Fast Visual Feature Selection and Tracking in a Hybrid Reconfigurable Architecture

Alessandro Bissacco Jason Meltzer Soheil Ghiasi
Majid Sarrafzadeh Stefano Soatto

UCLA CSD-TR050002

This report details the development of a fast visual feature tracking system which takes advantage of dedicated hardware to perform the computationally intensive step of selection. A software system uses the output of the hardware selector to develop tracks using filtering and data association techniques, and image-based validation.

1 Introduction

Advances in powerful, sometime inexpensive reprogrammable hardware has opened up opportunities for implementing vision systems in real-time. One existing paradigm that would benefit tremendously from dedicated hardware is feature tracking, which matches discrete locations in space across adjacent frames in video. Current feature trackers, such as multi-scale Lucas-Kanade [3], can track many hundreds of points at 60Hz or greater, but suffer from limitations on the speed of selection (for adding new points to the tracker) and require nearly all the available processing power of the CPU. By using separate, custom hardware to select hundreds of points per frame then associating these points to tracks in software, we can achieve greater than 60Hz real-time tracking without burdening the primary CPU, which can use the output of the system for other tasks, such as structure from motion.

1.1 Overview of the system

Our tracking system consists of a number of hardware and software components which operate in serial and parallel. Images are initially processed by a field programmable gate array (FPGA) which runs a modified Harris corner detector, taking the local maxima of scores above a threshold as feature points. These are provided to the tracking system, which uses a second-order Kalman filter to predict feature locations on the current frame. Given the prediction, previous tracks, and newly selected points from the FPGA, a data association technique is used to match new points to existing tracks (see section 3.1). To assist the matching process, we use normalized cross-correlation to prune incorrect correspondences. Finally, the new tracks are used to predict the feature locations in the next frame.

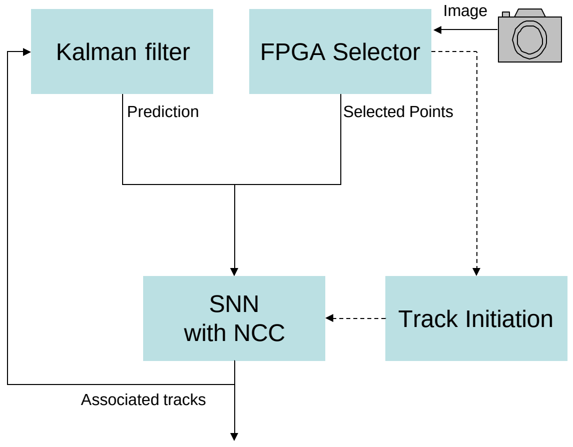

Figure 1: The components and flow of the tracking system are illustrated above.

2 Feature selection

Feature selection is implemented on a reconfigurable computer based on Field Programmable Gate Arrays (FPGA's). The method, proposed in [9], is a simplification to the Tomasi and Kanade selection technique [10] requiring only summations and multiplications on integers.

In the standard Tomasi-Kanade selection, for each point p in the image we compute the following matrix:

$$G = \begin{bmatrix} \sum_{k=1}^{N}(I_x^k)^2 & \sum_{k=1}^{N} I_x^k I_y^k \\ \sum_{k=1}^{N} I_x^k I_y^k & \sum_{k=1}^{N}(I_y^k)^2 \end{bmatrix} = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \quad (1)$$

by summing over a window of N pixels centered at p, where I_x and I_y are the image gradients. Let λ_1 be the smallest eigenvalue of G:

$$\lambda_1 = \min(\text{eig}(G))$$

Then p is selected if λ_1 is greater than a predefined threshold λ_t and has a local maximum at p.

If we call P_λ the characteristic polynomial:

$$P_\lambda = (a - \lambda)(c - \lambda) - b^2 \tag{2}$$

it can be shown [9] that the condition $\lambda_1 > \lambda_t$ is equivalent to:

$$P_{\lambda_t} > 0 \quad \text{and} \quad a > \lambda_t \tag{3}$$

With this approach instead of finding the eigenvalues of a 2×2 matrix it is sufficient to evaluate the polinomial P_{λ_t}, which needs only two multiplications.

The nonmaximum suppression is then performed on P_{λ_t}, which is shown [9] to have very strong similarity with the original quality measure λ_1.

3 Feature tracking

In our study of the point association and tracking problem we explored several approaches. We tested the straightforward solutions of matching selected points in adjacent frames first by nearest neighbor and then by Normalized Cross-Correlation of the surrounding patches, with very little success. The main reason for the failure of these attempts is that the output of the feature selector does not exhibit a stable behavior, points disappear and reappear from one frame to the next, and attempts of directly matching points in adjacent frames will inevitably produce poor results. Moreover, the computational load for complete NCC tests is too high for a real-time application, in particular given our goal of tracking hundreds of features at frame rates not lower than 60Hz.

We propose to enforce temporal continuity by assuming linear dynamics for the tracked points in the image sequence. If we assume that selected features are measurements of underlying tracks whose dynamics is described by a Gaussian ARMA model, then we have a standard problem of tracking and data association in a multitarget environment with less-than-unity probability of detection and presence of false alarms. This problem has been studied in the literature for decades and a number of effective approaches have been developed (see [1] for a review).

What makes our scenario different from the one considered in the target tracking literature is the presence of images, which can be used as a rich source of information for discrimination and false alarm rejection in the data association step. We explored extensions to established tracking techniques by exploiting measures of similarity between feature patches.

3.1 Tracking and Data Association

Among the various tracking algorithms proposed in the literature, the main ones are Joint Probabilistic Data Association (JPDA [1]), Global Nearest Neighbor (GNN [4]), and variants (Probabilistic Data Association [2], Suboptimal Nearest Neighbor [5], Cheap JPDA [6], Fast JPDA [8] and Suboptimal JPDA [7]) . In all these algorithms Kalman Filters are used for tracking, and the predictions from the filters are used to define the regions where the features are expected to appear in the next frame - the so called validation gates. In particular, if $\hat{y}^i$ and S^i are respectively the predicted measurement and the variance of the innovation of the Kalman filter associated to track i, then the validation gate of track i is the set of points y such that the normalized distance:

$$d^i(y) = \sqrt{(y - \hat{y}^i)^T (S^i)^{-1} (y - \hat{y}^i)} \tag{4}$$

is less than a predefined value g (typically $g = 4$).

The JPDA algorithm is an extension to multiple targets of the PDA algorithm. The PDA is a single track-multiple measurements association algorithm. It uses a weighted average of the measurements in the validation gate to update a track, where the weights are the posterior probabilities that the measurements have been generated by the track assuming less than one probability of detection and Poisson distributed false measurements. The JPDA is a multiple track-multiple measurements association scheme, where the probabilities of all the possible associations measurements-tracks are evaluated in order to compute the weights. This is done by forming clusters consisting of all tracks that have measurements in the intersection of their validation gates. For each cluster all the possible associations measurement-track are considered and the corresponding probabilities are computed. Since the number of association hypotheses increases exponentially in the number of tracks and points in a cluster, and in test images with 1000 selections we obtained clusters exceeding 100 tracks, a direct JPDA implementation is not possible for a real-time application. Even though we can use one of the several suboptimal solutions proposed in the literature (CJPDA, SJPDA, FJPDA), there is another main issue. JPDA shows undesirable characteristics when used in a dense target environment such as the one we consider. In particular track bias and track merging may occur when several closely spaced targets travel in the same direction. To solve this problem, in [6] a Nearest Neighbor implementation of JPDA (NNJPDA) has been suggested. In this variant, instead of using a weighted average of the observations in the validation gate to update a track, the observation with highest association probability is used.

In Global Nearest Neighbor, a track can be updated by at most one measurement and a measurement can be assigned to at most one track. The first step is to form an assignment matrix $A = [d^i(y^j)]$ where $d^i(y^j)$ is the normalized distance (4) between track i and measurement y^j. Then the solution to the track-measurement association problem is obtained by maximizing the number of assignments while minimizing the sum of the the normalized distances of the assigned track-measurement pairs. The Munkres algorithm can be used to find the optimal assignment, but its complexity is $O(n^3)$, where n is the bigger between the number of measurements and the number tracks. Since we want to be able to track hundreds of points in real time, we consider a suboptimal solution, the Suboptimal Nearest Neighbor. This is a greedy approach to the matching problem: it searches the measurement-to-track pair with the minimum normalized distance, makes the assignment, removes all the pairs contaning the assigned measurement or track, then repeats the first step until no more assigment is possible. The complexity of this algorithm is $O(n \log(n))$ due to the ordering step. A further decrease in complexity can be achieved by first segmenting the input data in clusters as in the JPDA algorithm, then applying the SNN matching within each cluster.

A comparison of the performance of the discussed tracking algorithms was presented in [11]. The tests were conducted on radar data of real closely spaced maneuvering targets, and the results favor the Nearest Neighbor approaches. Consequently, given also the strict computational constraints due to the goal of a real-time application, we opted for the Suboptimal Nearest Neighbor approach.

3.2 Tracking feature patches

A natural extension of conventional target tracking algorithms for the problem of feature tracking in images is to modify the notion of normalized distance by including validation based on the appearance of the tracked patch. Our solution consists in comparing the patch surrounding the candidate selected point at the current frame with the patch at the estimated track position in the previous frame. Then the normalized distance (4) of measurement y^j from track i at time t becomes:

$$d_t^i(y^j) = \begin{cases} \sqrt{(y^j - \hat{y}^i_{t|t-1})^T (S^i_t)^{-1} (y^j - \hat{y}^i_{t|t-1})} & \text{if } \mathrm{NCC}(I_t(y^j), I_{t-1}(\hat{y}^i_{t-1|t-1})) < h \\ \infty & \text{otherwise} \end{cases} \tag{5}$$

where h is a threshold, $\mathrm{NCC}(I_1(y_1), I_2(y_2))$ denotes the Normalized Cross-Correlation between the patch centered in y_1 on image I_1 and the patch centered in y_2 on image I_2, $y^i_{t|t-1}$, $y^i_{t-1|t-1}$ and S^i are the prediction, filtered estimate and innovation variance of track i computed at time $t-1$.

The effect of this modification is to reject from the candidate points all the selections whose appearance does not match the one of the track in the previous frame. This test allows to prune out wrong potential matches and is simple enough to allow for a real-time implementation.

4 Track initiation

In a scenario where track-data association is a main issue special care must be taken in the initiation of new tracks. This problem is well know and several algorithms have been proposed for its solution [1]. In view of some comparison studies on track initiator performances [12], we concluded that the logic-based method [1] was the most suitable for our application. The approach works as follows:

- Starting with a measurement in the first frame, a validation gate is set up for the second frame. For every measurement in the second frame falling in the validation region a potential track is set up.

- For each potential track the velocity of the track is computed from the first two frames and used to set a validation gate in the third frame. The measurement in the third frame closest to the prediction is used to extend the track.
- From the last three frames, track velocity and acceleration is computed and used to set a validation gate in the next frame. If no measurement falls within the gate, the track is terminated. Otherwise this step is repeated until the desired minimum number of measurements is reached.

Usually three or four is the number of measurements required for track initiation. Note that in the first two scans measurements can be used more than once to form potential tracks. When a track is successfully initiated, the initiator has a mechanism to suppress duplicate tracks and prevent multiple use of the same points.

By replacing the normalized distance used in the validation gate tests with the NCC-based distance (5), it was possible to naturally extend this track initiator scheme to exploit information on the appearance of the patch.

5 Implementation and experiments

For image acquisition, we are using a Teli CS8550DiF DCAM-compliant CCD camera, which has a resolution of 640x480 monochrome pixels and a refresh of 60 full (non-interlaced) frames per second.

5.1 Feature selection

The current version of the Harris selector is implemented on a Xilinx WILDSTAR/PCI FPGA board. The board has the folllowing features:

- Xilinx VIRTEX FPGA with 1 million system gates
- 1.6 Gbytes/second I/O Bandwidth
- 6.4 Gbytes/second Memory Bandwidth
- Processing clock 100 MHz
- Synchronous ZBT SRAM with 100 MHz access

Selection takes about 160 microseconds on average on 640×480 images, which is approximately constant with the number of features, since the algorithm runs on the entire image every frame. The board is connected to a host computer via the PCI bus. Because of limitations on the hardware, it takes about 20 milliseconds to move data from the camera to the memory on the FPGA board. This accidental hardware limitation does not allow us to display the complete end-to-end system running in real time. Therefore, for experimental purpose, we use a stored sequence of images as input to our algorithms and

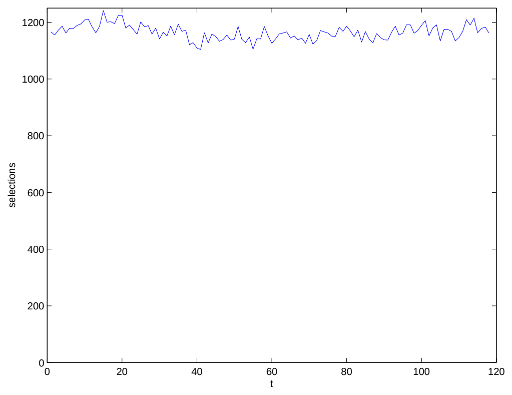

Figure 2: Number of feature selected in a outdoor sequence

measure the time required for each component. An alternative FPGA board, the Micro-Line C6713Compact, has on-board Firewire (along with drivers for DCAM-compliant cameras) which writes directly into memory accessible by the FPGA. This will eliminate the memory-transfer delay and allow the system to run in real-time. Due to its high cost, however, its purchase has been ruled out since the feasibility of the system can be equally well illustrated by careful timing of each component.

As described in section 2, a threshold parameter λ_t is used to control the number of selections per frame. All the local maxima of the polynomial P_{λ_t} satisfying conditions (3) are selected.

The image gradients I_x and I_y are obtained by computing the centered differences, which amounts to convolving the image with the kernels $\begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix}$ and $\begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix}^T$. This solution is computationally efficient but it is known to be rather sensitive to the noise. Significant improvements in accuracy and stability of results can be obtained by using smoothing kernels in the calculation on the image gradients. We plan to investigate this possibility in future developments.

In figure 2 we show the number of selections in a sample outdoor sequence, where we can see some the fluctuations of the total number of features. Besides that, it must be pointed out that there is significant additional interframe variation in the selections due to disappearing and reappearing of features. The performance of the tracker are very sensitive to the quality of the selections, but not particularly to their number. In figure 3 we show average and standard deviation of the lifetime of tracks for the previous sequence as function of average number of selected features. The flatness of the curve shows how the choice of the selection threshold is not critical for the overall lifetime of the tracks.

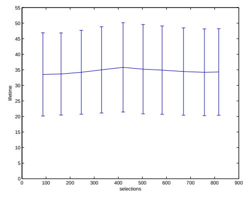

Figure 3: Tracks lifetime vs. number of selections for a 100 frame sequence: mean (solid line) and standard deviation (vertical bars).

5.2 Data association and tracking

The data association and tracking algorithm is composed of 3 modules: track initiator, filter/predictor, and data association.

5.2.1 Track initiation

Modified logic-based track initiation scheme as described in section (4), with the following parameters: frames required for successful track initiation (default 4), maximum number of track splits in the second frame (default 3), and maximum number of initiated tracks per frame.

The track initiation is computationally the most expensive part of the algorithm. If larger number of tracks or bigger frame rates are desired, big speed gains can be achieved by removing this module and initiating a track for each new selection.

5.2.2 Kalman prediction and filtering

We model the position $x(t)$ of the features on the image as second order Brownian motion:

$$\begin{aligned} \begin{bmatrix} x(t+1) \\ v(t+1) \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} + w(t) \ , \begin{bmatrix} x(0) \\ v(0) \end{bmatrix} = \begin{bmatrix} x_0 \\ 0 \end{bmatrix} \ , \ w(t) \sim \mathbf{N}(0, Q)) \\ y(t) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} + v(t) \quad , \quad v(t) \sim \mathbf{N}(0, R) \end{aligned} \tag{6}$$

with noise variances:

$$Q = \begin{bmatrix} q_x & 0 & 0 & 0 \\ 0 & q_x & 0 & 0 \\ 0 & 0 & q_v & 0 \\ 0 & 0 & 0 & q_v \end{bmatrix} \quad , \quad R = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}$$

x_0 is the position in the first frame, q_x, q_v, r are parameters to be tuned according to the dynamics of the scene: q_x set to a small number or zero, q_v related to the velocity of the features, and r based on the confidence in the measurements (because of discretization not smaller than 0.5). A linear time invariant Kalman filter is derived from the motion model (6). The initial variance P_0 of the filter estimate is obtained by first solving the Riccati equation for the filter, then applying n_0 prediction steps to the result. n_0 measures the number of measurements needed to reach the steady state and is a parameter of the algorithm. In this way we guarantee a monotonically decreasing error variance S, therefore a decreasing validation gate area.

5.2.3 Data association

For track-measurement association we implemented the Suboptimal Nearest Neighbor algorithm with modified normalized distance as described in 3.1 . A number of optimizations have been applied to the orginal algorithm in order to allow for real-time performance. First a JPDA-like clustering step is performed to reduce the number of the association hypotheses. Then tracks and measurements are organized in a structure with spatial information that allows to prune out pairings exceeding the maximum distance defined by the validation gate of the track. Finally, an optimized implementation of Normalized Cross Correlation has been included. Parameters of the algorithm are the size of the patch used for the NCC test and the threshold h used for calculating the normalized distance (5).

When no measurements can be associated to a track, the measurement update is step is not performed and an additional prediction step of the Kalman filter is applied.

A track is terminated when either a maximum number of consecutive predictions steps is attained, or the norm of the innovation exceeds a defined threshold, both parameters of the algorithm.

Figure 4 shows the trajectories of the filtered estimate for a small number of tracks in a video clip. It is clear from the zoomed plot how the filter successfully interpolates the noisy measurements. In figure 6 we see the innovation of the Kalman filters associated to these tracks, which appears to be close to zero mean. Figure 5 depicts the predicted positions together with the validation gates. We can see how the gates become smaller as new measurements are associated to the track.

To illustrate the performance of our tracker, we show in Table 1 the results of tracking a sequence of 100 frames with different selection algorithms and thresholds (see figures 7 and 8 for sample frames). The Harris feature selector

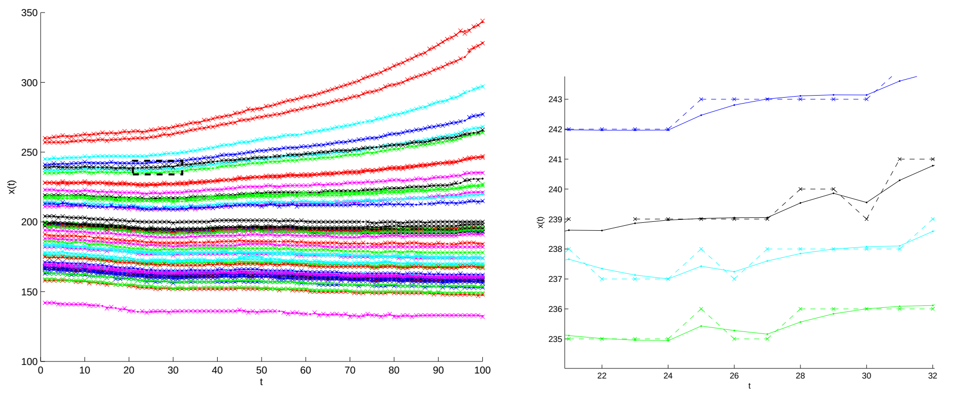

Figure 4: Kalman filter position estimates for 50 tracks in a 100 frames clip, x coordinate plotted. Estimates shown as dots connected with solid lines, measurement as crosses connected by dashed lines. The left plot displays all the tracks, the right plot is a zoom of the small dashed box on the left. It can be seen that most validation gates contain multiple measurements, which can be correctly associated thanks to the image-based validation and the temporal dynamics model encoded in the tracking filter.

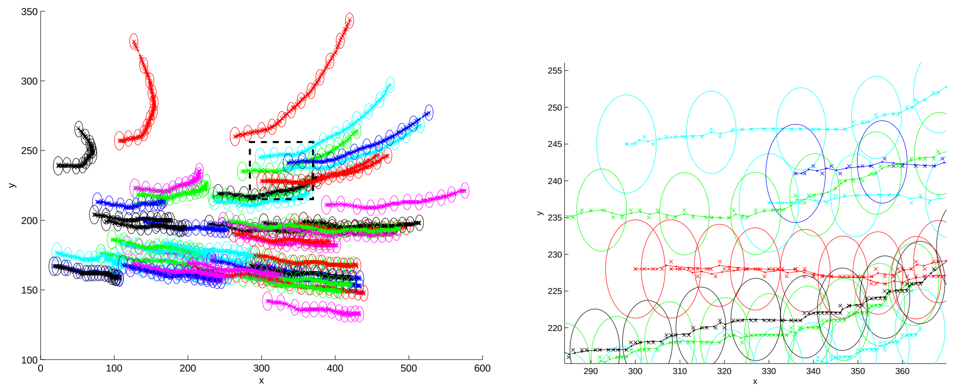

Figure 5: Kalman filter predictions and validation gates for the same sequence as in figure 4. Estimates shown as dots, measurement as crosses, validation gates as circles. The left plot displays all the tracks, the right plot is a zoom of the small dashed box on the left.

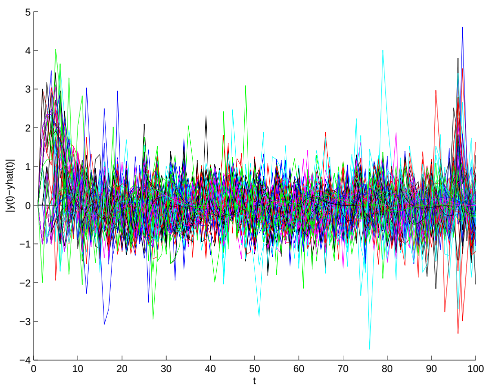

Figure 6: Kalman filter innovation for the sequence in figure 4, which appears close to zero mean.

Feature Selection Algorithm	Average Number of Selections	Average Number of Tracks	Average tracking time per frame (ms)	Frame rate (Hz)
Harris	1000	510	11.21	89
FPGA Tomasi-Kanade	1000	411	9.81	102
Harris	500	318	5.40	185
FPGA Tomasi-Kanade	500	250	4.74	211
Harris	200	128	2.54	393
FPGA Tomasi-Kanade	200	114	2.52	397

Table 1: Results and timing of the tracker with different selection algorithms

used in these experiments is the implementation available in the Intel OpenCV library [3]. It is clear from the data that the Harris corner detection significantly improves the number and quality of the tracks.

The timings in Table 1 were measured on a Pentium IV 3.2 GHz PC with 1 Gbyte of RAM and do not include selection time.

A Feature selection code

Below we include the VHDL source code of our FPGA feature selector.

```
FILE: pex_fs.vhd

------------------------------------------------------------------------
--
--  Copyright (C) 1998-2000, Annapolis Micro Systems, Inc.
--  All Rights Reserved.
--
------------------------------------------------------------------------

------------------------------------------------------------------------
--
--  Entity        : PEX
--
--  Architecture  : Template
--
--  Filename      : pex_template_arch.vhd
```

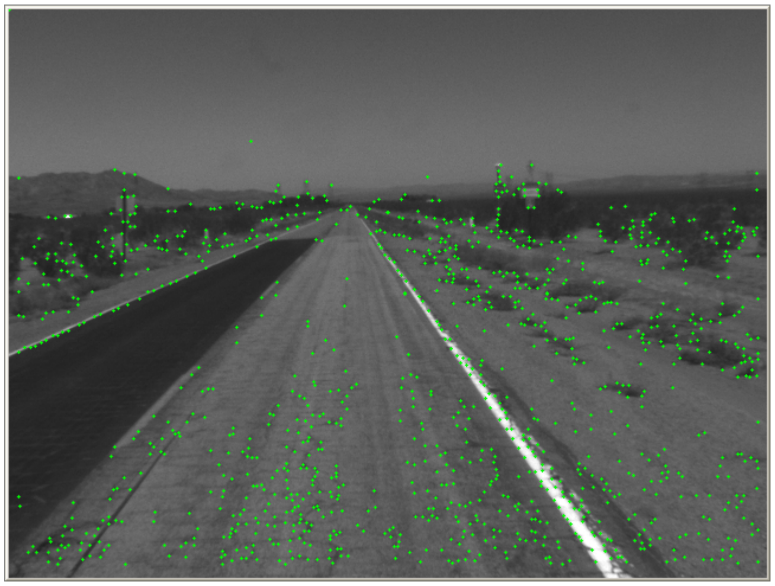

Figure 7: FPGA selection output, sample frame. There are 1162 feature selected, obtained with threshold parameter $\lambda_t = 50$. The sequence is available at `http://www.cs.ucla.edu/~bissacco/jpltrack/fpgaselect.mov`

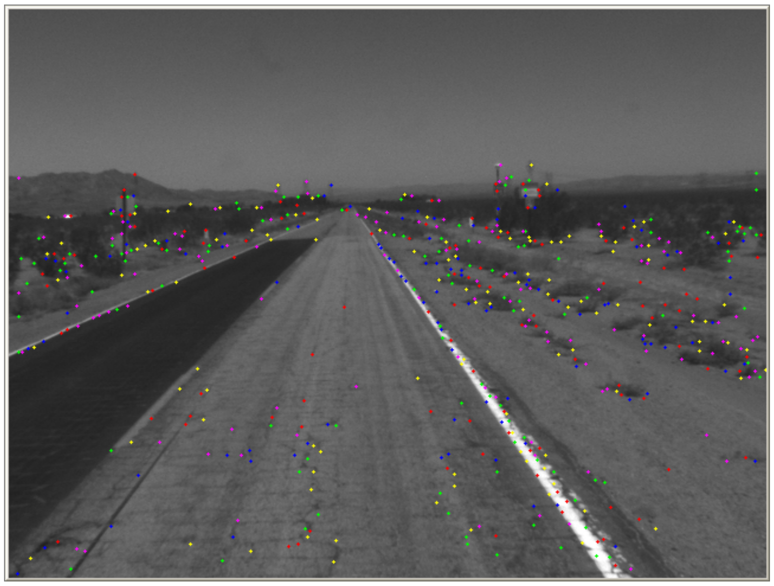

Figure 8: Tracking output, sample frame, obtained with selections in figure 7 as input. Tracks are displayed as dots of different colors, their total number in this frame is 562. The sequence is available at `http://www.cs.ucla.edu/~bissacco/jpltrack/fpgatrack.mov`, together with the results of tracking using the Harris selector `http://www.cs.ucla.edu/~bissacco/jpltrack/harristrack.mov`

```
--
--  Date          : 7/25/00
--
--  Description   : PEX architecture that should be used as a starting
--                  point for new PEX applications.  This file should
--                  be copied to the project directory, then customized
--                  to meet the needs of the application.
--
------------------------------------------------------------------------

------------------------- Library Declarations -----------------------

-- IEEE Libraries --
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use ieee.std_logic_arith.all;

-- Wildstar System Libraries -
library SYSTEM;
use SYSTEM.Xilinx_Package.all;
use SYSTEM.AMS_package.all;

-- Wildstar PEx Libraries --
library PEX_Lib;
use PEX_Lib.PE_Package.all;
--use PEX_Lib.PE_Mezz_Mem_package.all;
--use PEX_Lib.Mezz_Mem_Mux_pkg.all;
--use PEX_Lib.Mezz_Mem64_Mux_pkg.all;
use PEX_Lib.PEX_Mem32_Mux_pkg.all;
use PEX_Lib.PE_LAD_Mux_pkg.all;
use PEX_Lib.LAD_Mem32_Mux_pkg.all;
--use PEX_Lib.LAD_Mem64_Mux_pkg.all;

-- LAD Mux Libraries --
library LAD_Mux_Lib;
use LAD_Mux_Lib.LAD_Mux_pkg.all;

-- Mem Mux Libraries --
library Mem32_Mux_Lib;--, Mem64_Mux_Lib;
use Mem32_Mux_Lib.Mem32_Mux_pkg.all;
--use Mem64_Mux_Lib.Mem64_Mux_pkg.all;

  ----------------------------------------------------------------------
  --
  --  Below are all of the standard PE pad interface signals. Simply
  --  uncomment the signal(s) that are needed by the PE design.  All
  --  other unused signals may remain commented out.  Be sure to
  --  uncomment any component instances used by the interface.
  --
  ----------------------------------------------------------------------
----------------------- Architecture Declaration ---------------------

architecture Mux_Mem_Test of PEX is

  constant img_width  : integer := 640;
  constant img_height : integer := 480;

  signal GND            : std_logic := '0';
  signal Global_Reset   : std_logic := '0';
  signal Reset_Register : std_logic := '0';
  signal Clocks_In      : Clock_Std_IF_In_Type;
  signal Clocks_Out     : Clock_Std_IF_Out_Type;
--  signal LAD_Bus_In     : LAD_Bus_Std_IF_In_Type;
--  signal LAD_Bus_Out    : LAD_Bus_Std_IF_Out_Type;
  signal LEDs_Out     : LED_Std_IF_Out_Type;
--  signal Left_Mem_In    : Mem_Std_IF_In_Type;
--  signal Left_Mem_Out   : Mem_Std_IF_Out_Type;
--  signal Right_Mem_In   : Mem_Std_IF_In_Type;
--  signal Right_Mem_Out  : Mem_Std_IF_Out_Type;
--  signal Left_Mezz_In   : Mezz_Mem_Std_IF_In_Type;
--  signal Left_Mezz_Out  : Mezz_Mem_Std_IF_Out_Type;
--  signal Right_Mezz_In  : Mezz_Mem_Std_IF_In_Type;
--  signal Right_Mezz_Out : Mezz_Mem_Std_IF_Out_Type;
--  signal PE0_Bus_In     : PE0_Bus_Std_IF_In_Type;
--  signal PE0_Bus_Out    : PE0_Bus_Std_IF_Out_Type;
--  signal Top_Sys_In     : Systolic_Std_IF_In_Type;
--  signal Top_Sys_Out    : Systolic_Std_IF_Out_Type;
--  signal Bot_Sys_In     : Systolic_Std_IF_In_Type;
--  signal Bot_Sys_Out    : Systolic_Std_IF_Out_Type;

  ----------------------------------------------------------------------
  --
  --  Below are signals which can be used with the Mem_Mux and
  --  LAD_Mux interfaces.  Modify each signal to match the number of
  --  clients for each mux interface.
  --
  ----------------------------------------------------------------------
  signal count              : std_logic_vector (31 downto 0);
  signal reg_in             : std_logic_vector (31 downto 0);
  signal reg_out            : std_logic_vector (31 downto 0);
  signal pixin              : std_logic_vector (7 downto 0);
  signal pixout             : std_logic_vector (7 downto 0);
  signal lambda             : std_logic_vector (16 downto 0);
  signal flag         : std_logic;
  signal feature_quality : std_logic_vector (35 downto 0);
  signal bit_out            : std_logic;
  signal LAD_Mux_Bus        : LAD_Mux_vector   ( 0 to 3 );
  signal Left_Local_Mux     : Mem32_Mux_vector ( 0 to 1 );
  signal Right_Local_Mux    : Mem32_Mux_vector ( 0 to 1 );
  signal read_clk           : std_logic;
  signal K_Clk              : std_logic;
  signal i                  : integer;
  signal My_Registers      : LAD_Mux_register_vector(0 to 3);
  type state_value is (state_0, state_1, state_2, state_3, state_4, state_5, state_6 );
  signal state : state_value;

  component eigen is
     generic(img_width : positive := img_width);
     port (pixin : in signed (7 downto 0);
 clk : in std_logic;
     lambda : in signed (16 downto 0);
     flag : out std_logic;
 quality : out signed (31 downto 0);
   pixout : out signed (7 downto 0));
   end component;

 for all : eigen use entity pex_lib.eigen(synthesis);

begin

  K_Clk <= Clocks_In.K_Clk;

    u_Regfile: LAD_Mux_RegFile
    generic map
    (
      Mask            => x"7FF8",
      Base            => x"0000",
      L2Num           => 2
    )
    port map
    (
      Kclk            => K_Clk,
      LAD             => LAD_Mux_Bus(3),
      Regs            => My_Registers
    );

  read_mem : process ( Global_Reset, Clocks_In.M_Clk , Left_Local_Mux(1), Right_Local_mux(1) )

  begin
    if (Global_Reset = '1') then
      i                             <= 0;
      read_clk                      <= '0';
--    reg_out                       <= (others => '0');
      reg_in                        <= (others => '0');
      Left_Local_Mux(1).Addr        <= (others => '0');
      Right_Local_Mux(1).Addr       <= (others => '0');
      Left_Local_Mux(1).Data_Out    <= (others => '0');
      Right_Local_Mux(1).Data_Out   <= (others => '0');
      Left_Local_Mux(1).Req         <= '0';
      Right_Local_Mux(1).Req        <= '0';
      Left_Local_Mux(1).Write       <= '0';
      Right_Local_Mux(1).Write      <= '1';
      state <= state_0;
      LEDs_Out.Red_n                <= '1';
      LEDs_Out.Green_n              <= '1';
    My_Registers(2).Data_out  <= (others => '0');

    elsif (rising_edge (Clocks_In.M_Clk) ) then

    case state is

       when state_0 =>
            read_clk <= '0';
lambda <= My_Registers(0).Data_in(16 downto 0);
            if ( i > (img_width * img_height)) then
                state <= state_6;
            elsif (lambda > 1) then
                state <= state_1;
else
state <= state_0; -- stay in state_0 until lambda is non zero
            end if;

       when state_1 =>
--            read_clk <= '0';
    Left_Local_Mux(1).Write    <= '0';
            Left_Local_Mux(1).Req      <= '1';
            if (Left_Local_Mux(1).Akk = '1' ) then
              state <= state_2;
            else
              state <= state_1;
            end if;
       when state_2 =>
--              read_clk <= '0';
             if (Left_Local_Mux(1).Data_Valid = '1' ) then
               reg_in <= Left_Local_Mux(1).Data_In;
               Left_Local_Mux(1).Req <= '0';
               state <= state_3;
             else
--               Left_Local_Mux(1).Req <= '1';
               state <= state_2;
             end if;

      when state_3 =>
        read_clk <= '1';
        Left_Local_Mux(1).Write <= '0';
        Left_Local_Mux(1).Req <= '0';
        if (Left_Local_Mux(1).Data_Valid = '0') then
                Left_Local_Mux(1).Addr <= Left_Local_Mux(1).Addr + '1';
     i <= i + 1;
        state <= state_4;
        else
            state <= state_3;
        end if;

      when state_4 =>
--read_clk <= '1';
Right_Local_Mux(1).Data_Out <= reg_out;
            Right_Local_Mux(1).Write <= '1';
            Right_Local_Mux(1).Req <= '1';
            if (Right_Local_Mux(1).Akk = '1') then
              state <= state_5;
            else
                  state <= state_4;
            end if;

      when state_5 =>
--                read_clk <= '1';
                Right_Local_Mux(1).Req <= '0';
            if (Right_Local_Mux(1).Akk = '1') then
         state <= state_5;
            else
                Right_Local_Mux(1).Addr <= Right_Local_Mux(1).Addr + '1';
                 state <= state_0;
            end if;

      when state_6 =>
        LEDs_Out.Red_n <= '0';
My_Registers(2).Data_out <= x"0000FFFF";
         state <= state_6;

    end case;
   end if;
 end process read_mem;

  My_Registers(0).Data_out <= My_Registers(0).Data_in;
  My_Registers(1).Data_out <= "000000000000000" & lambda;


  my_eigen : eigen
     generic map
 ( img_width => img_width )
     port map
 (pixin     => signed(pixin) ,
  clk       => read_clk,
      lambda    => signed (lambda),
  flag      => flag,
  quality => signed(feature_quality),
      pixout    => signed(pixout));

  pixin <= reg_in(7 downto 0);
  reg_out <=  flag & feature_quality(35 downto 5); --throwing out the 5 least significant bits to fit to the 32-bit bus

 compute : process (Global_Reset)
begin
if (Global_Reset = '1' ) then
count  <= (others => '0');
   end if;

--  if (flag='1') then
--       reg_out <= x"000000ff";
--else
--   reg_out <= x"00000000";
--  end if;


-- compute : process (Global_Reset, read_clk)
--  begin
--  if (Global_Reset = '1' ) then
--       count  <= (others => '0');
--       reg_out <= (others => '0');
--  else
--    pixin <= reg_in(7 downto 0);
--reg_out <=  flag & flag & flag & flag & flag & flag & flag & flag; -- write FF if a feature, 0 otherwise;
--  if (flag='1') then
--       reg_out <= x"000000ff";
--else
--   reg_out <= x"00000000";
--  end if;
--  end if;

    --reg_out(31 downto 8)  <= (others => '0');
--  reg_out(7 downto 0) <= pixout;
--    reg_out <= x"00000000";
--    if (flag='1') then
  --     reg_out <= x"000000ff";
  --   reg_out(7 downto 0) <= x"ff";
    -- pixin <= x"0f";
   --else
 --reg_out(7 downto 0) <=pixout;
 -- pixin <= x"ff";
    --   reg_out <= My_registers(0).Data_In;
    --end if;
--  end if;
 end process compute;



  -- The following bridge component provides LAD-based access to PEX's local memories.

  U_LEFT_LOCAL_BRIDGE : LAD_Mem32_Bridge
    generic map
    (
      Mask            => x"7E00",
      Base            => x"1000"
    )
    port map
    (
      Kclk            => K_Clk,
      Mclk            => Clocks_In.M_Clk,
      LAD             => LAD_Mux_Bus(0),
      Mem             => Left_Local_Mux(0)
    );

  U_RIGHT_LOCAL_BRIDGE : LAD_Mem32_Bridge
    generic map
    (
      Mask            => x"7E00",
      Base            => x"1200"
    )
    port map
    (
      Kclk            => K_Clk,
      Mclk            => Clocks_In.M_Clk,
      LAD             => LAD_Mux_Bus(1),
      Mem             => Right_Local_Mux(0)
    );


  ----------------------------------------------------------------------
  --
  --  Below are all of the standard PE pad interface components. Simply
  --  uncomment the interface(s) that are needed by the PE design.  All
  --  other unused interfaces may remain commented out.  Be sure to
  --  uncomment any signal declarations used by the interface.
  --
  ----------------------------------------------------------------------

  U_Clocks : Clock_Std_IF
    generic map
    (
      K_CLK_DLL_OUT   => USE_1x,
      M_CLK_DLL_TYPE  => LOW_FREQ,
      P_CLK_DLL_TYPE  => LOW_FREQ,
      U_CLK_DLL_TYPE  => LOW_FREQ
    )
    port map
    (
      Pads          => Pads.Clocks,
      User_In       => Clocks_In,
      User_Out      => Clocks_Out
    );

--  --@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
--  --@@
--  --@@  NOTE: Use either the U_LAD_MUX_IF or the U_LAD_Bus,
--  --@@        but not simultaneously.
--  --@@
--  --@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

  U_LAD_MUX_IF : LAD_Mux_IF
    generic map
    (
      K_Clk_x2      => FALSE
    )
    port map
    (
      Kclk          => K_Clk,
      Reset         => Global_Reset,
      pads          => Pads.LAD_Bus,
      Clients       => LAD_Mux_Bus
    );

--  U_LAD_Bus : LAD_Bus_Std_IF
--    port map
--    (
--      K_Clk         => Clocks_In.K_Clk,
--      Global_Reset  => Global_Reset,
--      Pads          => Pads.LAD_Bus,
--      User_In       => LAD_Bus_In,
--      User_Out      => LAD_Bus_Out
--    );
--
--  --@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
--  --@@
--  --@@  NOTE: Use either the U_Left_Local_Mem_Mux or the
--  --@@        U_Left_Mem, but not simultaneously.
--  --@@
--  --@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

  U_Left_Local_Mem_Mux : Mem32_Mux_Priority_IF
    port map
    (
      Mclk          => Clocks_In.M_Clk,
      Reset         => Global_Reset,
      Pads          => Pads.Left_Mem,
      Clients       => Left_Local_Mux
    );

--  U_Left_Mem : Mem_Std_IF
--    generic map
--    (
--      INFF_Delay    => NODELAY,
--      OBUF_Drive    => SLOW_12mA
--    )
--    port map
--    (
--      M_Clk         => Clocks_In.M_Clk,
--      Global_Reset  => Global_Reset,
--      Pads          => Pads.Left_Mem,
--      User_In       => Left_Mem_In,
--      User_Out      => Left_Mem_Out
--    );
--
--  --@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
--  --@@
--  --@@  NOTE: Use either the U_Right_Local_Mem_Mux or the
--  --@@        U_Right_Mem, but not simultaneously.
--  --@@
--  --@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

  U_Right_Local_Mem_Mux : Mem32_Mux_Priority_IF
    port map
   (
      Mclk          => Clocks_In.M_Clk,
      Reset         => Global_Reset,
      Pads          => Pads.Right_Mem,
      Clients       => Right_Local_Mux
    );

--  U_Right_Mem : Mem_Std_IF
--    generic map
--    (
--      INFF_Delay    => NODELAY,
--      OBUF_Drive    => SLOW_12mA
--    )
--    port map
--    (
--      M_Clk         => Clocks_In.M_Clk,
--      Global_Reset  => Global_Reset,
--      Pads          => Pads.Right_Mem,
--      User_In       => Right_Mem_In,
--      User_Out      => Right_Mem_Out
--    );
--
  ----------------------------------------------------------------------
  --
  -- The following LAD_Mux component provides a global reset signal to
  -- the PE.  It is mapped to LAD address 0x7fff. To change it's
  -- mapping alter the constant passed to the Base generic.
  --
  ----------------------------------------------------------------------
  U_LAD_Mux_Reset : LAD_Mux_Reset
    generic map
    (
      Mask        => x"7F00",
      Base        => x"7F00"
    )
    port map
    (
      Rclk        => Clocks_In.M_Clk,
      Kclk        => K_Clk,
      LAD         => LAD_Mux_Bus(2),
      Reset       => Global_Reset,
      DLL_Reset_0 => Clocks_Out.M_Clk_DllRst,
      DLL_Reset_1 => Clocks_Out.P_Clk_DllRst,
      DLL_Reset_2 => Clocks_Out.K_Clk_DllRst,
      DLL_Reset_3 => Clocks_Out.U_Clk_DllRst
    );

  ---------------------------------------------------------------------


U_LEDs : LED_Std_IF
    port map
    (
      Pads          => Pads.LEDs,
      User_Out      => LEDs_Out
    );
  ----------------------------------------------------------------------
  --
  --  Global reset interface : Attach the Reset_Register signal to a
  --  register bit of a LAD bus accessible register.  This reset
  --  mechanism generates a one K_Clk cycle long pulse to the GSR line
  --  of the STARTUP block.  The STARTUP block is also synchronous to
  --  K_Clk.
  --
  ----------------------------------------------------------------------
--  U_Reset_Pulse_Gen : One_Shot
--    port map
--    (
--      Clk           => Clocks_In.K_Clk,
--      I             => Reset_Register,
--      O             => Global_Reset
--    );
--
--  U_Startup : STARTUP_VIRTEX_GSR
--    port map
--    (
--      GSR           => Global_Reset
--    );

  ----------------------------------------------------------------------
  --  NOTE :  The following line must remain in all designs
  --          to ensure that all of the PE pads are driven.
  ----------------------------------------------------------------------
  Init_PEX_Pads ( Pads );

end Mux_Mem_Test;



FILE: eigen.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity eigen is
  generic(img_width : positive := 320);   -- width of image
  port (pixin : in signed (7 downto 0);
clk : in std_logic;
lambda : in signed (16 downto 0);
flag : out std_logic;
quality : out signed(35 downto 0);
pixout : out signed (7 downto 0));
end eigen;

architecture synthesis of eigen is
  component diff
    generic (img_width : positive := 320);   -- width of image
    port (pixel : in signed (7 downto 0);
  clk : in std_logic;
  dx : out signed (7 downto 0);
  dy : out signed (7 downto 0);
  pixout : out signed (7 downto 0));
  end component;

  component mul
    generic (img_width : positive := 320);   -- width of image
    port (x : in signed (7 downto 0);
  y : in signed (7 downto 0);
  clk : in std_logic;
  w : out signed (17 downto 0));
  end component;
  component D_FF
generic(w : positive := 8);
port (d : in signed(w - 1 downto 0);
  clk : in std_logic;
  q : out signed(w - 1 downto 0));
  end component;

  component delay
    generic(w : positive := 8;   -- width of delay line
     n : positive := 1);  -- lenght of delay line
    port (din : in signed (w - 1 downto 0);
  clk : in std_logic;
  dout : out signed (w - 1 downto 0));
  end component;

--  signal ixi : std_logic_vector (8 downto 0);
  signal ix  : signed (7 downto 0);
  signal iy  : signed (7 downto 0);
  signal a   : signed (17 downto 0);
  signal c   : signed (17 downto 0);
  signal b   : signed (17 downto 0);
  signal b2  : signed (35 downto 0);
  signal a1  : signed (17 downto 0);
  signal c1  : signed (17 downto 0);
  signal ac  : signed (35 downto 0);
  signal p   : signed (35 downto 0);

  signal n0, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16, n17, n18, n19, n20, n21, n22, n23, n24, n25 : signed (36 downto 0);

begin
  gblock : diff
    generic map (img_width)
    port map (pixin, clk, ix, iy, pixout);

  ablock : mul
    generic map (img_width)
    port map (ix, ix, clk, a);

  bblock : mul
    generic map (img_width)
    port map (ix, iy, clk, b);

  cblock : mul
    generic map (img_width)
    port map (iy, iy, clk, c);

  s1 : a1 <= a - (lambda(16) & lambda);
  s2 : c1 <= c - (lambda(16) & lambda);
  s3 : ac <= a1 * c1;
  s4 : b2 <= b * b;
  s5 : p  <= ac - b2;

  n0(36) <= '0' when (p > 0 and a1 > 0) else '1';
  n0(35 downto 0) <= p;

  d0 : D_FF
   generic map (w => 37)
    port map (n0, clk, n1);
  d1 : D_FF
   generic map (w => 37)
    port map (n1, clk, n2);
  d2 : D_FF
    generic map (w => 37)
    port map (n2, clk, n3);
  d3 : D_FF
   generic map (w => 37)
    port map (n3, clk, n4);
  d4 : D_FF
    generic map (w => 37)
    port map (n4, clk, n5);

  d5 : delay
    generic map (w => 37, n => (img_width - 4))
    port map (n5, clk, n6);
  d6 : D_FF
    generic map (w => 37)
    port map (n6, clk, n7);
  d7 : D_FF
    generic map (w => 37)
    port map (n7, clk, n8);
  d8 : D_FF
    generic map (w => 37)
    port map (n8, clk, n9);
  d9 : D_FF
    generic map (w => 37)
    port map (n9, clk, n10);


  d10 : delay
    generic map (w => 37, n => (img_width - 4))
    port map (n10, clk, n11);
  d11 : D_FF
    generic map (w => 37)
    port map (n11, clk, n12);
  d12 : D_FF
    generic map (w => 37)
    port map (n12, clk, n13);
  d13 : D_FF
    generic map (w => 37)
    port map (n13, clk, n14);
  d14 : D_FF
    generic map (w => 37)
    port map (n14, clk, n15);


  d15 : delay
    generic map (w => 37, n => (img_width - 4))
    port map (n15, clk, n16);
  d16 : D_FF
    generic map (w => 37)
    port map (n16, clk, n17);
  d17 : D_FF
    generic map (w => 37)
    port map (n17, clk, n18);
  d18 : D_FF
    generic map (w => 37)
    port map (n18, clk, n19);
  d19 : D_FF
    generic map (w => 37)
    port map (n19, clk, n20);


  d20 : delay
    generic map (w => 37, n => (img_width - 4))
    port map (n20, clk, n21);
  d21 : D_FF
    generic map (w => 37)
    port map (n21, clk, n22);
  d22 : D_FF
    generic map (w => 37)
    port map (n22, clk, n23);
  d23 : D_FF
    generic map (w => 37)
    port map (n23, clk, n24);
  d24 : D_FF
    generic map (w => 37)
    port map (n24, clk, n25);


quality <= n13(35 downto 0);
flag <= '0' when  (n13(36) = '1') else --shows whether the pixel is a feature or not (if it is the local maximum in the 5x5 window around it)
'0' when ((n1(36) = '0') and n13 <= n1) else
'0' when ((n2(36) = '0') and n13 <= n2) else
'0' when ((n3(36) = '0') and n13 <= n3) else
'0' when ((n4(36) = '0') and n13 <= n4) else
'0' when ((n5(36) = '0') and n13 <= n5) else
'0' when ((n6(36) = '0') and n13 <= n6) else
'0' when ((n7(36) = '0') and n13 <= n7) else
'0' when ((n8(36) = '0') and n13 <= n8) else
'0' when ((n9(36) = '0') and n13 <= n9) else
'0' when ((n10(36) = '0') and n13 <= n10) else
'0' when ((n11(36) = '0') and n13 <= n11) else
'0' when ((n12(36) = '0') and n13 <= n12) else
'0' when ((n14(36) = '0') and n13 < n14) else
'0' when ((n15(36) = '0') and n13 < n15) else
'0' when ((n16(36) = '0') and n13 < n16) else
'0' when ((n17(36) = '0') and n13 < n17) else
'0' when ((n18(36) = '0') and n13 < n18) else
'0' when ((n19(36) = '0') and n13 < n19) else
'0' when ((n20(36) = '0') and n13 < n20) else
'0' when ((n21(36) = '0') and n13 < n21) else
'0' when ((n22(36) = '0') and n13 < n22) else
'0' when ((n23(36) = '0') and n13 < n23) else
'0' when ((n24(36) = '0') and n13 < n24) else
'0' when ((n25(36) = '0') and n13 < n25) else
'1';

end synthesis;


configuration ceigen of eigen is
  for synthesis
    -- default
  end for;
end ceigen;


FILE: delay.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity D_FF is
generic(w : positive := 8);
port (d : in signed(w - 1 downto 0);
  clk : in std_logic;
  q : out signed(w - 1 downto 0));
end D_FF;

architecture simple of D_FF is
begin
q <= d when (clk = '1' and clk'event) else unaffected;
end simple;

-------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity delay is
  generic(w : positive := 8;   -- width of delay line
  n : positive := 320);  -- lenght of delay line
  port (din : in signed (w - 1 downto 0);
clk : in std_logic;
dout : out signed (w - 1 downto 0));
end delay;

architecture synthesis of delay is

component D_FF
generic(w : positive := 8);
port (d : in signed (w - 1 downto 0);
clk : in std_logic;
q : out signed (w - 1 downto 0));
end component;

type bank is array (0 to n-1) of signed (w-1 downto 0);
signal wire : bank;

begin
  gen_many : for i in 1 to n generate
just_a_label_1: if i = 1 generate
dffx : D_FF generic map (w) port map (din, clk, wire(1));
end generate;

    just_a_label_2: if (i > 1 and i < n) generate
dffx : D_FF generic map (w) port map (wire(i-1), clk, wire(i));
end generate;

just_a_label_3: if i = n generate
dffx : D_FF generic map (w) port map (wire(n-1), clk, dout);
end generate;
  end generate;
end synthesis;

configuration cdelay of delay is
  for synthesis
    -- default
  end for;
end cdelay;


FILE: gradient.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity diff is
  generic(img_width : positive := 320);   -- width of image
  port (pixel : in signed (7 downto 0);
clk : in std_logic;
dx : out signed (7 downto 0);  -- gradient in x direction
dy : out signed (7 downto 0);  -- gradient in y direction
pixout : out signed (7 downto 0));
end diff;

architecture synthesis of diff is

  signal n1, n2, n3, n4, n5, n6, n7 : signed (7 downto 0);
  signal temp_dx, temp_dy : signed (7 downto 0);


  component D_FF
generic(w : positive := 8);
port (d : in signed(w - 1 downto 0);
  clk : in std_logic;
  q : out signed(w - 1 downto 0));
  end component;

  component delay
    generic(w : positive := 8;   -- width of delay line
     n : positive := 1);  -- lenght of delay line
    port (din : in signed (w - 1 downto 0);
  clk : in std_logic;
  dout : out signed (w - 1 downto 0));
  end component;

begin

  d0 : D_FF
    port map (pixel, clk, n1);
  d1 : D_FF
    port map (n1, clk, n2);
  d2 : delay
    generic map (n => (img_width - 2))
    port map (n2, clk, n3);
  d3 : D_FF
    port map (n3, clk, n4);
  d4 : D_FF
    port map (n4, clk, n5);
  d5 : delay
    generic map (n => (img_width - 2))
    port map (n5, clk, n6);
  d6 : D_FF
    port map (n6, clk, n7);

  temp_dx <= signed ("0" & n5(7 downto 1)) - signed ("0" & n3(7 downto 1)); --pixel values are unsigned. the minus implemented in ieee.std_logic_signed generates
  temp_dy <= signed ("0" & n7(7 downto 1)) - signed ("0" & n1(7 downto 1)); --signed results with unsigned operands

  dx_ff : D_FF
port map (temp_dx, clk, dx);
  dy_ff : D_FF
port map (temp_dy, clk, dy);

  pixout <= n7;

end synthesis;

configuration cdiff of diff is
  for synthesis
    -- default
  end for;
end cdiff;


FILE: mul.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

entity mul is
  generic(img_width : positive := 320);   -- width of image
  port (x : in signed (7 downto 0);
y : in signed (7 downto 0);
clk : in std_logic;
w : out signed (17 downto 0));
end mul;

architecture synthesis of mul is
  signal w_temp : signed (17 downto 0);

  signal z   : signed (15 downto 0);
  signal q1  : signed (15 downto 0);
  signal q2  : signed (15 downto 0);
  signal q3  : signed (15 downto 0);
  signal q4  : signed (15 downto 0);
  signal q5  : signed (15 downto 0);
  signal q6  : signed (15 downto 0);
  signal q7  : signed (15 downto 0);
  signal q8  : signed (15 downto 0);


  component D_FF
generic(w : positive := 8);
port (d : in signed(w - 1 downto 0);
  clk : in std_logic;
  q : out signed(w - 1 downto 0));
  end component;

  component delay
    generic(w : positive := 8;   -- width of delay line
    n : positive := 1);  -- lenght of delay line
    port (din : in signed (w - 1 downto 0);
  clk : in std_logic;
  dout : out signed (w - 1 downto 0));
  end component;

begin

  z <= x * y;

  d1 : D_FF
    generic map (w => 16)
    port map (z, clk, q1);

  d2 : D_FF
    generic map (w => 16)
    port map (q1, clk, q2);


  d3 : delay
    generic map (w => 16, n => img_width - 2)
    port map (q2, clk, q3);

  d4 : D_FF
    generic map (w => 16)
    port map (q3, clk, q4);

  d5 : D_FF
    generic map (w => 16)
    port map (q4, clk, q5);

  d6 : delay
    generic map (w => 16, n => img_width - 2)
    port map (q5, clk, q6);

  d7 : D_FF
    generic map (w => 16)
    port map (q6, clk, q7);

  d8 : D_FF
    generic map (w => 16)
    port map (q7, clk, q8);

    w_temp <= (z(15) & z(15) & z) + (q1(15) & q1(15) & q1) + (q2(15) & q2(15) & q2) + (q3(15) & q3(15) & q3) + (q4(15) & q4(15) & q4)
     + (q5(15) & q5(15) & q5) + (q6(15) & q6(15) & q6) + (q7(15) & q7(15) & q7) + (q8(15) & q8(15) & q8);

w_ff : D_FF
  generic map(w => 18) port map(w_temp, clk, w);

end synthesis;

configuration cmul of mul is
  for synthesis
    -- default
  end for;
end cmul;
```

References

[1] Y. Bar-Shalom and T. E. Fortmann. Tracking and Data Association. Academic-Press, Boston, 1988

[2] Y. Bar-Shalom and X. R. Li Mutitarget-multisensor tracking: principles and techniques. Storrs, CT: YBS, 1995

[3] Jean-Yves Bouguet Pyramidal Implementation of the Lucas Kanade Feature Tracker. Included in OpenCV documentation, 2000.

[4] S. S. Blackman Multiple target tracking with radar applications. Norwood MA: Artech House, 1986.

[5] A. Farina and F. A. Studer Radar data processing I - Introduction and Tracking. Research Studies Press, 1985.

[6] R. J. Fitzgerald Development of practical PDA logic for multitarget tracking by microprocessor. In *Multitarget-Multisensor Tracking: Advanced Applications*. Norwood MA: Artech House, 1990.

[7] A. Roecker and G. L. Phillis Suboptimal joint probabilistic data association. IEEE Transactions on Aerospace and Electronic Systems, 29, 2 (1993).

[8] B. Zhou and N. K. Bose Development of practical PDA logic for multitarget tracking by microprocessor. In *IEEE Trans. Aerosp. Electron. Syst.*, 29(2), 352-363, 1993.

[9] A. Benedetti and P. Perona Real-time 2-D Feature Detection on a Reconfigurable Computer. In *Proc. CVPR*, 1998

[10] C. Tomasi and T. Kanade Detection and Tracking of Point Features. Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, Apr. 1991.

[11] H. Leung, Z. Hu and M. Blanchette Evaluation of Multiple Radar Target Trackers in Stressful Environments In *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 2, april 1999.

[12] Z. Hu, H. Leung and M. Blanchette Statistical performance analysis of track initiation techniques. In *IEEE Transactions on Signal Processing*, 45, 2, 445-456, 1997.