into two major blocks. 1) Pre-processing system where the quality of image is improved by filtering, removing the DC component. The DC component is the average value which is calculated when the matrix was unoccupied. The DC component of the matrix has to be subtracted from every sample. Image filtering improves the quality of image. We applied 3 x 3 median to each frame. 2) The feature calculation system in which area A, weight W and pressure P are calculated from the processed matrix. Along with this, the 7 Hu's moments [6] are also calculated which are invariant to translation, scale and rotation. By doing this the whole matrix is reduced to a set 10 features. The details of different steps of processing and classification are described in our previous work [12]. This paper discusses the hardware implementation of feature calculation.

HLS- High Level Synthesis

High level synthesis has a new birth in FPGA design methodology with ever growing complexity in circuit and design explorations. Traditional design methodology involved HDL (hardware description languages) to describe hardware which requires a lot of hardware understanding and experience. This has been and till now a more safe choice for ASIC design. This has forced software designers to stick to high level languages to describe their algorithms. To speed-up algorithms, designers usually opt to move algorithms to GPU or use special high performance servers.

FPGA have been emerged recently as an alternative design options for off-loading compute intensive tasks to achieve high performance at low-power [11]. FPGA provides the flexibility of programmable hardware. With the latest advancement in chip technology, the density or the number of gates per unit area has increased dramatically which provides abundant number of resources on the chip. However, to utilize the computational power of FPGA a standard easy-to-use programming interface was lacking. This brought the HLS into the design methodology. HLS takes the abstraction of design entry to C/C++ level where a software engineer can write algorithms as if they are writing for CPU and the HLS compiles the code into hardware.

While using HLS, the design can be entered in C/C++ and can be verified completely using a C/C++ test bench. The HLS design methodology uses a twofold verification methodology. 1) Pre synthesis, which is validating the algorithm at C level. This makes sure that the algorithms work well functionally. This is a great advantage, since it reduces the time which is needed to right the RTL (Register Transfer Logic) of the same algorithm and then verify it at the RT level. 2) RTL verification which make sure that the algorithm produces the same result as in C validation at the RT level.

In HLS a C++ written function corresponds to a block of RTL in hardware description language with function arguments as inputs to module. All the logic inside the function is compiled by extracting the control and data path and creating a directed flow graph [13]. After compilation of such function, HLS tool gives estimates of resources, timing and throughput of the design.

Hardware Architecture of Smart-Matrix Processing System

Our hardware implementation consists of two sub-blocks i-e preprocessing system where we improve the quality of captured frame by removing DC component and noise. The other block is the matrix processing system where we calculate the features like area, weight and pressure. We also calculate the Hu's moments which describe the shape of contact area and weight distribution of the image. Figure 1 shows the block diagram of FPGA design. Our current implementation is based on the processing of these

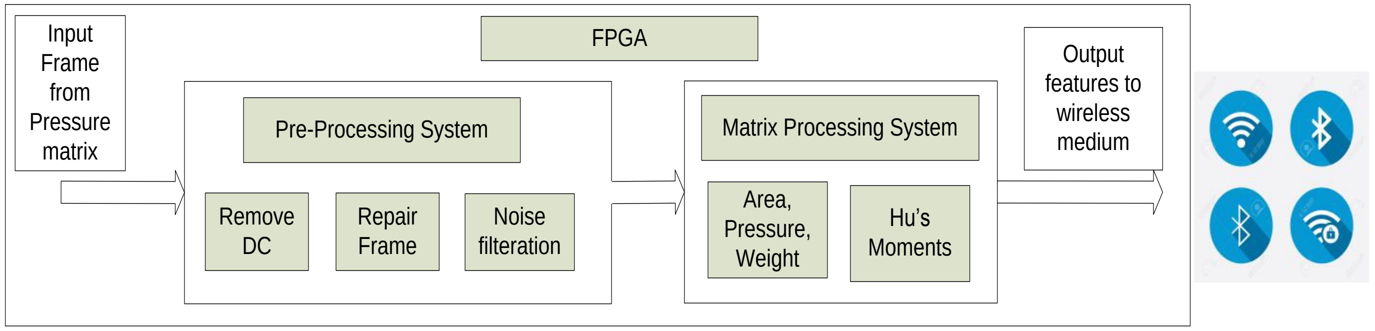


Figure 1 Block diagram of hardware design implemented using HLS

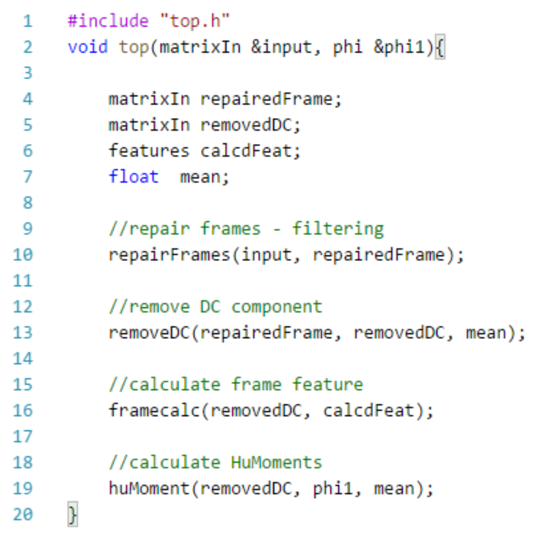
frames and calculations of their features. These two parts of our system have the most data processing in it. For example if we consider a matrix size of 80 x 80, and consider every sample a float then we have to process 80x80x4 bytes every 1/40 sec. However, the data size is reduced to 10 feature values of area, pressure, weight and Hu's 7 moments after processing. This resultant data can be processed easily within a microprocessor or can be transferred over a wireless medium (Bluetooth LE) to mobile where we can do the classification of this data. Our results will show architecture exploration in this direction.

Figure 2Figure 1 depicts the realization of our architecture in Vivado HLS tool. The top function contains sub-functions. The input to top-module is complete matrix and the output of the system is features. Each sub-module comprises of C/C++ {.cpp,.h} file which is compiled into individual hardware block.

FPGA Design Flow for Designed Architecture

Our hardware implementation works on individual frames since it would require a lot of FPGA memory to store individual frames in FPGA. We followed the following design flow. First we converted each function used in pre-processing and matrix processing to HLS compatible C++ functions. Each function is compiled into separate hardware block with inputs and outputs. After the conversion, block level testing is performed. This is done to make sure that individual blocks perform correctly. Second, these functions were combined into single top level function. Again we test overall system with the Matlab results. After functional verification, the design is synthesized into RTL. Now the testing is done at RTL level. The results of RTL verification and C++ validation can be slightly different in terms of accuracy.

During HLS, we defined a reference 100 MHz clock for our design. The HLS tool will try to meet this design requirement while utilizing maximum available resources in the selected FPGA. During HLS compilation we have provided the several resource constraints, as described in Table 2. In HLS terms they are known as 'Directives'. These constraints can be provided using #pragma in the design file or can be added as separate file.



```
#include "top.h"
void top(matrixIn &input, phi &phi1){

    matrixIn repairedFrame;
    matrixIn removedDC;
    features calcdFeat;
    float  mean;

    //repair frames - filtering
    repairFrames(input, repairedFrame);

    //remove DC component
    removeDC(repairedFrame, removedDC, mean);

    //calculate frame feature
    framecalc(removedDC, calcdFeat);

    //calculate HuMoments
    huMoment(removedDC, phi1, mean);
}
```

Figure 2 View of implemented design inside the HLS tool- A top level function with sub-functions

Each of the directive guides the HLS to restrict/use certain resources to achieve design requirements. In our design we have used floating point arithmetic because we want to keep the accuracy of results as high as possible.

In this work, we selected zynq devices which support Vivado HLS design flow. The details of the selected devices are shown in Table 1. In the results section, we provided a performance and power analysis with these two devices.

FPGA Feature	FPGA1	FPGA2
Device Name	Xc7z020clg484-1	xc7z045ffg900-2
Family	Zynq	Zynq
Package	Clg484	Ffg900
Speed	-1	-2
LUT	53200	218600
FF	106400	437200
DSP	220	900
BRAM	280	1090

Table 1. Details of resources of 2 different zynq 7000 FPGA devices used in this experiment

We validated our system at the C++level by taking samples from our recorded data. As already mentioned the test bench is written in C++ and verified completely at the C++ level. However the same test can be used to verify the RTL of the design. This is huge benefit of HLS, as the tool will automatically convert the C++ test bench into RTL test bench. This data was previously recorded with gym activity on Smart-Mat. We achieve 90-95 % (at C++ level) accurate results as compared to Matlab. We also verified our design at the RTL level which has 4% less accurate results as compared to Matlab. The inaccuracy happened because C++ and RTL have different library definition for the same function.

HLS Directive	Functionality
ALLOCATION	Control the usage of hardware instance e.g multiplier, adder, functional
PIPELINE	Add pipeline stages within loops
DATAFLOW	Ensures that task within function starts as soon as the data is available

Table 2 Details of 'Directives' used in the our design

Results

In this work we selected various matrix sizes for sensor matrix and calculated results of speed-up in terms of frequency or fps and compared with Matlab. A server with Xeon(R) CPU E5-2640 v2 @ 2.00GHz, 20 MB cache and 32 GB Memory was selected to run Matlab. The calculation of sampling frequency is shown in equation (1). The '*data rate*' is the actual speed of wireless channel or the processing rate of Matlab (to produce features). The '*sample_size*' depends on ADC. In this work we have used 24-bit ADC to sample sensor data.

$$Frequency\ (fps) = \frac{data\ rate\ \left(\frac{KB}{s}\right)}{maxtrix\_size\ \times sample\_size(B)} \qquad (1)$$

Figure 3 shows the comparison of Matlab fps with different wireless standards. This comparison explains that for small matrix size (5x5) the Matlab fps outnumbered wireless fps. Thus, it is not feasible to send this data over wireless module and then process is later on Matlab. On the other side, for higher matrix size Matlab fps lie under the supported wireless but real-time processing of frame cannot be supported since the Matlab fps dropped down drastically as matrix size increases. As it can be seen from Figure 3 that for small matrix size the design space is restricted by the speed of wireless link whereas the designs with higher matrix sizes the design space is restricted due to slow processing of frames by Matlab. We selected Bluetooth 2.0 and BluetoothLE for prospective wireless standards which are suitable for sensing applications because of low power requirements.

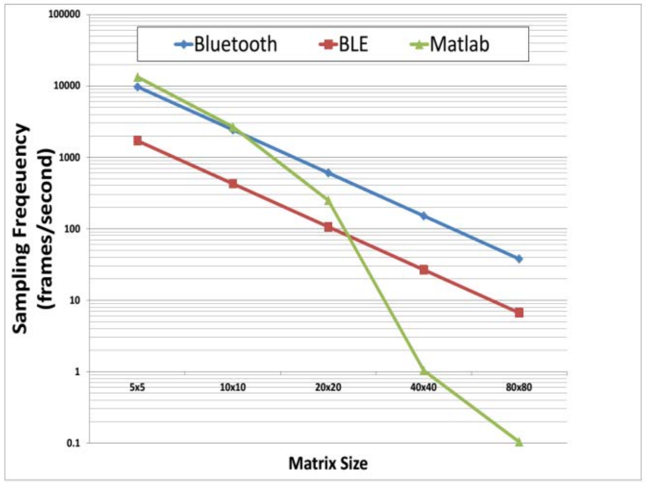


Figure 3 Comparison of frequency (logarithmic scale) between Matlab and wireless standards

The fps is directly dependent on data rate of either FPGA or Matlab, which in essence depends on how fast the data is processed per unit time. FPGA, on the other hand, processes the same amount of data in comparatively less time. Our initial results show tremendous increase in our FGPA based design as compared with Matlab.

Figure 4 shows the improvement while using two different FPGA's as an alternative processing unit compared to Matlab. In this scenario the sensor data is processed in FPGA and the resultant features are then transferred via wireless protocols. We selected two different FPGA devices from Xilinx Zynq 7000 family. The specification of each FPGA is given in Table 1. The two horizontal lines in the plot represent the maximum fps supported by different version of Bluetooth protocol. The design space provides the complete picture for FPGA designs which are compatible with wireless protocols. For example keeping the current design (clock speed, resources constraints), the matrix size of 10x10 is supported by Bluetooth 2.0 but unsupported by BLE. The curves for both FPGA's are similar to Matlab but less steep because FPGA processing of frames improved the fps.

FPGA 2 has more capacity and resources that's why the fps is higher as compared to FPGA 1. However, for each FPGA selected the ratio of increase is low with higher matrix size. This happens because the amount of data for higher matrix processed is increased, so the FPGA has relatively less resources to improve the timing of design. When the matrix size is reduced (meaning less resources requirement), the available resources become abundant and it produces high performance design. However, as stated earlier we have kept the design frequency and constraints constant. For small matrix size we can reduce the frequency since this would result in reduced power consumption for applications with small matrix size.

It is evident from Figure 4 that each FPGA provides an opportunity to use wireless transmission due to fast and parallel processing of matrix frame. FPGA based design improves the runtime requirement of processing of samples, and this compresses the data into features of small sizes.

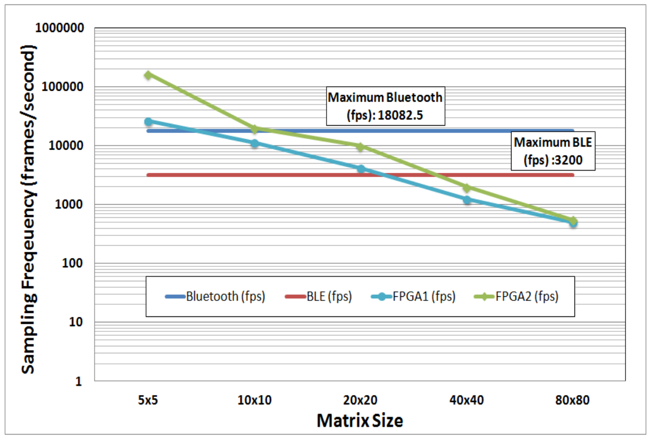


Figure 4 Design space exploration using different FPGA devices

Figure 5 presents the power analysis of our design. The power analysis is done using Xilinx power estimator (XPE) 2016.1 as explained in [14]. This tool gives the estimates of total on-chip power depending on the number of resources, clock speed and interfaces used in the design. The results are based on very high level of abstraction. We expect to receive similar results after synthesis and placement within FPGA. This tool provides a reasonable architecture level estimate for the design. It can be seen in Figure 5 that FPGA 2 which has more resources, has high power consumption than FPGA 1.

The power consumption of large matrix size is high. This is also consistent with the fact the high resources (flip flops, LUT, BRAM) require more power. Our power calculations also suggest that a typical battery with a capacity of 500 mAh can last up to 3-5 hours without replacement.

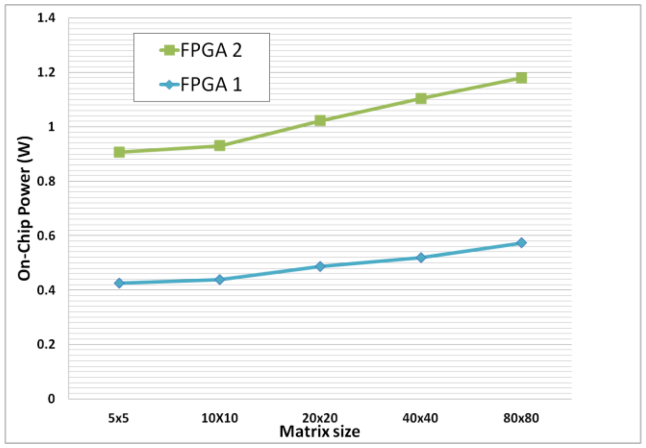


Figure 5 Variation of on-chip power with different matrix size along with the comparison between different FPGA devices

Conclusion

In this work, we presented a way to speed up pressure sensing algorithms using FPGA, providing an option of real-time processing of sensor data with the possibility of using wireless medium to transfer data to mobile. These tasks are compute intensive and require longer runtime when executed in Matlab. We have proposed an alternative and easy approach which can be used in many feature extraction algorithms which needs acceleration. We used high level synthesis, which is a new and efficient approach especially for software programmers with little knowledge in FPGA, to accelerate their image processing and feature extraction algorithm in FPGA using C/C++. We have presented here a comparison of speed achieved while implementing algorithms related to pressure sensing. Simulation results based on pressure sensing matrix demonstrate both reduced processing time per frame and reduced output data-rate, which enhance the design-room of sensing matrix with higher channel number and/or higher sampling rate, supporting meanwhile still real-time data processing and wireless transmission.

Acknowledgements

This work was partially supported by the collaborative project SimpleSkin under contract with the European Commission (#323849) in the FP7 FET Open framework. The support is gratefully acknowledged. The authors would also like thank all students and colleagues who volunteered as test subjects in our experiments.

References

1. Alan Bränzel, Christian Holz, Daniel Hoffmann, Dominik Schmidt, Marius Knaust, Patrick Lühne, René Meusel, Stephan Richter, and Patrick Baudisch. 2013. GravitySpace: tracking users and their poses in a smart room using a pressure-sensing floor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13). ACM, New York, NY, USA, 725-734. DOI=http://dx.doi.org/10.1145/2470654.2470757
2. David G. Lowe. 1999. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2* (ICCV '99), Vol. 2. IEEE Computer Society, Washington, DC, USA, 1150-.
3. E. G. Pereira, L. C. Oliveira, M. R. A. Morais, A. M. N. Lima and H. Neff, "Implementation of a FPGA-based data acquisition and processing system for image sensors employed in SPR biosensing," 2014 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings, Montevideo, 2014, pp. 884-889

4. Endo, T. Isshiki, D. Li and H. Kunieda, "A design method for real-time image denoising circuit using High-Level Synthesis," 2016 7th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES), Bangkok, 2016, pp. 30-35.
5. G. van der Wal et al., "FPGA acceleration for feature based processing applications," 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Boston, MA, 2015, pp. 42-47.
6. Hu, M.-K. Visual pattern recognition by moment invariants, computer methods in image analysis. *IRE Transactions on Information Theory 8* (1962).
7. Linares-Barranco, F. Gómez-Rodríguez, V. Villanueva, L. Longinotti and T. Delbrück, "A USB3.0 FPGA event-based filtering and tracking framework for dynamic vision sensors," 2015 IEEE International Symposium on Circuits and Systems (ISCAS), Lisbon, 2015, pp. 2417-2420
8. M. Jacobsen, Z. Cai and N. Vasconcelos, "FPGA implementation of HOG based pedestrian detector," 2015 International SoC Design Conference (ISOCC), Gyungju, 2015, pp. 191-192.
9. M. Schmid, N. Apelt, F. Hannig and J. Teich, "An image processing library for C-based high-level synthesis," 2014 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, 2014, p
10. Nagendra Bhargava Bharatula, Paul Lukowicz, and Gerhard Tröster. 2008. Functionality-power-packaging considerations in context aware wearable systems. *Personal Ubiquitous Comput.* 12, 2 (January 2008), 123-141. DOI=http://dx.doi.org/10.1007/s00779-006-0106-3
11. R. Rasul et al., "FPGA Accelerated Computing Platform for MATLAB and C/C++," Frontiers of Information Technology (FIT), 2013 11th International Conference on, Islamabad, 2013, pp. 166-171.
12. Sundholm, M., Cheng, J., Zhou, B., Sethi, A. Smar t-Mat: Recognizing and Counting Gym Exercises with Low-cost Resistive Pressure Sensing Matrix. In UbiComp '14 Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing, 373-382, http://dx.doi.org/10.1145/2632048.2636088
13. Xilinx high level synthesis guide. http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf
14. Xilinx power estimation (XPE). http://www.xilinx.com/products/technology/power/xpe.html
15. Zhou, B., Cheng, J., Sundholm, M., and Lukowicz, P. From Smart Clothing to Smart Table Cloth: Design and Implementation of a Large Scale, Textile Pressure Matrix Sensor. In *Springer Lecture Notes on Computer Science (LNCS) series*, ARCS 2014 – International Conference on Architecture of Computing Systems (2014)