MBSEM Image Acquisition and Image Processing in LabView FPGA

Shammi Rahangdale, Paul Keijzer, P.Kruit
Department of Imaging Physics
Delft University of Technology
Delft, Netherlands.
s.l.rahangdale@tudelft.nl

***Abstract*— Multi-beam scanning electron microscopy (MBSEM), has been developed to reduce the acquisition time by scanning multiple pixels simultaneously. The signal from the 14 x 14 beams is captured on a camera which reads out the position and intensity for each beam on the sample. But as we work with multiple beams and pixels we need a powerful technique for image acquisition and image processing algorithms. We use Field Programmable Gate Arrays (FPGA's), often used as an implementation platform for real time image acquisition and processing applications, because their structure is able to exploit spatial and temporal parallelism. This paper presents a technique for dealing with the various constraints of the camera and efficient mapping for image acquisition and processing operations on FPGA.**

Keywords*— *Microscopic Imaging and analysis; 2D image acquisition; MatLab and LabView FPGA.

I. Introduction

The Scanning Electron Microscope (SEM) is a powerful tool and widely used for the inspection of biological and semiconductor samples with high resolution. However, for large area or 3D imaging, SEM imaging is a time consuming process. The MBSEM [1,2] is developed to reduce the acquisition time by scanning multiple pixels at the same time. Our MBSEM contains 196 beams. All electron beams are scanning simultaneously the signal from the beams separately guided to a detection plane and converted into 196 light 'Blobs' using a YAG screen (Fig. 1). Theoretically we could place an array of 14x14 detectors (APD's or PMT's) here. However, the 'Blobs' move with the scanning electron beams and may have a different pitch for different microscope settings.

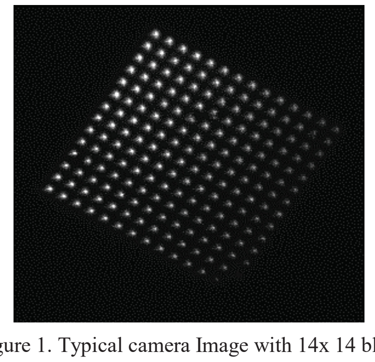
Figure 1. Typical camera Image with 14x 14 blobs

So we 'simulate' such a detector array using a fast camera. Now we need a powerful algorithm for the image processing and acquisition. We make use of a LabView FPGA application and a hardware module that contains a camera and an FPGA. This is an excellent platform for real time image processing and acquisition. There are several modes of operation that can be used to implement an image processing algorithm: streamed, offline or hybrid. The system architecture contains a hardware modules (a frame grabber with on-board FPGA, camera link, camera and other required peripheral devices) and a software module (LabView Vision and LabView FPGA). With this combination we are able to acquire all the images from the camera at a rate of several thousand frames per second and send them to the FPGA for processing.

II. Hardware Arictecture

The hardware setup of our MBSEM system (Fig. 2). Consists of a camera, a NI PXI-6259, and a NI PCIe-1473R frame grabber with on board FPGA. For detecting the signal, we require a sensor which can capture the light (photons) coming from the sample when we scan. We selected the Optronis (CL600x2 CMOS) camera, camera link with 8 bit grey levels. This camera has several desirable features for our software implementation such as Region Of Interest (ROI) selection and several modes of data streaming. The camera sends only the pixels within the ROI to the FPGA. Synchronising the camera with the beam deflection, the NI PXI-6259 sends a sync signal to the camera either directly or

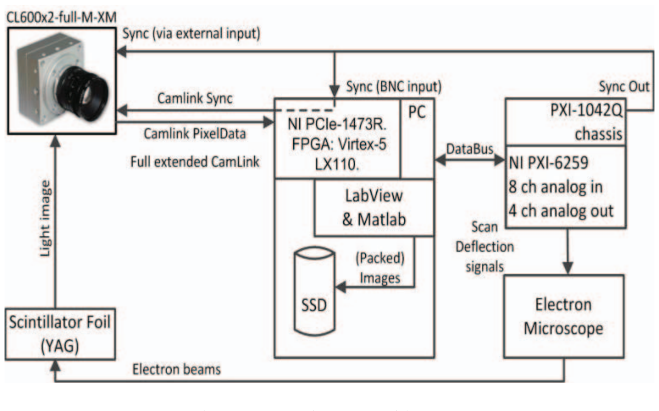

Figure 2. Hardware Architecture

via the FPGA camera link. Our hardware architecture supports both online and offline image processing: on board online processing by means of FPGA, and offline processing using raw images saved on Solid State Drive (SSD).

The pixel information of all images is to be passed through the camera's CMOS chip to the software. The camera frame size required to separate the 196 blobs is less than 200x200 pixels of 8 bits. Full configuration camera link supports 5.44 Gbit/s. USB3 and CoaXPress (CXP) can be expected to yield cameras with faster data streaming in the near future. At the moment we are using the camera link in 10 Tap/8bit mode. This means that camera link transfers groups of 80 bits (8bit grey levels of ten pixels) at a rate of 85 MHz. The camera also supports running camera link in 4 Tap/10bit mode. This offers more grey levels, but would reduce frame rate by 60%. The CL600x2-Full-M-XM camera does not support an 8 Tap/10-bit mode. So we selected a 10Tap/8bit mode. The graph in Fig. 3 shows the maximum frame rate for both modes, both for the camera link and the camera chip.
At camera frame sizes of 200x200 = 40 kB pixels and in 10 Tap/8bit mode, the camera link supports a framerate of 21,3 kFps. At this frame size the camera chip can generate 11,9 kFPS, limiting the dwell time to a minimum of 85 µs. This is 850 times slower than our ultimate goal of 100 ns.

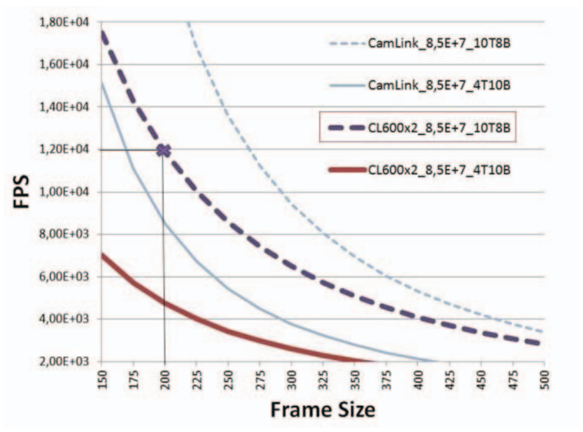

Figure 3. FPS versus Frame size

III. SOFTWARE ARICTECTURE

The software needs to determine the intensities in the 14x14 blobs from each camera image. There is one camera image per electron beam scan position. Our method includes the creation of a mask in Matlab, to facilitate fast blob-intensity determination, and the design of an image processing algorithm in LabView using Xilinx system generator. We used a Hardware Descriptive Language (HDL) code and netlists that can be synthesized and optimized using the Xilinx vivado tool. The algorithm is implemented in a Xilinx virtex-5 LX100 FPGA. The Xilinx system generator generates a bit file which can be loaded into the FPGA.

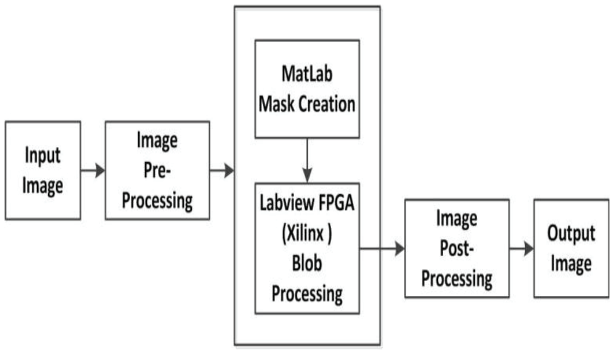

Figure 4. Software Architecture

The flow of the MBSEM imaging software architecture is depicted in Fig. 4. It includes image pre-processing, mask creation, Blob processing and post-processing. Pre-processing makes data available in a suitable format for processing and post-processing makes data readable (display) for MatLab or other FPGA platforms downstream. The MBSEM imaging algorithm is implemented in MatLab and Labview FPGA. The algorithm will be explained in the following.

A. *Mask Creation*

All 196 beams on the sample move synchronously to the next position of the scan, from left to right and from top to bottom (raster scan). The step size corresponds to the desired resolution of the final image. But with this scan, the blobs also move over the detector so in order to calculate the grey value of a blob in the camera image, the FPGA needs to know the location of that specific blob. The FPGA is not well suited for finding all blob positions for all images sufficiently fast. If the corners of the scan are known, the positions of the blobs during the scan can be derived from that. We use a linear interpolation between the coordinates of the 4 corners of the scan. Before starting the full scan four images in the four extreme corners of the e-beam deflection range are taken.

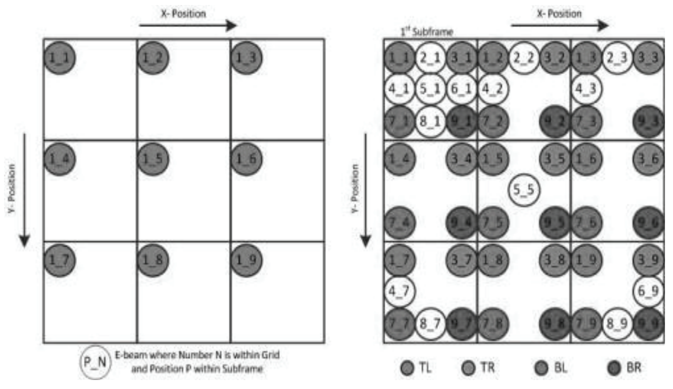
X- Position
Y- Position
1st Subframe
P_N E-beam where Number N is within Grid and Position P within Subframe
TL
TR
BL
BR

Figure 5. Example for a 9 beam grid in 9 different positions, building a 9x9=81 pixel mega frame

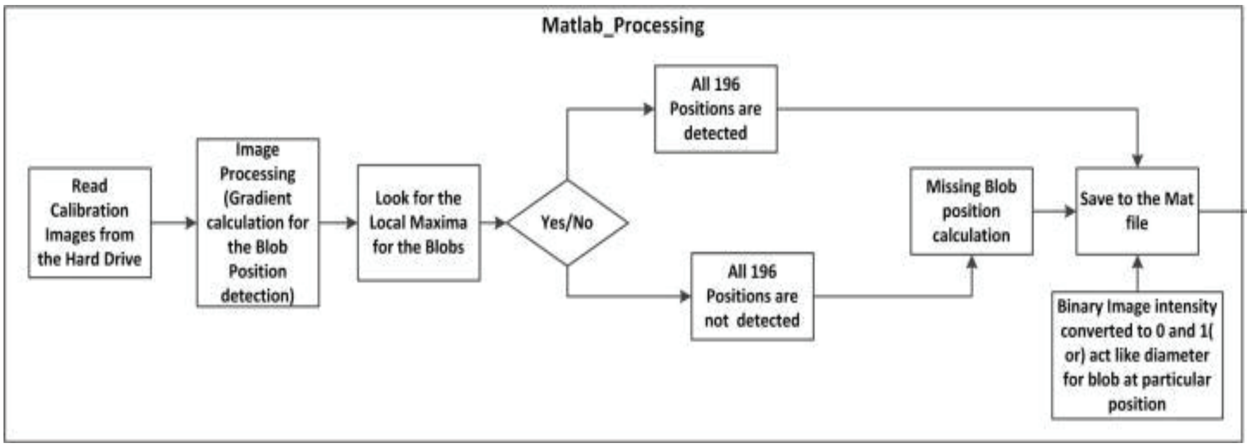
Matlab_Processing
Read Calibration Images from the Hard Drive
Image Processing (Gradient calculation for the Blob Position detection)
Look for the Local Maxima for the Blobs
Yes/No
All 196 Positions are detected
All 196 Positions are not detected
Missing Blob position calculation
Save to the Mat file
X and Y Info
Binary Image intensity converted to 0 and 1(or) act like diameter for blob at particular position

Figure 6. Mask Processing to generates coordinates for intensity calculation

Then we process these four images in MatLab (Fig. 6) to find the exact x and y coordinate of their blobs.

This is done by computing the intensity partial derivatives $\partial f/\partial x$ and $\partial f/\partial y$ at every pixel location in the image to obtain the gradient of the image. As we are dealing with digital quantities, a digital approximation of the partial derivative in the neighbourhood of a point is needed.

$$g_x = \partial f(x, y)/\partial x = f(x+1, y) - f(x, y)$$

$$g_y = \partial f(x, y)/\partial y = f(x, y+1) - f(x, y)$$

The magnitude of the vector ∇f, denoted as $M(x, y)$, is given by

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{(g_x^2 + g_y^2)}$$

M (x, y) is the value of the rate change in the direction of the gradient vector. The next step is to determine the position of the blobs by finding the local maximum for each blob and refining the position with a gaussian fit via maximum likelihood estimation. By repeating this method we can calculate the x and y coordinates for all the blobs. After finding the coordinates our algorithm cross checks that we have information from all 196 blobs. If not, then by finding the distance in x and y direction with respect to the neighboring blobs we can estimate the coordinates of the missing blobs. After following the above steps for the four extreme corner images, the masks coordinates have been created and named as TL, TR, BL and BR (Fig. 5). Later we can use these four masks to calculate the intermediate mask coordinates for finding the intensity of the blobs.

B. Image Processing in the FPGA

Algorithms for image processing are mainly classified in three levels: low, intermediate and high. Low-level algorithms operate on individual pixels or neighborhood's. Intermediate-level algorithms convert pixel data into a

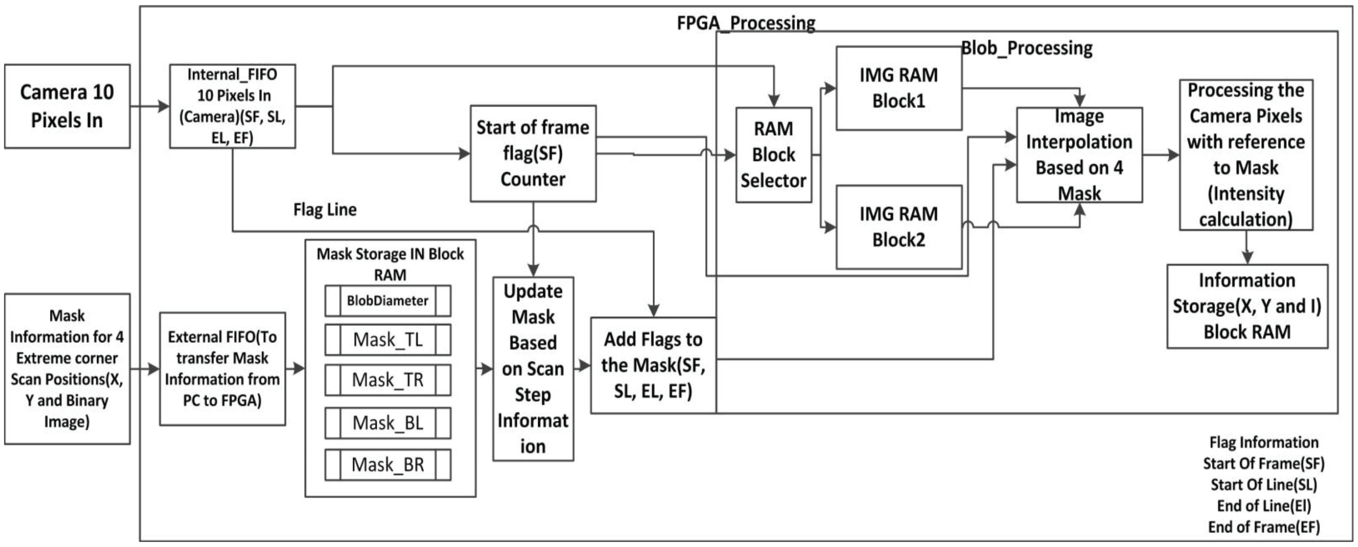
FPGA_Processing
Blob_Processing
Camera 10 Pixels In
Internal_FIFO 10 Pixels In (Camera)(SF, SL, EL, EF)
Start of frame flag(SF) Counter
RAM Block Selector
IMG RAM Block1
IMG RAM Block2
Image Interpolation Based on 4 Mask
Processing the Camera Pixels with reference to Mask (Intensity calculation)
Information Storage(X, Y and I) Block RAM
Flag Line
Mask Information for 4 Extreme corner Scan Positions(X, Y and Binary Image)
External FIFO(To transfer Mask Information from PC to FPGA)
Mask Storage IN Block RAM
BlobDiameter
Mask_TL
Mask_TR
Mask_BL
Mask_BR
Update Mask Based on Scan Step Information
Add Flags to the Mask(SF, SL, EL, EF)
Flag Information
Start Of Frame(SF)
Start Of Line(SL)
End of Line(El)
End of Frame(EF)

Figure. 7. Block diagram of the FPGA based calculation for finding intensities using a mask

different representation. High-level algorithms aim to extract meaning from the image using information from another level. So in moving from low to high-level representation there is a corresponding decrease in exploitable parallelism due to the change in pixel data to more descriptive representation. Our algorithm is designed at the intermediate level to reach the processing speed while using the parallelism offered by FPGA. Fig. 7 shows the steps involved in the proposed algorithm to reduce the processing time. Groups of pixels [4] are acquired from the camera and sent to one of the IMG RAM blocks to buffer the complete image. Two IMG RAM blocks dynamically update alternately to fill and process the image at the same time. The algorithm uses four extra RAM blocks to store the masks coordinate that have been generated in the calibration sequence (refer to above section). A counter counts the Start of Frame (SF) flags to update the intermediate mask using interpolation. The interpolation needs the four masks coordinate (X, Y) and frame number from the SF counter to update a new mask for the upcoming image. This update is used for calculation of the intensity around those coordinates. The intensity information is stored in the storage RAM block before it is sent to the host PC.

C. Device Utilization:

The optimized MBSEM imaging algorithm with Virtex-5 consumes 26.7% of the total resources present on the FPGA. So there is room to extend the FPGA design while adding extra functionalities like single beam formation or to control the microscope parameters. The compilation result and device utilization are given in the following table.

TABLE I. FPGA DEVICE UTILIZATION TABLE

Device Utilization	Used	Total	Percent
Total Slices	4616	17280	26.7
Slice Registers	9974	69120	14.4
Slice LUT's	10283	69120	14.9
Block RAM's	52	128	40.6
DSP's	0	64	0.0

IV. SUMMARY AND CONCLUSION

This paper presents the implementation of a camera with frame grabber (LabView), mask creation (MatLab) and image processing (LabView FPGA) for the MBSEM Microscope. This is a general method for online image processing using a calibration mask. The use of masks in image processing will accelerate the overall processing time. In later stage we want to increase the frame rate by several orders of magnitude using faster imaging sensor and further customizing our algorithm for the sensor. This method, combining masks and FPGA image processing can be used for basic operations in image processing like template matching and object detection.

REFERENCES

[1] A. Mohammadi-Gheidari, P. Kruit, Electron optics of multi-beam scanning electron microscope, Nuclear Instruments and Methods in Physics Research Section A, Volume 645, 21 July 2011, Pages 60–67.

[2] A.L. Eberle, S. Mikula, R. Schalek, J.W. Lichtman, M.L. Knothetate & D. Zeidler, High-resolution, high-throughput imaging with a multi-beam scanning electron microscope, Journal of Microscopy, Vol. 259, Issue 2 2015, pp. 79-164.

[3] Downton, A and Crookes, D., Parallel Architecture for Image Processing , Electronics & Commuincation Engineering Journal, vol. 10, pp. 139-151, Jun, 1998.

[4] Ali, K.M.A.; Ben Atitallah, R.; Hanafi, S.; Dekeyser, J.-L., "A generic pixel distribution architecture for parallel video processing," in ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on , vol., no., pp.1-8, 8-10 Dec. 2014.