© Journal of the Technical University - Sofia
Plovdiv branch, Bulgaria
"Fundamental Sciences and Applications"
Vol. 26, 2020

Speed focused FPGA based Canny computations: pipelining

Dimitre Kromichev

Department of Marketing and International Economic Relations, University of Plovdiv, 24 Tzar Asen Street, Plovdiv 4000, Bulgaria

dkromichev@yahoo.com

Abstract. Presented is a complete study of the functional model of pipelining in FPGA based Canny. Distinguished are high-, mid- and low level of pipelining. The speed capabilities of pipelining in Gaussian filtering, Sobel, gradient magnitude and direction, non-maximum suppression are explored at each level. Provided are the formulae for calculating the exact number of clock cycles required to execute these Canny modules with respect to the two input variables: image size and Gaussian filter size. Mathematically proved is the upper limit of FPGA based Canny speed in terms of pipelining.

1. Introduction

In FPGA based Canny, the two basic parameters of speed are: maximum clock frequency and minimum number of clock cycles required to calculate an accurate result at this clock frequency. The technique of choice to address the latter is pipelining. Its specifics depend on:

- Two input variables: image size and Gaussian filter size
- Execution of Sobel and non-maximum suppression computations require the simultaneous availability of all pixels pertaining to the square neighbourhood.

On that basis, pipelining in FPGA based Canny demands a multilevel approach.

In the literature, described are FPGA implementations of Canny which use pipelining to increase speed [4][9][10]. Pipelining is of particular importance for distributed Canny [1][2][3][8]. Although widely considered a tool of choice for boosting speed [5][6][7], none of the works available so far provide an in-depth analysis of the pipelining specifics in FPGA based Canny. Nor is there a reliable exploration of the impact of pipelining complexity on Canny's execution with respect to Gaussian filter size and organization of computations in Gaussian smoothing module.

The objective of this paper is to thoroughly study FPGA based Canny pipelining by distinguishing high-, mid- and low level of pipelining functionality in Gaussian filtering, Sobel, gradient magnitude and direction, non-maximum suppression. The task is to analyze the efficiency of each level in the Canny modules with respect to proving the upper limit of speed in FPGA based Canny, as well as provide formulae for calculating the exact number of clock cycles required to execute these Canny modules by considering the impact of the two input variables: image size and Gaussian filter size. Relevant to the conclusions arrived at are only gray-scale images. The targeted hardware is Intel (Altera) FPGAs.

2. High level pipelining

It requires that for a definite number of clock cycles Gaussian filtering, Sobel, gradient magnitude and direction, and non-maximum suppression modules execute simultaneously. For an input image

 Online ISSN 2603-459X

$u(m,n)$ and Gaussian filter of size zxz, the number of clock cycles required by each module to process the entire image is:

- Gaussian filtering

$$nTclk(Gauss) = \{x \in N \mid 1 \le x \le [(m-(z-1))*(n-z)]*S + (m-(z-1))*T\} \quad (1)$$

where

$nTclk(Gauss)$ is the clock cycles required to filter the image,
S is the clock cycles between two consecutive filtered pixels,
T is the clock cycles required to filter pixel #1.

- Sobel filtering

$$nTclk(Sobel) = \{x \in N \mid 1 \le x \le [(m-(z+1))*(n-(z+1))]*Y + E\} \quad (2)$$

where

$nTclk(Sobel)$ is the clock cycles required to filter an image with Sobel,
Y is the clock cycles required to calculate an x-/y-gradient,
E is the clock cycles required to execute division #1 in Sobel.

- Gradient magnitude and direction

$$nTclk(Magn) = nTclk(Direct) = \{x \in N \mid 1 \le x \le [(m-(z+1))*(n-(z+1))]*Q\} \quad (3)$$

where

$nTclk(Magn)$ is the clock cycles required for all gradient magnitudes,
$nTclk(Direct)$ is the clock cycles required for all gradient directions,
Q is the clock cycles required for a gradient magnitude/direction.

- Non-maximum suppression

$$nTclk(NMS) = \{x \in N \mid 1 \le x \le [(m-(z+1))*(n-(z+1))]*H + F\} \quad (4)$$

where

$nTclk(NMS)$ is the clock cycles required for all non-maximum values,
H is the clock cycles required by a single non-maximum value,
F is the clock cycles required by non-maximum value #1.

Therefore, on the basis of (1), (2), (3) and (4) high-level pipelining (Figure 1) is expressed as:

$$nTclk(Gauss) \cap nTclk(Sobel) \cap nTclk(Magn) \cap nTclk(Direct) \cap nTclk(NMS) = \{nTclk(PipeHigh)\} \quad (5)$$

where

$nTclk(PipeHigh)$ is the clock cycles during which high level pipelining is realized.

Therefore, for particular image size and Gaussian filter size $nTclk(PipeHigh)$ is a constant.

As Figure 1 shows, the start of high-level pipelining is characterized by a specific delay in each module after Gaussian:

- Sobel

$$DsS = nTclk(Gauss(p\#1)) + \sum_{i=1}^{n} nTclk(RAMp_i) + nTclk(RAMw-to-r(p\#1)) \quad (6)$$

where

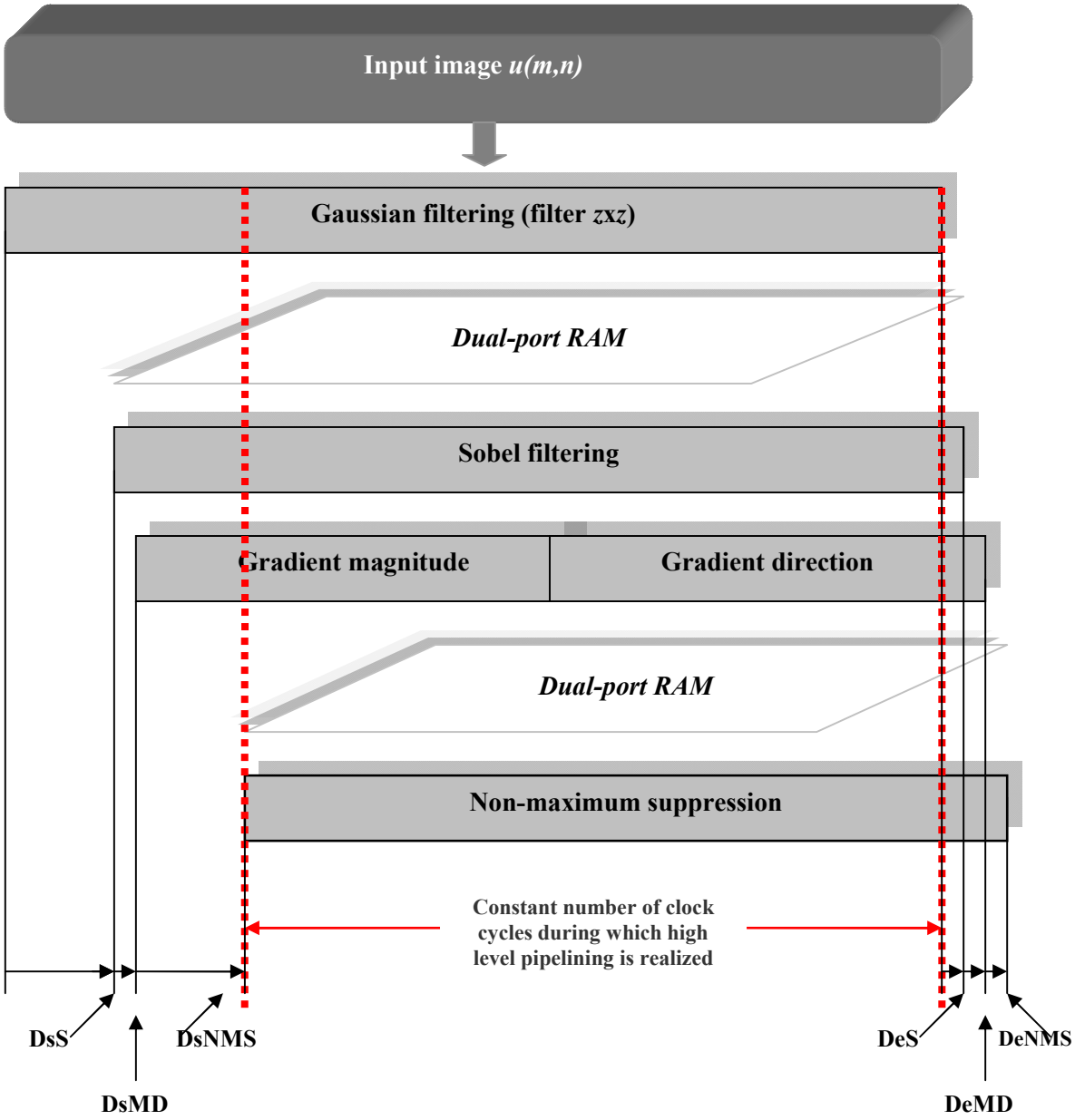

Figure 1. Graphical representation of high level pipelining

DsS is the start delay of high-level pipelining in Sobel,

$nTclk(Gauss(p\#1))$ is the clock cycles required to filter pixel #1with Gaussian,

$nTclk(RAMp_i)$ is the clock cycles required to store a single Gaussian filtered pixel in Dual-port RAM,

$\sum_{i=1}^{n} nTclk(RAMp_i)$ is the clock cycles required to store the necessary minimum of pixels in Dual-port RAM

$nTclk(RAMw-to-r(p\#1))$ is the clock cycles required by Dual-port RAM for operations write-read for Gaussian pixel #1.

In (6), $nTclk(RAMp_i)$ depends entirely on the technology of computations in Gaussian filtering. In (6), $\sum_{i=1}^{n}$ denotes the exact number of Gaussian filtered pixels required to start Sobel and guarantee uninterrupted execution when one image row transitions into the next row.

- Gradient magnitude and direction:

$$DsMD = nTclk(Sobel) \tag{7}$$

where

$DsMD$ is the start delay of magnitude and direction,
$nTclk(Sobel)$ is the clock cycles required by Sobel.

- Non-maximum suppression:

$$DsNMS = nTclk(Magn/Direct(p\#1)) + \sum_{i=1}^{n} nTclk(RAMp_i) + nTclk(RAMw-to-r(p\#1)) \tag{8}$$

where

$DsNMS$ is the start delay of non-maximum suppression,
$nTclk(Magn/Direct(p\#1))$ is the clock cycles required to calculate magnitude and direction #1,
$nTclk(RAMp_i)$ is the clock cycles required to store a single magnitude and direction in Dual-port RAM,
$\sum_{i=1}^{n} nTclk(RAMp_i)$ is the clock cycles required to store the necessary minimum of pixels in Dual-port RAM,
$nTclk(RAMw-to-r(p\#1))$ is the number of clock cycles required by Dual-port RAM for operations write-read for magnitude and direction #1 .

In (8), $nTclk(RAMp_i)$ depends entirely on the technology of computations in gradient magnitude and direction. In (8), $\sum_{i=1}^{n}$ denotes the exact number of magnitude values and direction values required to start non-maximum suppression and guarantee uninterrupted execution when one image row transitions into the next row.

According to (6), (7) and (8), DsS, $DsMD$ and $DsNMS$ are constants.

As Figure 1 shows, the end of high-level pipelining in each module after Gaussian is characterized by a specific delay which is equal to:

- Sobel

$$DeS = (zS + zS + 1) * (z + 1) \tag{9}$$

where

DeS is end delay of high-level pipelining in Sobel,
zS is the side of Sobel filters,
z is the side of Gaussian filter.

- Gradient magnitude and direction:

$$DsMD = Y \tag{10}$$

where

$DeMD$ is end delay in gradient magnitude and direction.

- Non-maximum suppression:

$$DeNMS = (zNMS + 1) * Q \tag{11}$$

where

$DeNMS$ is end delay in non-maximum suppression,

$zNMS$ is the side of non-maximum suppression square neighbourhood.

According to (9), (10) and (11), DeS, $DeMD$ and $DeNMS$ are constants.

Thus, on the basis of high level pipelining, the formula for calculating the number of clock cycles required to execute Gaussian filtering, Sobel, gradient magnitude and direction, and non-maximum suppression is:

$$nTclk\sum = nTclk(PipeHigh) + DsS + DsMD + DsNMS + DeMD + DsNMS = nTclk(Gauss) + DeS + DeMD + DeNMS \tag{12}$$

where

$nTclk\sum$ is the total number of clock cycles required for Gaussian filtering, Sobel, gradient magnitude and direction, non-maximum suppression.

High-level pipelining is realized for every image $u(m,n)$ in which

$$\begin{aligned} m &> z^2 + zS^2 \\ n &> z^2 + zS^2 \end{aligned} \tag{13}$$

where

z is the side of Gaussian filter,

zS is the side of Sobel filters.

3. Mid-level pipelining

This type is defined on the basis of computations in two consecutive Canny modules. It is represented by two sets of arithmetic operations which are executed simultaneously

$$\begin{aligned} A &= \{a(n+am) \mid n \in N, am \in Z, n = 1, 0 \le am \le tc\} \\ D &= \{d(n+dm) \mid n \in N, dm \in Z, n = 1, 0 \le dm \le 4 * w - 1\}. \end{aligned} \tag{14}$$

where

tc is a constant determined by the organization of computations in Gaussian filtering,

$w = \frac{z-1}{2}$, z is the side of Gaussian filter and $z = 3$.

Within the sets (14):

- For $am \ge 1$ and $dm \ge 1$, the inputs of operations *a(n+am)* and *d(n+dm)* represent the outputs of operations $a(n + (am - 1))$ and $d(n + (dm - 1))$
- For $am = 0$ and $dm = tc$, the input of operation $d(n + am)$ is the output of operations $a(n + dm)$.

Mid-level pipelining (Figure 2) is possible only if

$$\sum_{i=1}^{tc+1} nTclk(a(n_i)) \ge \sum_{i=1}^{4*w} nTclk(d(n_i)) \tag{15}$$

where

$nTclk(a(n_i))$ is the clock cycles required to execute a single operation from set A,

$nTclk(d(n_i))$ is the clock cycles required to execute a single operation from set D.

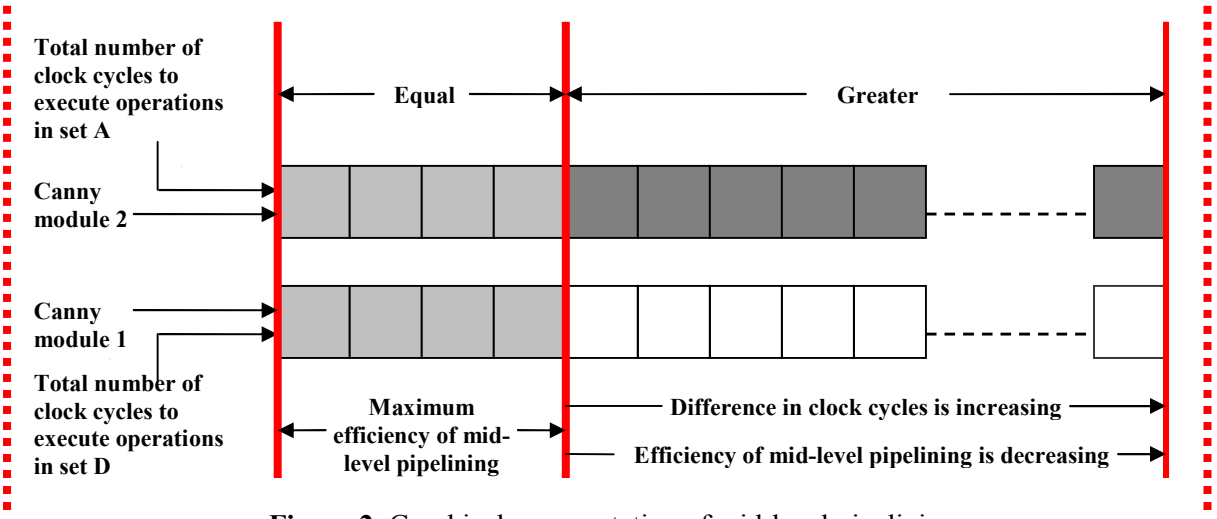

Figure 2. Graphical representation of mid-level pipelining

As Figure 2 shows, the notation "≥" in inequality (15) represents the specifics of mid-level pipelining:

- If

$$\sum_{i=1}^{tc+1} nTclk(a(n_i)) = \sum_{i=1}^{4*w} nTclk(d(n_i)) \tag{16}$$

there are no empty clock cycles between the end of arithmetic operation $d(n4*w)$ and the start of arithmetic operation $d(n1)$. Therefore, in his case the efficiency of pipelined computations is at its maximum.

- If $\sum_{i=1}^{tc+1} nTclk(a(n_i)) > \sum_{i=1}^{4*w} nTclk(d(n_i))$, then $P = \sum_{i=1}^{tc+1} nTclk(a(n_i)) - \sum_{i=1}^{4*w} nTclk(d(n_i))$ (17)

where

P is the pause measured in empty clock cycles which occurs between the end of arithmetic operation $d(n4*w)$ and the start of arithmetic operation $d(n1)$.

Therefore, in this case the efficiency of pipelined computations is decreased proportionally to P. There is a special case to be considered:

$$D = \{d(n+dm) \mid n \in N, dm \in Z, n = 1, dm = 1\}. \tag{18}$$

In that case, the following inequality must be satisfied

$$\sum_{i=1}^{tc+1} nTclk(a(n_i)) \geq nTclk(a(n)) \tag{19}$$

where

$nTclk(a(n))$ is the clock cycles required to execute the operation from set D.

4. Low level pipelining

This type is defined on the basis of computations within a Canny module. It focuses on the simultaneous execution of consecutive arithmetic operations within a single clock cycle. There are four variants (Figure 3):

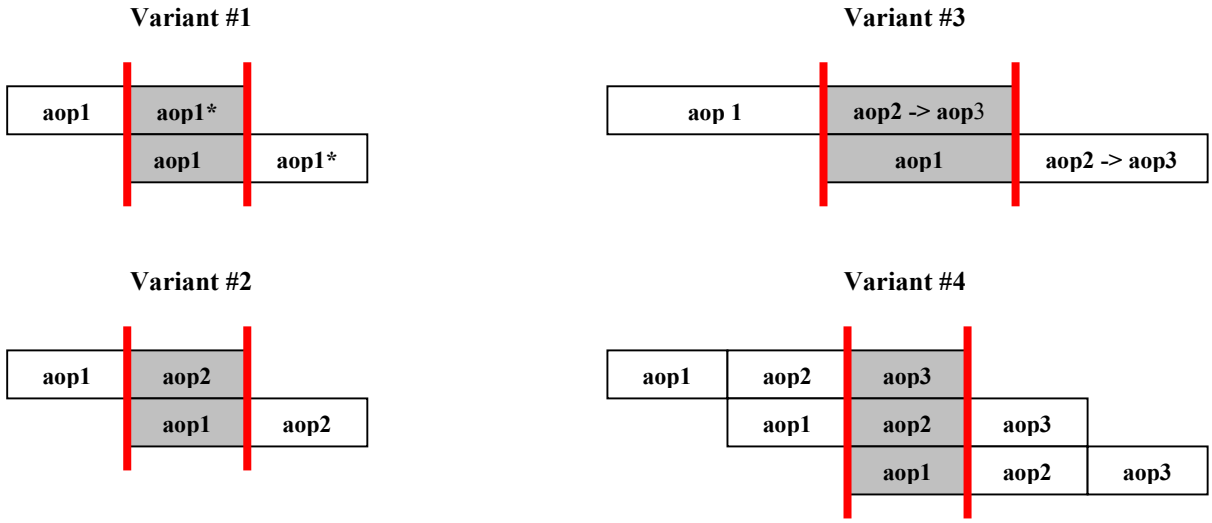

Figure 3. Graphical representation of low level pipelining

- Variant #1. It includes 2 identical consecutive arithmetic operations: $aop1$, $aop1*$. The output of $aop1$ is the input of $aop1*$. Hence,

$$Tàop1 < Taop1*. \tag{20}$$

where
$Taop1$ is the propagation delay of $aop1$,
$Taop1*$ is the propagation delay of $aop1*$.

Therefore, in this case low level pipelining requires that

$$Tclk > Taop1* \tag{21}$$

where
$Tclk$ is the period of the system clock.

In this case, the low level pipelining is 2 stage.

- Variant #2. It includes 2 different consecutive arithmetic operations: $aop1$, $aop2$. The output of $aop1$ is the input of $aop2$. In this case, the following inequalities must be satisfied:

$$\text{If } Taop1 > Taop2 \text{ then } Tclk > Taop1 \tag{22}$$
$$\text{If } Taop1 < Taop2 \text{ then } Tclk > Taop2 \tag{23}$$

where
$Taop2$ is the propagation delay of $aop2$.

In this case, the low level pipelining is 2 stage.

- Variant #3. It includes 3 different consecutive arithmetic operations: $aop1$, $aop2$, $aop3$.

The output of $aop1$ is the input of $aop2$. The output of $aop2$ is the input of $aop3$. In this case:

$$\text{If } Taop1 \geq Taop2 + Taop3 \text{ then } Tclk > Taop1 \tag{24}$$

where
$Taop3$ is the propagation delay of $aop3$

In this case, the low level pipelining is 2 stage.

- Variant #4. It includes 3 different consecutive arithmetic operations:, $aop1$, $aop2$, $aop3$.

The output of $aop1$ is the input of $aop2$. The output of $aop2$ is the input of $aop3$. In this case:

$$\text{If } Taop1 < (Taop1 + Taop3) \,\&\, (Taop1 > Taop2) \,\&\, (Taop1 > Taop3) \text{ then } Tclk > Taop1 \qquad (25)$$

In this case, the low level pipelining is 3 stage.

With all four variants, low level pipelining is possible only if every clock cycle a new value is fed to the combinational logic $aop1$.

5. Conclusion

Thoroughly studied is the pipelining in FPGA based Canny modules: Gaussian filtering, Sobel, gradient magnitude and direction, and non-maximum suppression. Defined are high-, mid- and low level of pipelining. Each level is analyzed with respect to its speed capabilities in the respective Canny module. Provided are formulae for calculating the exact number of clock cycles required to execute the computations in these Canny modules with respect to the two input variables: image size and Gaussian filter size. Mathematically proved is the upper limit of FPGA based Canny speed in terms of pipelining.

References

[1] Aravindh G., Manikandababu C. S. 2015 Algprithm and Implementation of Distributed Canny Edge Detector on FPGA *ARPN Journal of Engineering and Applied Sciences* Vol. **10** (7), pp.3208-3216

[2] Chandrashekar N.S., Nataraj K.R. 2017 A Review on FPGA Implementation of Distributed Canny Edge Detector *Proc. of Int. Conf. on Current Trends in Eng., Science and Technology*, pp. 394-399

[3] Divya. D, Sushmap S. 2013 FPGA Implementation of a Distributed Canny Edge Detector *International Journal of Advanced Computational Engineering and Networking*, Vol.**1** (5) pp.46-51

[4] Fangxin Peng, Xiaofeng Lu, Hengli Lu, Sumin Shen 2012 An improved high-speed canny edge detection algorithm and its implementation on FPGA *Proceedings of SPIE, Pattern Recognition and Basic* Technologies Conference Vol. **8350**

[5] Pallavi Ramgundewar, Hingway S .P., Mankar K. 2015 Design of modified Canny Edge Detector based on FPGA for Portable Device *Journal of The International Association of Advanced Technology and Science* Vol. **16** (2), pp. 10-16

[6] Poonam S. Deokar and Anagha P. Khedkar 2015 Implementation of Modified Distributed Canny Edge Detector Algorithm Using FPGA *International Journal of Information Research and Review* Vol. **2** (8), pp. 999-1003

[7] Shraddha Y. Swami, Jayashree S. Awati 2017 Implementation of Edge Detection Filter using FPGA *Proceedings of 49th IRF International Conference*, pp. 26-30

[8] Thombare Ashwini, S. B. Bagal 2015 A Distributed Canny Edge Detector with Threshold Segmentation *International Journal of Modern Trends in Engineering and Research*, pp. 1567-1572

[9] Supraya K, 2017 Hardware Implementation Of Canny Edge Detection Algorithm With FPGA *Journal of Engineering Science and Technology* Vol. **12**, No. 9, pp. 2536-2550

[10] Yu Chen, Caixia Deng, Xiaxia Chen 2015 An Improved Canny Edge Detection Algorithm *International Journal of Hybrid Information Technology* Vol.**8** (10), pp. 359-370