Real-time Embedded Image Processing on the PowerFFT™ Processor

Laurens Bierens
doubleBW Systems B.V.
Delftechpark 26, NL-2628 XH Delft, The Netherlands
Phone: +31 (0)15 2600 432 Fax: +31 (0)15 2600 431
E-mail: bierens@doubleBW.com

Abstract — **In this paper we present the PowerFFT processor – a Fast Fourier Transform (FFT) optimized DSP developed by doubleBW – in image processing. The PowerFFT is available in a turn-key PC-based prototyping environment, including programming software libraries. We elaborate on common operations that occur in imaging applications such as infrared and optical reconnaissance, industrial vision, medical imaging, etc. Typically these operations can be represented as real-time convolution or correlation of a stream of frames with a two-dimensional kernel. Such image processing operations are widely used for image enhancement, edge detection, template matching, etc.**

Keywords— **image processing; two-dimensional signal processing; real-time processing; FFT; DSP.**

I. INTRODUCTION

The PowerFFT has an unprecedented performance in two-dimensional signal processing. It combines flexibility (programmable), high-accuracy (floating point), extreme performance (5.7 GFLOPS per processor) and low-power consumption (less than 2W per processor). The processor is specifically designed for high-performance multi-dimensional signal processing [1,2] such as required in real-time image processing.

We elaborate on general real-time signal processing applications in imaging applications, like industrial vision, infrared and optical reconnaissance, and medical imaging, see e.g. [3,4]. Usually, image frames are grabbed from a CCD camera producing grayscale images, i.e. images with pixels of n-bit representing 2^n grey levels. The challenge is the real-time convolution or correlation of a stream of frames with a two-dimensional kernel. Such image processing operations are widely used for image enhancement, edge detection, template matching, etc.

Image processing on a standard platform, like a PC or workstation, is often not an option for two main reasons:

- The application is time-critical;
- The amount of data generated by the CCD camera is too large.

Popular alternative solutions for real-time image processing are:

1. Parallel DSP computing;
2. FPGA based dedicated hardware.

Parallel DSP computers are flexible as they can be programmed in standard C/C++-language. Nevertheless, the solution is bulky (typical it consist of multiple DSP boards) and the power consumption is proportionally high. Further, the costs of such systems are high and they require a substantial software engineering effort for optimal code implementation.

Dedicated FPGA based solutions may be smaller, typically a single board containing a few FPGAs and some memory components. However, this solution requires a substantial hardware engineering effort (VHDL programming). Although FPGAs are re-configurable, it is also a rigid solution as the processing capability of FPGAs is often inversely proportional to the amount of flexibility. Further, the high power consumption for FPGAs operating on such high data rates is often not desirable in embedded applications.

Therefore, we will present a generic solution for such real-time image processing problems based on the PowerFFT. We show that the PowerFFT is superior in terms of performance, design time, and footprint, compared to the alternatives above. As a standard PowerFFT PCI Card in a mid-range PC it ensures a sustained image processing performance of 17 frames/sec for 1K x 1K grayscale images inclusive of displaying the images. The programming of the PowerFFT is done using the PowerFFT Development Kit [5], a user friendly integrated development environment which assists the engineer in designing and optimizing custom algorithms for the PowerFFT (the concept of the compiler is presented in [6]).

The application itself is written in standard C, and can be embedded in MatLab or Labview environments. The PowerFFT hardware can be readily embedded in

mega-pixel camera or frame grabber environments, as we show with an application example.

The outline of this paper is as follows. First, in Section II we present a generic algorithm description of the image processing kernel. Then we introduce in Section III the PowerFFT and its unique features. In Section IV we show how the image processing kernel is programmed on the PowerFFT. We discuss the performance issues and give application examples using a turn-key PowerFFT image processing prototyping platform in Sections V and VI, respectively.

II. Algorithmic Description

A. FFT Theory

In this section we provide the basic formulas used in the following sections. Many of them can be found in DSP and FFT textbooks, such as [7]. For convenience, we describe them in one-dimensions, but the extension to two-dimensions is straightforward using and additional index. Let x[n], n = 0,1,...,N-1 and h[m], m = 0,1,...,M-1, be two digital sequences with $M \le N$. Then the following operations are defined as:

- Convolution of h with x:

$$y[n] = \sum_{m=0}^{M-1} h[m]x[n-m]$$

- Correlation of h with x:

$$y[n] = \sum_{m=0}^{M-1} h[m]x[n+m]$$

- Circular convolution of h with x:

$$y[n] = \sum_{m=0}^{M-1} h[m]x[(n-m) \bmod N]$$

- N-points Discrete Fourier Transform (DFT) of x:

$$X[m] = \sum_{n=0}^{N-1} x[n]W_N^{-nm}, W_N = e^{2\pi j/N}$$

- N-points Inverse DFT (IDFT) of X:

$$x[m] = \sum_{n=0}^{N-1} X[n]W_N^{nm}$$

It is well known that in case N is a power of 2, an N-point (I)DFT can be calculated efficiently using an N-point Fast Fourier Transform (FFT). The number of operations required for the calculation is in the order of NlogN instead of N^2. If the length of the sequence x is not a power of 2 it should be padded with zeros until its length is a power of 2.

B. The generic image processing kernel

The generic image processing kernel can be described mathematically as follows:

$$\mathbf{R} = \mathbf{H} * \mathbf{I}$$

where $*$ denotes the two-dimensional convolution[1], and

I $N_1 \times N_2$ image
H $M_1 \times M_2$ kernel, with $M_1,M_2 \le N_1,N_2$
R $N_1 \times N_2$ convolution result

[1] Correlation is considered here as a special case of convolution.

Well known is the fact that circular convolution can be executed efficiently in frequency domain using the FFT. Let L_1 and L_2 be the nearest power of 2 larger then or equal to N_1 and N_2, respectively. Let **I′** and **H′** be the zero padded versions of **I** and **H**, respectively, and let $\boldsymbol{I}$ and $\boldsymbol{H}$ be the FFTed images of **I′** and **H′**, then, according to the convolution theory the following hold:

$$\boldsymbol{R} = \boldsymbol{H} \times \boldsymbol{I}$$

with $\boldsymbol{R}$ the FFTed image of **R′**. The multiplication denotes the element-wise multiplication. This principle is known as “fast convolution” or “frequency domain convolution”[2] and is generally applied for reasons of computational efficiency. Further, we introduced the “shift parameters” K_1 and K_2 in the zero padding of the kernel **H**. This results in an equivalent shift of **R** in **R′**. The shift operation is defined in Figure 1 (we only define this for the one-dimensional case, as the two-dimensional extension is straightforward).

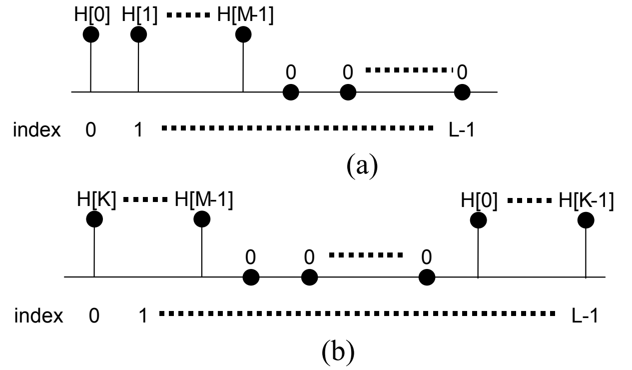


Figure 1. Shift operation: no shift, K=0 (a), and negative shift of -K (b). Note that the shift operation is defined regarding the FFT-length L. Positive shift is straightforward.

Generally, the FFTed kernel $\boldsymbol{H}$ can be pre-computed as it is fixed for each incoming **I**. Further, note that according to the convolution theory if $L_1 = N_1$ and $L_2 = N_2$, the frequency domain convolution equals a circular convolution. An algorithm could now look as follows:

```
H′ ← Zeropad_Kernel(H,M1,M2,L1,L2,K1,K2)
H ← FFT_Kernel(H′,L1,L2)
For each frame I
        I′ ← Zeropad_Image(I,N1,N2,L1,L2)
        I ← FFT_Image(I′,L1,L2)
        R ← Element_Multiply(H,I,L1,L2)
        R′ ← IFFT_Image(R,L1,L2)
        R ← Select_ROI(R′,xroi,yroi,P1,P2)
Endfor
```

Note that we introduced a Region Of Interest (ROI) defined by the parameters (x_{roi},y_{roi}), the index of the first element of the ROI, and (P_1,P_2), which is the size

[2] Note that according to the convolution theory [7] correlation in frequency domain is simply done by conjugation of $\boldsymbol{H}$.

of the ROI. For the ROI parameters the following hold:

$$0,0 \le x_{roi}, y_{roi} < L_1, L_2 \text{ and } P_1, P_2 \le L_1 - x_{roi}, L_2 - y_{roi}$$

In Figure 2.a a block diagram of the algorithm is shown and in Figure 2.b the geometry of **I**, **H**, and **R** in relation to the parameters.

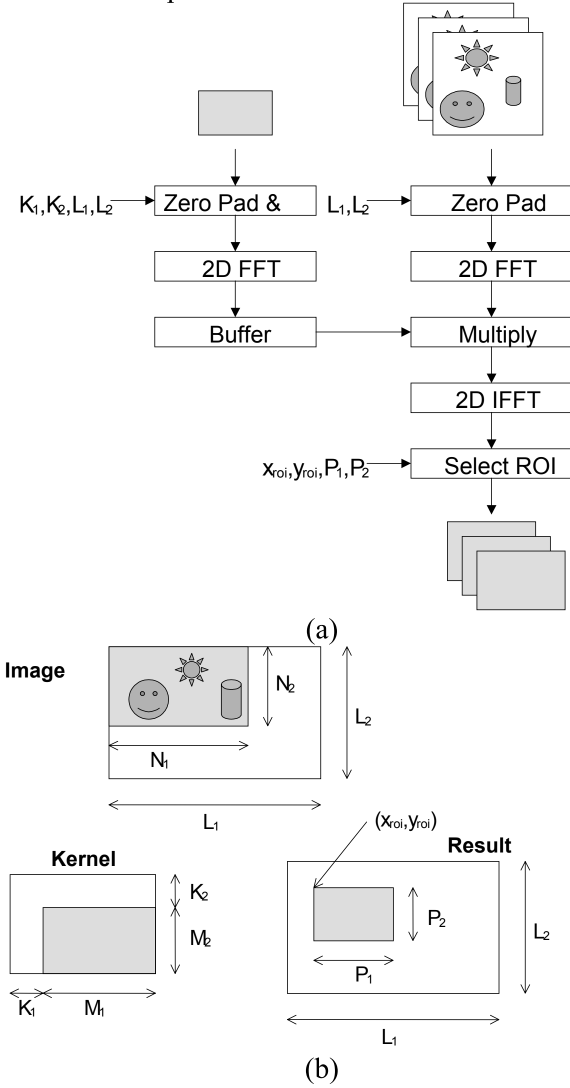


Figure 2. Block schematic of the image processing algorithm (a) and geometry of *I*, *H*, and *R* in relation to the parameters.

The following gives some examples of how to choose the image size, kernel size and the FFT size to select full convolution or circular convolution of **H** and **I**. We assume that the $K_1 = K_2 = x_{roi} = y_{roi} = 0$, as they do not affect the convolution results, only the ordering of the pixels.

- Full (linear) convolution (Figure 3.a):

  $N_1+M_1-1 \le L_1$, $N_2+M_2-1 \le L_2$,
  $P_1 = N_1+M_1-1$, $P_2 = N_1+M_1-1$

- Circular convolution (Figure 3.b):

  $N_1 = L_1$, $N_2 = L_2$, $P_1 = N_1$, $P_2 = N_1$

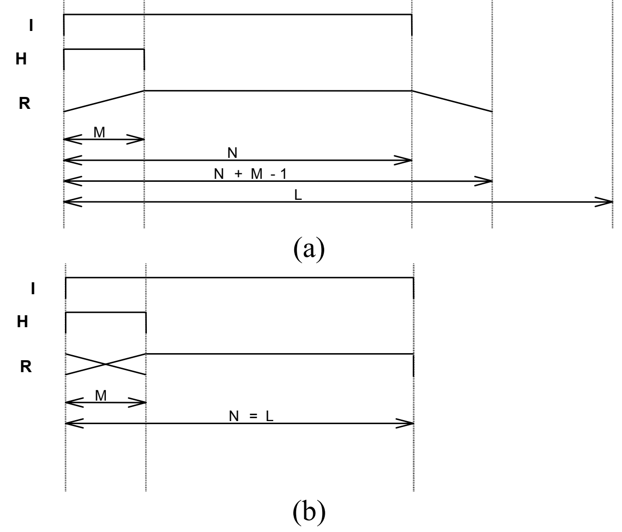


Figure 3. Linear convolution (a) and circular convolution (b).

III. GENERAL FEATURES OF THE POWERFFT

The Fast Image Processing Kernel is basically a mapping of the algorithm described in the previous section onto the PowerFFT architecture. The general structure of the PowerFFT is shown in Figure 4 [2].

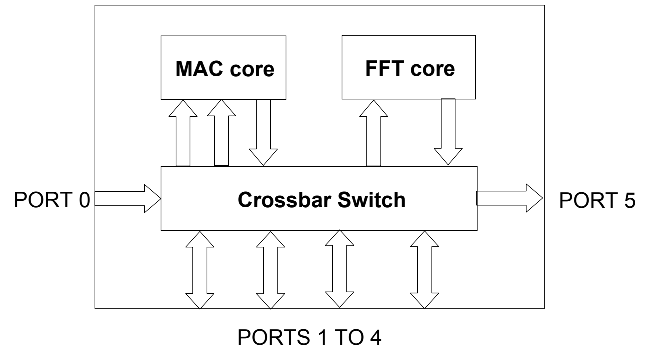


Figure 4. General structure of the PowerFFT.

The processing core offers an optimized FFT processing capability, among others:

- Up to 1024 pts FFT/IFFT, incl. windowing;
- Up to 1024 pts convolutions / correlations;
- Up to 1024 pts FFT/IFFT + vector multiply.

The PowerFFT has the option to use it in combination with 4 SDRAM memory banks. The processing core's capability is extended with memory banks to, among others, the following functionality:

- Up to 1M pts FFT/IFFT, incl. windowing;
- Up to 1M pts convolutions / correlations;
- Up to 1024 x 1024 pts 2D FFT/IFFT, incl. windowing;
- Up to 1024 x 1024 pts convolutions / correlations.

An on-chip crossbar switch takes care of the communication between input, output, memory banks, and processing core, allowing concurrent I/O and

processing. For example, the following actions can be executed concurrently:

- Load sequence n from input to memory bank 1;
- Process sequence n-1 from memory bank 2 and store result in memory bank 4;
- Write sequence n-2 from memory bank 4 to output.

The unique architecture of the PowerFFT eliminated the usual I/O bottlenecks in stream based processing. Further we should note explicitly that one of the key operations in the image processing in combination with SDRAM memory is the transposition of the data set. Reading and/or writing data from and to SDRAM is usually fast in only one direction if we use conventional memory addressing. However, with a proprietary addressing scheme the PowerFFT is able to read and/or write from and to SDRAMs in two dimensions without significant performance penalty, which makes the PowerFFT superb for image processing applications. Figure 5 shows a picture of the PowerFFT PCI Card.

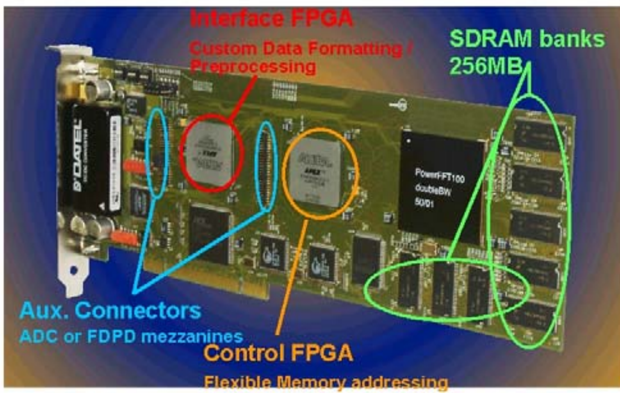


Figure 5. PowerFFT PCI Card for usage in standard desktop PCs. Indicated are some key items added to assess the full functionality of the PowerFFT.

IV. IMAGE PROCESSING KERNEL FOR THE POWERFFT

Now let us consider the algorithm described in Section II.B. The first step in this algorithm is preparation of the kernel. The kernel is calculated in advance, and usually there is no time critical requirement in the preparation. The kernel is loaded from the input, zero padded and stored in SDRAM 1. Then, using the intermediate SDRAM 2, the kernel is first horizontal FFTed, and then vertically FFTed. The 2D spectrum of the kernel is again stored in SDRAM 1 where it remains during the processing.

The actual image processing is now somewhat more complicated. As we want the maximum image processing throughput, we should pipeline the process. First we consider the image processing for a single image, i.e. no pipelining is required. Then we can do the image processing in 6 steps. For each step we indicate the source, the process, and the target, as shown in Table 1. A graphical view of 6 steps is shown in Figure 6. In the Appendix A a simple MatLab program is given that emulates the steps.

Table 1. Image processing steps.

Step	Source	Process	Target	Description
1	P0	Load	P1	Load image
2	P1	FFT rows	P2	Row FFT image
3	P1, P2	FFT columns and multiply with kernel	P3	Col FFT and mpy image
4	P3	IFFT columns	P2	Col IFFT image
5	P2	IFFT rows	P4	Row IFFT image
6	P4	Offload	P5	Offload image

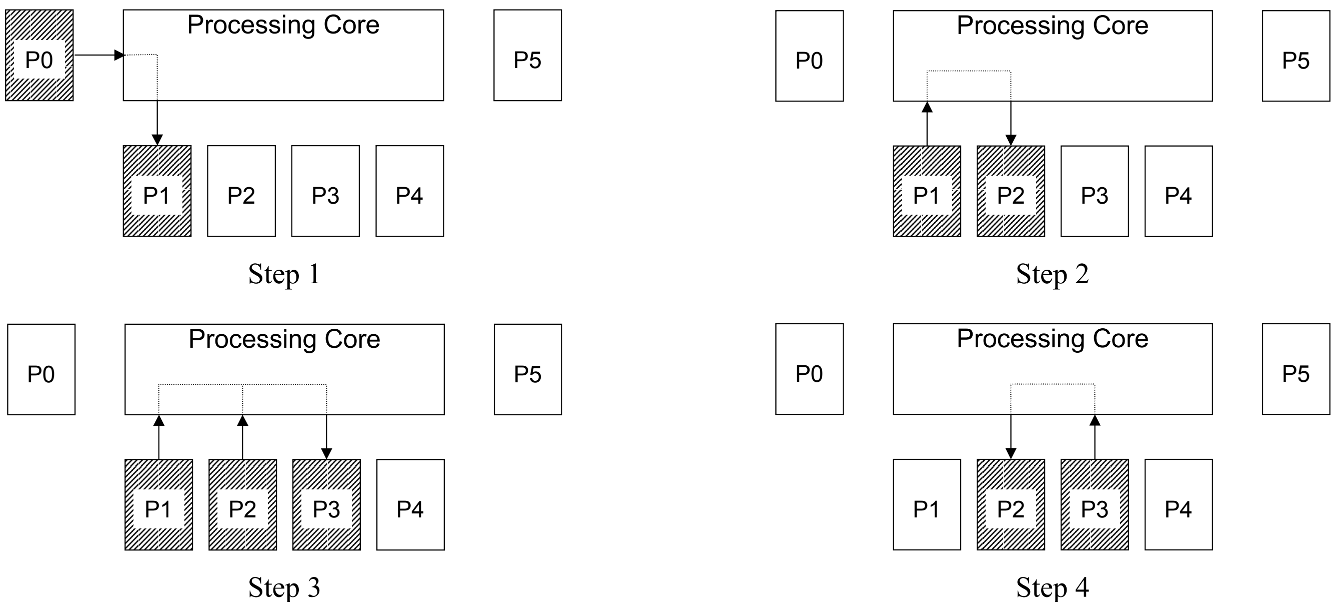


Figure 6. Image processing on the PowerFFT in six steps. Note that the kernel spectrum is pre-stored in SDRAM 1.

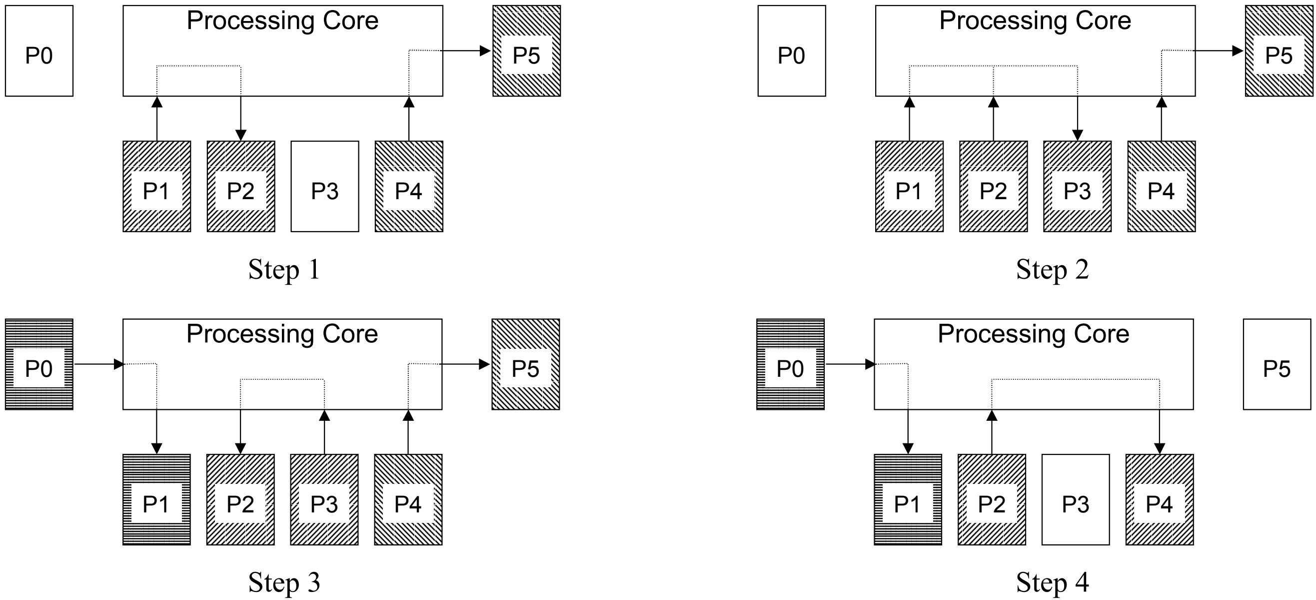


Figure 7. Pipelined image processing on the PowerFFT. The successive image I(j-1), I(j) and I(j+1) are depicted by the "SE→NW" diagonal pattern, the "SW→NE" diagonal pattern, and the horizontal patterns, respectively.

As we can see form Figure 6, Step 1 occupies only one of the four SDRAM banks, and the same holds for Step 6. Both of these steps do not use the processing core. During Steps 2 to 5, the true "processing" steps, we see that at least one SDRAM bank is free. Thus we can introduce the pipelining principle, which is basically the following.

Assume that we have at least three images in the pipeline: I(j-1), I(j) and I(j+1). Then pipelining means that loading I(j+1) and offloading I(j-1) can be executed during the processing of I(j).

As we already saw from Figure 6, the memory resources are available for the pipelining. Then we can "squeeze" the algorithm to be executed in only four steps instead of six steps. This is shown in Figure 7, where the different patterns for the source and targets indicate occupation for I(j-1), I(j) or I(j+1). Note that Step 1 assumes that I(j) is loaded into P1, and that I(j-1) is in P4 to be offloaded. The time schedule of the image processing with and without pipelining is shown in Figure 8.

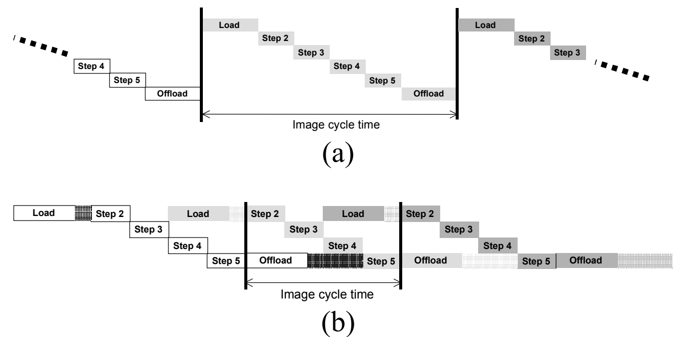


Figure 8. Pipelining issues. In (a) no pipelining is applied. Note that Step 1 and Step 6 are named Load and Offload according to their functionality. In (b) it is shown the effect of pipelining. The Load and Offload time can be subtracted from the image cycle time (frames per second = 1 / image cycle time). The shaded areas in (b) indicate the additional time available for the Load and Offload without performance penalty.

V. PERFORMANCE ISSUES

From Figure 8.b we can now determine the image cycle time in terms of the "time per step". First we assume that the time for Step 2 to 5 is the same, say T_{step}. This only holds for square images, and simplifies the performance analysis, but the extension to more general cases is straightforward. Further we observe that in case the offload time $T_{off} \leq 3T_{step}$, and the load time $T_{load} \leq 2T_{step}$, then the image cycle time is $4T_{step}$. However, if $T_{off} > 3T_{step}$ and $T_{load} > 2T_{step}$, then the image cycle time is extended with T_{off}-$3T_{step}$ and T_{load}-$2T_{step}$, respectively. In summary, we can derive:

image cycle time

$$= 4T_{step}+\min\{0, T_{off}-3T_{step}\}+\min\{0,T_{load}-2T_{step}\}$$
$$= 4T_{step} + T_{io}$$

where $T_{io} = \min\{0, T_{off}-3T_{step}\}+\min\{0,T_{load}-2T_{step}\}$ is the additional time needed for the I/O of the data. If $T_{io} > 0$ then the performance bottleneck is due to the I/O. If $T_{io} = 0$, then the performance bottleneck is due to the processing.

In order to give performance values, we need to calculate T_{step}. Note that from Table 1 we know that Steps 2 to 5 consists of straightforward FFTs. We assume that the image size is $N \times N$, N is a power of 2, and the processing is cyclic convolution (again, these assumptions are not mandatory but only for the sake of simplicity). Further, we assume that $N \leq 1024$. Larger image sizes can be processed by the PowerFFT but the current memory sizes do not allow this. Furthermore, FFTs larger than 1K points cannot be processed completely on-chip, and must use the memory banks. Although this is feasible it complicates the algorithm development unnecessarily for the scope of this paper. From the PowerFFT data sheet we know that the time to perform an N-point FFT is given as $T_{fft} = N/F_{pfft}$ where F_{pfft} is the I/O clock speed of the PowerFFT. Steps 2 to 5 each consists of N subsequent FFTs, thus

$$T_{step} = N \cdot T_{fft} = N^2/F_{io} \text{ and}$$
$$\textit{image cycle time} = 4N^2/F_{pfft}$$

Thus the frame rate (= 1 / *image cycle time*) is inverse proportionally to the image size. In Figure 9 the frame rate as function of the image size is plotted for different F_{pfft}. The frame rates are also listed in Table 2.

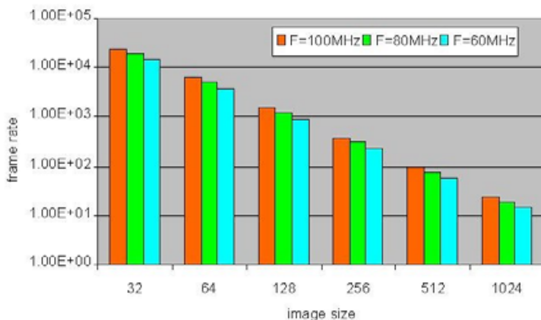


Figure 9. Frame rate plotted for different F_{pfft} and image size.

Table 2. Frame rate for different F_{pfft} and image size

N	Frame rate		
	F_{pfft}=100MHz	F_{pfft}=80MHz	F_{pfft}=60MHz
32	24.5e3	19.5e3	14.6e3
64	6.1e3	4.9e3	3.7e3
128	1.5e3	1.2e3	0.9e3
256	380	310	230
512	95	76	57
1024	24	19	14

VI. APPLICATION EXAMPLE

A real-time image processing kernel evaluation platform has been developed based on standard mid-range PC (Pentium III, 500MHz, 256MB work memory) and a low-end PCI frame grabber card. In Figure 10 a schematic view of the system is shown. A simple digital camera can be connected to the frame grabber.

For the purpose of a demonstration a simple web-cam type of camera has been selected, and 512 x 512 pixel frames at a frame rate of 25 frames per second are grabbed. Note that the frame grabber dumps its 8bit grayscale frames into the PC's main memory, and from there the data is transfered to the PowerFFT Card. Direct transfer of the frames from frame grabber to the PowerFFT Card would allow a much more efficient use of the available PCI bandwidth. Nevertheless, we are able to process the 25 frames/sec second with user-defined kernels, and display the images on the PC-monitor. In Figure 11.a a picture of the system is shown and in Figure 11.b the real-time display of the camera image is shown.

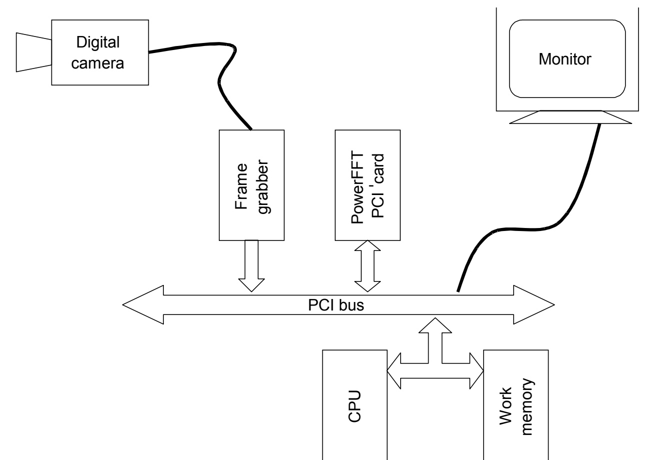
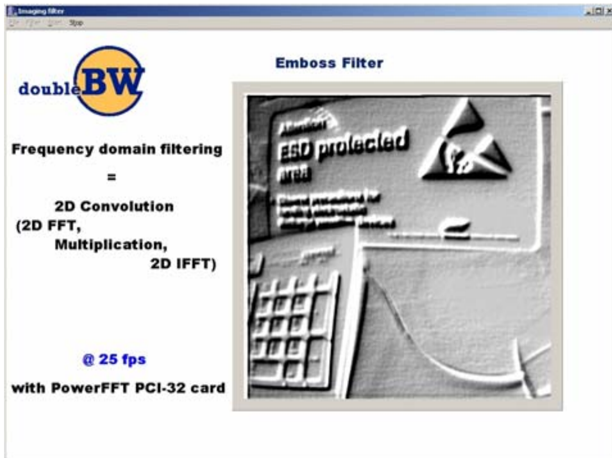


Figure 10. Block schematics of a real-time image processing prototyping system.

(a)



(b)

Figure 11. Picture of the image processing system including camera (a) and a screen dump of the real-time display (b).

VII. Conclusions

We presented a prototyping platform for real-time embedded image processing applications based on the PowerFFT. The platform allows application engineers to develop their algorithms and application in common PC-like environments, and evaluate the application immediately in a real-life situation using a standard digital camera, a PCI frame grabber, and the PowerFFT PCI Card.

The platform is targeted at general real-time image processing applications, like industrial vision, infrared and optical reconnaissance, and medical imaging. These applications are often related to real-time convolution or correlation of a stream of frames with a two-dimensional kernel. Such image processing operations are widely used for image enhancement, edge detection, template matching, etc.

Simple software library calls in C/C++ or MatLab are available to support the evaluation of these algorithms using a digital camera connected to the PowerFFT Card through the PCI bus. The performance of such applications range from 17 frames/second for an 1K x 1K images to a few hundred frames/second for 256 x 256 images.

Appendix A. Example for Algorithm Simulation in MatLab

In the following we present a MatLab code example which simulates the image processing kernel as implemented on the PowerFFT (see Section IV) in MatLab. The buffers P0 to P5 refer to the ports of the PowerFFT as shown in Figure 4.

```
%% Image of size NxN prestored on input
%% P0(1:N,1:N) Kernel spectrum is
%% already precalculated and is located
%% in P1(N+1:2*N,1:N)

N = ; %% power of 2

%% Step 1: Load image
P1(1:N,1:N) = P0(1:N,1:N);

%% Step 2: Row FFT image
for j=1:N
  P2(j,1:N) = fft(P1(j,1:N));
end;

%% Step 3: Col FFT and mpy image
for j=1:N
  P3(1:N,j) = ...
    P1(N+1:2*N,j).*fft(P2(1:N,j));
end;

%% Step 4: Col IFFT image
for j=1:N
  P2(1:N,j) = ifft(P3(1:,N,j));
end;

%% Step 5: Row IFFT image
for j=1:N
  P4(j,1:N) = ifft(P2(j,1:N));
end;

%% Step 6: Offload image
P5(1:N,1:N) = P4(1:N,1:N);
```

Appendix B. Standard Library Routines in C/C++

The complete image processing kernel application software is based on PowerFFT specific C/C++ library routines:

1. `Init2DConvolution()` initializes the convolution engine;
2. `ConvSendKernel()` sends the kernel to the convolution engine;
3. `ConvSendImage()` sends an image to the convolution engine;
4. `ConvReceiveResult()` receives the result of a image correlation in the convolution engine;
5. `ResetConvolution()` resets the convolution engine.

These library routines are developed using the PowerFFT Development Kit [5,6], an integrated development environment for standard PowerFFT cards.

A simple “plug&play” C-application is shown below. On a standard mid-range PC including the

PowerFFT PCI32 Card a sustained frame rate of 1K x 1K pixel frames of 17 frames/sec is measured, independent of the kernel type. This application software allows rapid prototyping of many real-time image processing applications, like template matching, image enhancement, etc.

```
int main(void)
{
  /* parameter list that determines */
  /* the configuration of the image */
  /* processing kernel (values are  */
  /* chosen arbitrary               */
  int kwidth = 16; int kheight = 16;
  int kformat = 1; int kstartx = 0;
  int kstarty = 0; int iwidth = 1024;
  int iheight = 1024; int iformat = 8;
  int xroi = 0;int yroi = 0;
  rwidth = 1024; int rheight = 1024;
  int rformat = 8; int rscale = 0;
  int no_frames = 20;int corr = 0;
  int dcadd = 0;
  char *resdata, *imgdata, *kerneldata;
  int rc;

  /* do memory management */
  rc = fn_conv_init(kwidth, kheight,
                    kformat, kstartx,
                    kstarty, iwidth,
                    iheight, iformat,
                    xroi, yroi,
                    rwidth, rheight,
                    rformat, rscale,
                    no_frames, corr,
                    dcadd);

  if(rc == SENDKERNEL)
    rc = fn_conv_send_kernel(kerneldata);

  while(rc != INITCONV)
  {
    switch(rc)
    {
      case RECVIMAGE:
        rc = fn_conv_recv_result(resdata);
      /* start post-processing thread */
        break;
      case SENDIMAGE:
        rc = fn_conv_send_image(imgdata);
        /* start pre-processing thread */
        break;
      default: break;
    }
  }

}
```

ACKNOWLEDGEMENT

The author wish to acknowledge Friso Brugmans for his useful input on the programming examples.

REFERENCES

[1] P.C.R. Beukelman and L.H.J. Bierens, Fastest floating-point single-chip FFT processor, Proc. ProRISC99, STW, 1999

[2] PowerFFT Datasheet, Version June 2002, download from www.doubleBW.com

[3] A.A. Hastbacka, A fast patter recognizer for autonomous target recognition and tracking for advanced naval attack missiles, Proc. SPIE Aerosense, 2001

[4] N. Voss, B. Mertsching, Design and omplementation of Accelerated Gabor filter bank using parallel hardware, FPL2001, pp. 451-460, G. Brebner, R. Woods (Eds.), Springer-Verlag, 2001

[5] PowerFFT Development Kit Data Sheet, download from www.doubleBW.com

[6] N. Kopp, Compilers and Software for Data-driven Reconfigurable Embedded DSP Architectures, Proc. Progress 2002

[7] W.W. Smith and J.M. Smith, Handbook of Real-Time Fast Fourier Transforms, IEEE, 1995