FPGA Implementation of a People Counter for an Ultra-Low-Power Wireless Camera Network Node

Leonardo Gasparini, Massimo Gottardi and Nicola Massari
Smart Optical Sensors and Interfaces
Fondazione Bruno Kessler
Trento, Italy
Email: gasparini@fbk.eu

Dario Petri
Dept. of Information Engineering and Computer Science
University of Trento, Italy
Email: petri@disi.unitn.it

Roberto Manduchi
Dept. of Computer Engineering
University of California
Santa Cruz, CA 95064, USA
Email: manduchi@soe.ucsc.edu

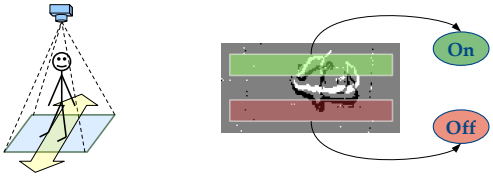
***Abstract*—Wireless Camera Network (WCN) nodes differ from traditional Wireless Sensor Network (WSN) nodes because of the huge amount of data generated by the sensing element. In order to be able to operate on batteries for a long period, a WCN node needs to extract the information contained into an image and synthesize it into a short message that can be wirelessly transmitted with a limited amount of power. Unfortunately, this approach typically brings the power consumed by the processing unit at the same level as the transceiver, known to be the major source of power consumption in WSN. Thus, there is the need to design efficient algorithms that can be implemented on low-power devices. In this paper we propose the implementation of a Dijkstra-based people counting algorithm for ultra-low-power FPGAs. The developed code has been integrated on the prototype of a WCN node that consumes as little as 5mW.**

I. Introduction

Wireless Sensor Network (WSN) nodes are able to work for several months when powered with a couple of standard AA batteries [1]. The employed sensors, such as temperature and pressure sensors, typically generate a limited amount of data at quite low rates. Thus, there is no need for high processing capabilities within the node and an ultra-low-power device is able to control the entire process (involving acquisition, processing and wireless communication).

Things get different if we equip the node with a camera. In [2], for example, the authors present a highly flexible, high performance WCN node that can be used for several applications. Nevertheless, in order to provide the system with enough computational power, the node consumes several hundreds of milliwatts, i.e. two orders of magnitude higher than a standard WSN.

In [3] an ultra-low-power WCN node has been proposed employing a smart imager [4] and a flash-based FPGA. The sensor generates binary images where pixels with logic value "1" (active) represent the high contrast points of the scene. In practice we get an image that represents the edges of the objects present in the scene. Moreover, the sensor may also perform on-chip frame differencing to perform motion detection. The FPGA is an Actel IGLOO [5] which controls the sensor, buffers and processes the images. Two clock domains are implemented in order to save power. One runs at a low frequency ($\simeq$ 15 KHz) and clocks the units responsible for controlling the system; the other one, mainly used to process the images, runs at 10 MHz and is activated only when needed.



(a) (b)

Fig. 1. Camera node setup (a) and an acquired image representing a person that moves upwards (b). Image (b) shows the two implemented VILs. Each VIL can be *on* or *off* according to the number of non-zero pixels that it contains.

In this paper we propose the FPGA implementation of a people counter [6] based on the Dijkstra algorithm [7]. Such algorithm has been designed on the basis of the images provided by the aforementioned sensor. By exploiting the low-power characteristics of the system, the entire system (FPGA and sensor) consumes less than 5 mW when running the algorithm at 30 fps.

This paper is organized as follows. In Sec. II we summarize the algorithm for counting people, while in Sec. III we describe the FPGA implementation of the algorithm. Conclusions are drawn in Sec IV.

II. People Counting Algorithm

The algorithm aims at counting people walking through a door or a corridor. The camera node is placed on top of the monitored area facing downwards, as shown in Fig. 1(a). Three types (*modes* M_k) of events are considered:

- no people are crossing the area (M_{none});
- one single person enters the scene, i.e. it moves from the top to the bottom of image (M_{in});
- one single person exits, i.e. it walks upwards (M_{out}).

The algorithm is based on the Virtual Inductive Loop (VIL) mechanism defined by Viarani *el al.* in [8]. A VIL consists on a portion of the image that assumes a binary state (on or off) according to the absence/presence of foreground objects within it. In our case, we have two VILs defined by two non-overlapping rectangular windows which are as wide as

978-1-4244-9137-7/11/$26.00 ©2011 IEEE

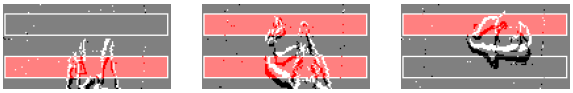


Fig. 2. An acquired sequence of images representing a person walking upwards and the implemented VILs. The system state is represented by a two-bit variable, describing the status of the VILs.

the monitored passage and are aligned along the direction of movement; see Fig. 1(b). We detect the foreground by means of the motion detection mechanism implemented on the sensor, and the state of each VIL is determined by the number of active pixels with respect to a threshold. By concatenating the state of the VILs, we get a binary string that represents the state S of the system for that frame, as shown in Fig. 2.

In this context, we define:

- *segments* as contiguous frames characterized by the same state; a segment σ is therefore entirely described by an index j, a state S and a duration (in terms of frames) t: $\sigma^j = (S^j, t^j)$;
- *intervals* as groups of contiguous segments which represent/are classified as the same event, i.e. a person that enters in the area, one that exits, or no one around; an interval I is entirely described by an index i, a sequence of segments Σ, and by the mode M_k that it belongs to: $I^i = (\Sigma^i, M_k^i)$.

We can classify intervals as belonging to one of the three available modes by analyzing the state transitions. In fact, according to our experiments, a person that walks in one direction generates a sequence of states which is similar to the one generated by other people moving in the same direction. At the same time, such a sequence has very little in common with the ones originated from people walking in opposite direction. Moreover, we discovered that an event of mode M_k constituted by a sequence $\mathbf{S} = (S^1, S^2, \ldots, S^N)$ is a Markov process of the second order, i.e. its likelihood $P(\mathbf{S}|M_k)$ is such that:

$$P(\mathbf{S}|M_k) = P(S^2|S^1; M_k) \cdot \prod_{n=3}^{N} P(S^n|S^{n-2}, S^{n-1}; M_k). \tag{1}$$

In practice this means that the n^{th} state in the sequence depends just on the two previous states.

The people counter monitors the segments generated by the flow of people and finds the combination of contiguous, non-overlapping intervals that maximize the product of the likelihoods. This is carried out by extracting all possible intervals $I^i = (\Sigma^i, M_k^i)$ and determining their likelihood $P^i = P(\mathbf{S}^i|M_k^i)$. Then, the cost function $P_{tot} = \prod_i P^i$ is maximized under the following constraints: (a) all segments must be covered by one and only one interval, and (b) each interval cannot be longer than T_{max}, currently set to 2 seconds.

In order to calculate the P^i's, we need to know the first- and second-order transition probability maps $P(S^2|S^1; M_k)$ and $P(S^n|S^{n-2}, S^{n-1}, M_k)$ for each mode M_k. We estimate them in a training phase.

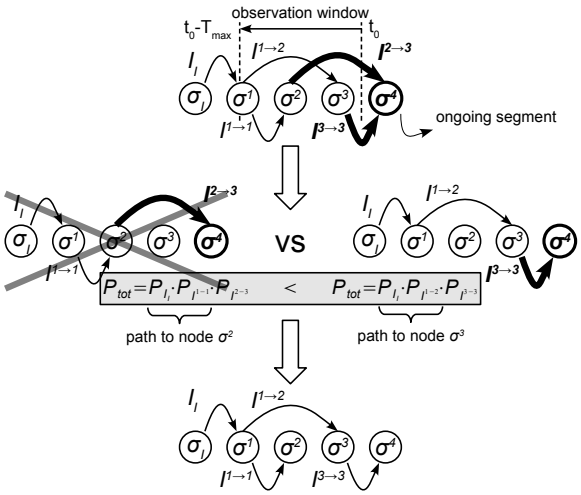


Fig. 3. The image shows the process of insertion of a new node (σ_4) into the graph. The first step requires to find the new edges (bold arrows) and calculate their cost. Each edge is associated to an interval. For example, the edge connecting σ^2 to σ^4 represents the $I^{2\to3}$ interval, i.e. the interval that includes segments σ^2 to σ^3. No edge can be drawn connecting σ^1 to σ^4 because the $I^{1\to3}$ interval does not fit int the T_{max}-wide observation window (sketched with the dashed vertical lines at the top). Then, all the paths to the current node have to be compared and only the one with maximum cost is kept.

The problem can be represented through an oriented graph in which we add a node for each segment and we draw an edge for every interval that we take into account (we will then use the terms node and segment and the terms edge and interval interchangeably). More specifically, if the interval includes the segments $\sigma^i, \sigma^{i+1}, \ldots, \sigma^{j-1}$ then the corresponding edge $I^{i\to(j-1)}$ will connect the node σ^i to the node σ^j. No cost is associated to the nodes, while the cost of each edge is given by the likelihood P^i of the interval $I^i = (\Sigma^i, M^i)$ that it represents. The graph origins on an initial node σ_I signifying that the people counter has been activated, and ends on a final node σ_F created when a single segment longer than T_{max} occurs. The combination of intervals which is more likely to have happened corresponds to the highest cost path from σ_I to σ_F. This is found by using a modified version of the Dijkstra algorithm, as following described with reference to Fig. 3.

At every state transition, a new segment σ^{n+1} is generated, thus we insert a new node into the graph. Then, we create edges to it, i.e., according to our representation, we find all the possible intervals that end with the segment σ^n. Once we have found the edges, we need to compare all the paths that bring to node σ^{n+1} and keep only the one that maximizes the cost. Since we keep track of the highest cost path to each node σ^{n-p}, this task simply involves multiplying the cost of each newly generated edge $I^{(n-p)\to n}$ by the total cost of the path up to σ^{n-p} (which is the source node of that edge). At the end, there will be only one edge directed inward for each node, and all we need to remember about this edge is: its source node, its cost and the mode associated to the interval that it represents. When the system state does not change for a period longer than T_{max}, a σ_F node is added to the graph.

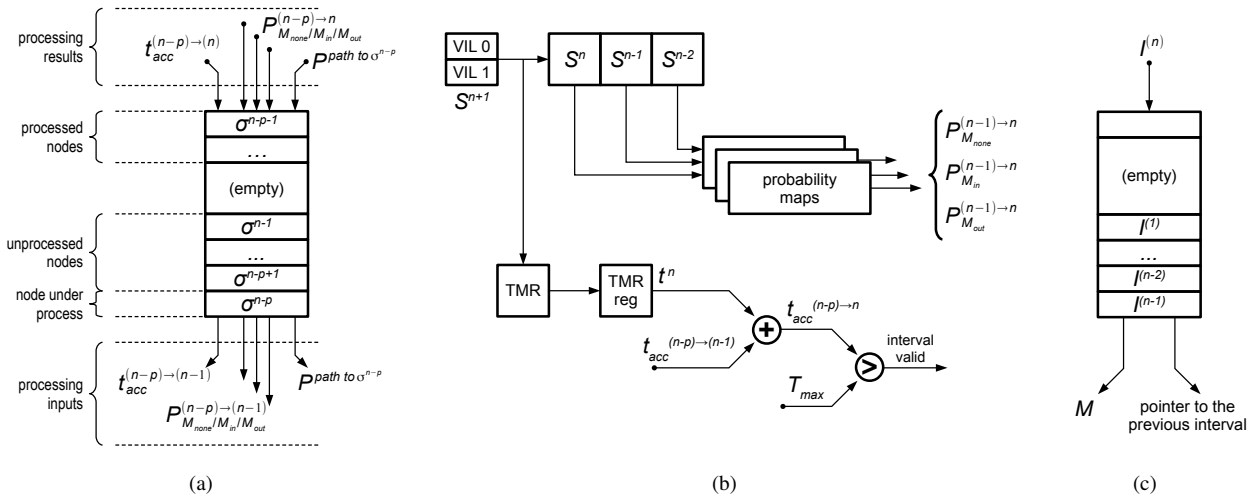


Fig. 4. FIFO memories and node insertion. In (a) the *node memory* is present; the figure shows its state in the middle of the processing phase, while computing the path to node σ^n through a generic node σ^{n-p}. (b) depicts the architecture of the *VILs*, the *three cell memory* and the *LUTs*, the *timer* with the *adder* and the T_{max}*-comparator*. The edge memory is drawn in (c).

III. FPGA IMPLEMENTATION

In hardware, the algorithm described in the previous section is carried out with the following architecture:

- an *image processing unit*, that extracts the current system state from the acquired image;
- *Look Up Tables (LUTs)*, containing the probability maps;
- a *three cell memory*, that keeps track of the states of the last segments for LUT addressing;
- a *timer*, that measures the duration of segments, and an associated *adder* to calculate the total duration of intervals;
- a *node First In, First Out (FIFO) memory*, that keeps track of the most recent nodes in the graph;
- a set of multipliers that calculate the likelihood of M_k intervals (*edge multipliers*) and the total cost of paths (*path multiplier*);
- a set of *comparators*, used (1) to determine if a group of contiguous segments is longer than T_{max}, (2) to find the mode that is more likely for a given a sequence of contiguous segments, and (3) to extract the path with the maximum cost;
- an *edge FIFO memory*, that stores the results;
- *registers* and *multiplexers*;
- a *control unit*, that manages the whole process.

Fig. 4 and Fig. 5 represent the block diagram for the architecture. 8 bit fixed point representation is used for the probabilities in the LUTs. The multipliers generate 16 bit fixed point numbers. This is the format used in the FIFO memories, also. Since numbers are always ≤ 1.0, all the bits are dedicated to the fractional part.

Every time a state transition occurs, we insert the node σ^{n+1} in the graph and find the maximum cost path to it. Node insertion is carried out by saving the VIL state in the three cell memory and the timer value into a dedicated register before resetting them for the next acquisition. Then, in order to build the first edge, we extract the last node contained in the node's memory (let's call it node σ^{n-q}). The node data is given by:

- the cumulative period of time $t_{acc}^{(n-q)\rightarrow(n-1)}$ that includes the node itself up to node σ^{n-1};
- the partial likelihoods $P_{M_k}^{(n-q)\rightarrow(n-1)}$ of the M_k intervals that origin from the node and contain nodes up to the σ^{n-1} one;
- the cost of the maximum cost path from σ_I to the node σ^{n-q} itself.

The partial likelihoods are stored in the node memory in order to avoid to perform the same calculations at every node insertion.

The adder sums the value provided by the timer to the cumulative period and the result $t_{acc}^{(n-q)\rightarrow n}$ is then compared with T_{max}. If it is greater, then the σ^{n-q} node is discarded from the node memory and the following node (the σ^{n-q+1}) is read. Otherwise, the sequence of segments $\sigma^{n-q}, \ldots, \sigma^n$ represents a valid interval and we have to compute its likelihood for each M_k. Therefore, we fetch the transition probabilities $P_{M_k}^{(n-1)\rightarrow n}$ stored in the LUTs using the content of the three cell memory for addressing. This is shown in Fig. 4(b).

Then, we multiply the so obtained transition probabilities by the partial likelihoods to obtain the costs $P_{M_k}^{(n-q)\rightarrow n}$ of the intervals that include the segments from σ^{n-q} to σ^n. In the event that one further segment is generated, the data about the σ^{n-q} segment are written back to the node memory, with the updated values for t_{acc} and the partial likelihoods. In the next step, the three intervals are compared and only the most likely one is kept into account. Its likelihood $P_{M_{max}}^I = max_k(P_{M_k}^{(n-q)\rightarrow n})$ is multiplied with the cost of the path to σ^{n-q} by the interval multiplier to achieve the cost of the path to σ^{n+1}. The result is temporarily stored in the last comparator's output register, along with the corresponding mode and the index q that defines the newly generated interval $I^{(n-q)\rightarrow n}$. Fig. 5 describes the hardware architecture that carries out these operations.

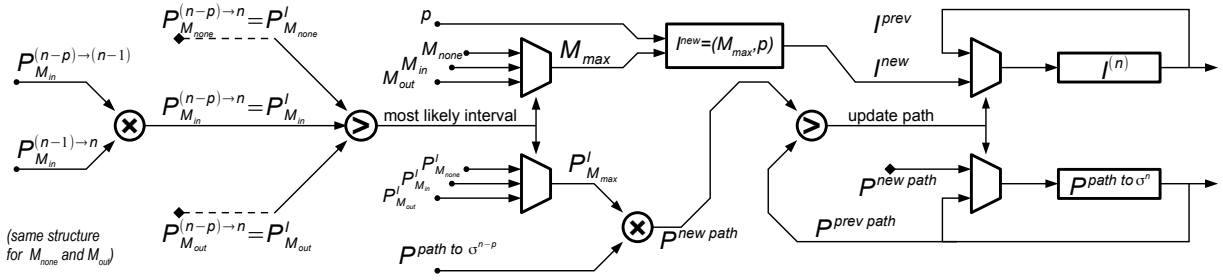


Fig. 5. Architecture of the processing elements that calculate the maximum cost path to a new node σ^{n+1}. The entire process requires: (1) to calculate the most likely mode M_k for the interval constituted by the segments $\sigma^{n-p}, \ldots, \sigma^n$; (2) to multiply the interval likelihood by the cost of the path from σ_I to σ^{n-p} to achieve the cost $P^{new\ path}$ of the entire path to σ^n through σ^{n-p}; (3) compare the so obtained cost with the temporary maximum cost $P^{prev\ path}$ resulting from the edges computed so far and, in case, overwrite it.

test events	156	163
errors	0.0%	0.1%
missed detections	3.2%	3.1%
false alarms	0.0%	13.7%

TABLE I
CLASSIFICATION PERFORMANCE.

Then, the process starts over with a new node, the σ^{n-p+1}, thus generating a different path to node σ^n. Its cost $P^{new\ path}$ is compared with the previous one $P^{prev\ path}$ and of the two, only the one with higher cost is kept. Iteratively, all the nodes σ^{n-p}, with $0 < p \leq q$, that are contained within the node memory are processed. One further interval is created, consisting in the only σ^n segment. At the end of the process, the path comparator provides the data about the maximum cost path. Its cost is saved into the node memory of Fig. 4(a) along with the partial likelihoods and the timer's value as a new element, while the mode and the source of the last interval in the path (i.e. the data about the last edge) are saved into the edge memory present in Fig. 4(c). Then the system waits for another segment.

When the timer exceeds T_{max} the process is interrupted and the control unit can extract the intervals that are more likely to have happened by reading the data from the edge memory. Each element will contain the mode associated to the edge and the pointer to the previous valid element in the memory. This is like going back along the maximum cost path jumping from the first segment of an interval to the one of the previous interval.

This architecture, implemented on the Actel IGLOO FPGA, has been tested with a clock running at 10 MHz. At such an operating frequency, the system can safely acquire images at 10 fps. We tested the algorithm on Matlab, acquiring two long movies each including more than 150 events. We exchanged the movies in the role of training set and test set. The achieved results are quite satisfactory, as shown in Tab. I. In fact, very few errors occur, and few missed events are present. Only in one situation the algorithm generates a great amount of false alarms. This is due to the presence of long shadows in the test movie which are not present in the one used for training. Therefore, a person and its shadow are classified as two people walking in a row in the same direction.

IV. CONCLUSION

The lack of ultra-low-power implementations of image processing algorithms slows down the development of camera-based smart WSN nodes. In this paper we demonstrated that it is possible to implement a quite complex algorithm on an ultra-low-power FPGA. The developed application is a people counter based on the Dijkstra algorithm and runs smoothly on an FPGA-based node clocked at 10 MHz and acquiring images at 30 fps, with good classification performance. Despite the intrinsic complexity of the application, the node consumes as little as 5 mW. The key factors consist on the employment of a smart vision sensor that performs pre-processing directly on chip, and the implementation of an efficient architecture within the FPGA that exploits its intrinsic multi-tasking nature.

REFERENCES

[1] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proceedings of the 4th international symposium on Information processing in sensor networks*. IEEE Press, 2005, pp. 48–es.

[2] P. Chen, P. Ahammad, C. Boyer, S. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan *et al.*, "CITRIC: A low-bandwidth wireless camera network platform," in *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*. IEEE, 2008, pp. 1–10.

[3] L. Gasparini, R. Manduchi, M. Gottardi, and D. Petri, "Performance analysis of a wireless camera network node," in *Instrumentation and Measurement Technology Conference (I2MTC), 2010 IEEE*. IEEE, 2010, pp. 1331–1336.

[4] M. Gottardi, N. Massari, and S. Jawed, "A 100μw 128 $\times$ 64 pixels contrast-based asynchronous binary vision sensor for sensor networks applications," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 5, pp. 1582 –1592, may 2009.

[5] "Igloo low-power flash fpgas datasheet," http://www.actel.com/techdocs/-ds/low-power-fpgas.aspx#igloo, Actel Corporation, Mountain View, CA 94043 USA, 2009.

[6] L. Gasparini, R. Manduchi, and M. Gottardi, "An ultra-low-power contrast-based integrated camera node and its application as a people counter," in *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, 29 aug - 1 sep 2010, pp. 547 –554.

[7] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[8] E. Viarani, "Extraction of traffic information from images at deis," in *Image Analysis and Processing, 1999. Proceedings. International Conference on*, 1999, pp. 1073 –1076.