Fingerprint Matching on Splash 2

Nalini K. Ratha, Anil K. Jain & Diane T. Rover
Department of Computer Science
and Department of Electrical Engineering
Michigan State University
East Lansing, MI 48824

March 1, 1994

Abstract

Fingerprints are routinely used as a tool for personal identification and authentication. The process of matching a pair of fingerprints is usually based on the number and locations of local features known as 'minutiae'. The matching algorithm needs to be able to handle noisy data, a large-scale database, and high query load. A special-purpose hardware accelerator is needed for matching in order to provide reasonable response time. In this paper, we have described a method of achieving near-ASIC performance for rolled fingerprint matching using Splash 2. The sequential and parallel algorithms for fingerprint matching are presented. We obtained a matching speed of the order of 10^5 *matches per second.*

1 Introduction and Background

Fingerprints have been used in personal identification applications for a long time. They have been accepted as a tool for identifying criminals universally. Manual methods of matching fingerprints vary depending on the problem at hand. We have two types of situations in the area of criminal identification. The two categories are characterized on the basis of information available for matching. In the case of rolled fingerprint matching, the suspect is cooperative and all of his/her fingerprints (called rolled fingerprints) are used for identification. In a sense, we are trying to establish the suspect's identity. In the second category, we have latent fingerprints lifted from a scene-of-crime, which are characterized by smudgy, unclear, and partial impressions. Obviously, matching of latent fingerprints is more difficult. For rolled fingerprints,

Henry classification scheme is used where as for the latent fingerprints, Batley's formula [1] is used. In both the cases, a human fingerprint expert carries out the detailed matching.

1.1 Automation in Fingerprint Matching

Usually the number of records with which a test (query) fingerprint image needs to be matched is very large ($\approx 10^6$). The matching task, being mostly repetitive, is ideally suited for automation. In the last three decades, substantial efforts have been made to automate fingerprint identification.

- Semi-automatic methods:
  The computer is used to match the Henry Formula of the fingerprints with minor variations in ridge counts.

- Automatic matching methods:
  An image processing system is used to automatically extract features from a digital image of the fingerprint. The features are then used for matching against a stored database of fingerprints represented in terms of the same features. A survey of available commercial automatic fingerprint identification systems (AFIS) is described in [9].

1.2 Features Used in Matching Fingerprints

A fingerprint is characterized by alternating ridges and valleys. On a closer analysis, it can been seen that the ridges do not always run continuously. They have various anomalies in terms of ridge bifurcations, ridge endings, ridge crossovers, and small ridges. Collectively, these are called as 'minutiae'. For the purpose of automatic fingerprint matching, the features used are simply the ridge endings and ridge bifurcations. A minutiae is characterized by its position (x-y coordinate) and the angle the dominant ridge makes with the x-axis at the point of interest. No effort is made in the feature extraction process to distinguish between the two categories (ridge endings and ridge bifurcations) as one could be easily confused with the other depending on the inking conditions.

A digital gray scale image of a fingerprint is shown in Figure 1. In Figure 2, the two types of ridge features used for fingerprint identification are shown.

In this paper, we are interested in matching rolled fingerprints against a database of rolled fingerprints.

1.3 Computational Requirements

As mentioned earlier, a typical database of fingerprints contains millions of records. The number of queries for identification is also very high. Generally,

Figure 1: A grayscale image of a fingerprint

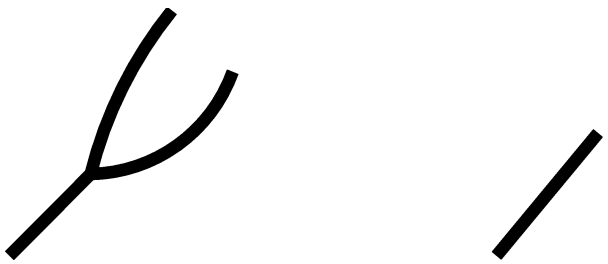


Figure 2: Features used for matching

the queries need to be answered within a short (say, 24-hour) time period. This poses a heavy computational load on the matching system. Even if a single match took, say, 1 millisecond of CPU time, matching against a database of 1 million fingerprints would require a total of 10^3 seconds of CPU time. If we have to process 100 queries per day, we would need 10^5 seconds or 27.78 hours of CPU time alone without I/O time involved in reading the database.

The computational load of matching is high primarily due to the following three factors: (i) a query fingerprint is usually of poor quality (more so for a latent fingerprint), (ii) database is very large; and (iii) structural distortion of the fingerprint images requires complex matching algorithms.

1.4 Special-Purpose Hardware

A fingerprint matching system can be implemented on any one of the following hardware platforms for high speed matching: (i) supercomputers, (ii) special-purpose hardware accelerators, and (iii) application specific integrated circuit (ASIC). However, these methods lack: (i) scalability, (ii) good price/performance, (iii) easy programmability, and (iv) chip level technology independence. We need a solution which can satisfy these constraints. The Field Programmable Gate Array (FPGA)-based solution meets most of the requirements, including: (i) chip level technology independence, (ii) close to ASIC performance at an affordable price, and (iii) easy programmability (through Hardware Desciption Languages). Using FPGAs, one can reprogram the available hardware and new algorithm can be synthesized on the same hardware. Instead of a fixed set of instructions as in the case of an array processor, FPGAs provide flexible "configurable logic blocks (CLBs)" which can be configured differently on each of the PEs in the array. FPGAs also provide high-speed custom computing at a much lower price than ASICs.

The paper is organized as follows. In section 2, we describe the sequential algorithm for matching a pair of rolled fingerprints. Splash 2 architecture and its features are described in section 3. The parallel version of the matching algorithm is presented in section 4. The speed of sequential method is presented and the expected speed for the parallel implementation is discussed in section 5. Conclusions and future work are mentioned in section 6.

2 Sequential Matching Algorithm

In this section, we describe a sequential algorithm for matching a pair of fingerprints. A query fingerprint is matched with every fingerprint in the database, discarding candidates with matching scores below a user-specified threshold. Rolled fingerprints usually contain a large number of minutiae (between 50 and 100). Since our interest here is in matching, we assume that the core point of the fingerprint is known and the fingerprints are oriented properly. The matching algorithm is summarized below.

1. Input : Query fingerprint feature vector, rolled fingerprint database.

2. Output : All the stored fingerprints and the corresponding matching scores whose matching score > T, where T is a user-specified threshold.

   Let the query fingerprint be represented as a N-dimensional feature vector ($\mathbf{f_1^q}$,$\mathbf{f_2^q}$,.....,$\mathbf{f_N^q}$). The query fingerprint core point is located at (C_x^q, C_y^q). Similarly the r^{th} reference (database) fingerprint is represented as an M-dimensional feature vector : ($\mathbf{f_1^r}$,$\mathbf{f_2^r}$,.....,$\mathbf{f_M^r}$). The reference fingerprint core point is located at (C_x^r, C_y^r).
   The i^{th} feature $\mathbf{f_i}$ is a vector with three components ($f_i(x), f_i(y), f_i(\theta)$).

   The components of a feature vector are shown geometrically in Figure 3.

3. (a) Normalization or Registration:

   For (i=1,M) do
   $f_i^r(x) = f_i^r(x) - C_x^q$
   $f_i^r(y) = f_i^r(y) - C_y^q$

   The fingerprints are assumed to be registered with a known orientation. Hence, there is no need of normalization for rotation.

   (b) Bounding Box:

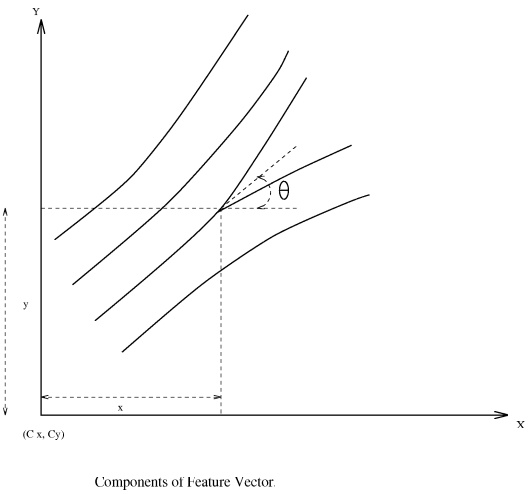


Figure 3: Components of features used for matching.

Compute the common bounding box for the query and reference fingerprints (i.e., eliminate the effects of features which do not contribute to matching).

- Let (x_q^t, y_q^t) and (x_q^b, y_q^b) define the bounding box for the query fingerprint, where x_q^t is the x-coordinate of the top left corner of the box and x_q^b is the x-coordinate of the bottom right corner of the box.
- Let (x_r^t, y_r^t) and (x_r^b, y_r^b) define the box for reference fingerprint.
- The intersection of these two boxes is the desired box.

  Count the minutiae features in this box for both query and reference fingerprints. Let's say query print has M_e^q and reference print has N_e^r features in this box.

(c) Tolerance Vector Computation:

Two features $\mathbf{f_i^r}$ and $\mathbf{f_j^q}$ are said to match if
$f_j^q(x) - t_i^r(x) \leq f_i^r(x) \leq f_j^q(x) + t_i^r(x)$, and
$f_j^q(y) - t_i^r(y) \leq f_i^r(y) \leq f_j^q(y) + t_i^r(y)$, and
$f_j^q - t_i^r(\theta) \leq f_i^r(\theta) \leq f_j^q(\theta) + t_i^r(\theta)$.

where $t_i^r = (t_i^r(x), t_i^r(y), t_i^r(\theta))$ is the tolerance vector.

Tolerance vector for each feature is computed as follows. If the distance from the core point to the present feature is less than K then
$t_i^r(x) = \text{ldcos}(\phi)$,

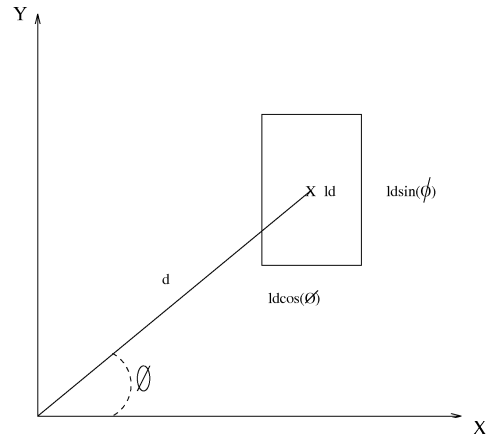


Figure 4: Tolerance box for X- and Y-components.

$t_i^r(y)$ = ldsin(ϕ), and
$t_i^r(\theta)$ = r.
ϕ= Angle of the line joining core and i^{th} makes with X-axis. where K, p, q are prespecified constants. Otherwise,
$t_i^r(x) = k_1$,
$t_i^r(y) = k_2, and$
$t_i^r(\theta) = k_3$
where k_1, k_2 and k_3 are prespecified constants. Tolerance box is shown geometrically in Figure 4.
(d) Compute matching score (MS (q,r)):

Set the number of paired features, $m_p = 0$;
For all query features $\mathbf{f_i^q}$, i=1,2, ... N_e^q do
If $\mathbf{f_i^q}$ matches with any feature in $\mathbf{f_j^r}$ then increment m_p.
Mark the corresponding feature in f_r as paired (to avoid future pairing by any other feature).

Matching score between the query fingerprint and the r^{th} reference fingerprint is defined as
MS(q,r) = $\frac{m_p^r * m_p^r}{(M_e^r * N_e^q)}$

4. Repeat the procedure for all fingerprints in the database. All the reference fingerprints with a score greater than T are output.

A similar matching algorithm is described in [8]. Depending on the desired accuracy, more than one finger could be used in matching. In

that case, a composite score needs to be computed for each set.

3 Splash 2 Architecture

The Splash 2 system is an attached processor developed by the Supercomputing Research Center (SRC). The Splash 2 is connected to a Sun SPARCstation 2 host. It is based on Xilinx 4010 FPGAs. A Splash 2 system could consist of up to 16 processing boards, each board with 16 FPGAs.

Figure 5 shows a system-level view of Splash 2 architecture. The host is connected to the Splash 2 through an interface board which extends the address and data buses from the Sun workstation into the address/data buses in the backplane. The Sun can read/write to memories and memory-mapped control registers of Splash 2 via these buses. A detailed description of the system is given in [2]. We describe the major components below.

Each Splash 2 processing board has 16 processing elements ($X_1 - X_{16}$) in addition to a seventeenth Xilinx 4010 (X_0) which controls the data flow into the processor board. There is a 36-bit data path running through all the PEs. The PEs can get data either from their respective memory or from any other PE. The PEs are connected through a crossbar which is programmed by X_0. Each PE has 512 KB of private memory. The Sun host can read/write this memory. A broadcast path also exists by suitably programming X_0. The processor organization for a PE is shown in Figure 6.

Splash 2 system allows the capability of multiple PEs running the same instructions and receiving data through a global broadcast bus. The other advantage of Splash 2 is the superior Xilinx 4010-based FPGA which has better features than its predecessor Xilinx 3090. The individual memory available with each PE makes it convenient to store temporary results and tables.

Programming FPGAs is different from usual high-level programming in C or C++. The design automation process has two steps, namely: (i) simulation and (ii) synthesis. In simulation, we are interested in verifying the designed logic through VHDL. Usually this involves verifying the results as described in VHDL code. In synthesis, the main concern is to get the best placement of the logic in FPGA in a fashion which will also minimize the timing delay. In addition, we obtain timing for the entire logic. In case this timing is not acceptable, we repeat the design process.

To program Splash 2, we need to program each of the PEs (X_1- X_{16}), the crossbar, and the host interface. The host interface takes care of data transfers in and out of Splash 2 board. The crossbar sets the communication paths between PEs. A special library is available for these facilities for VHDL programming as described in [3].

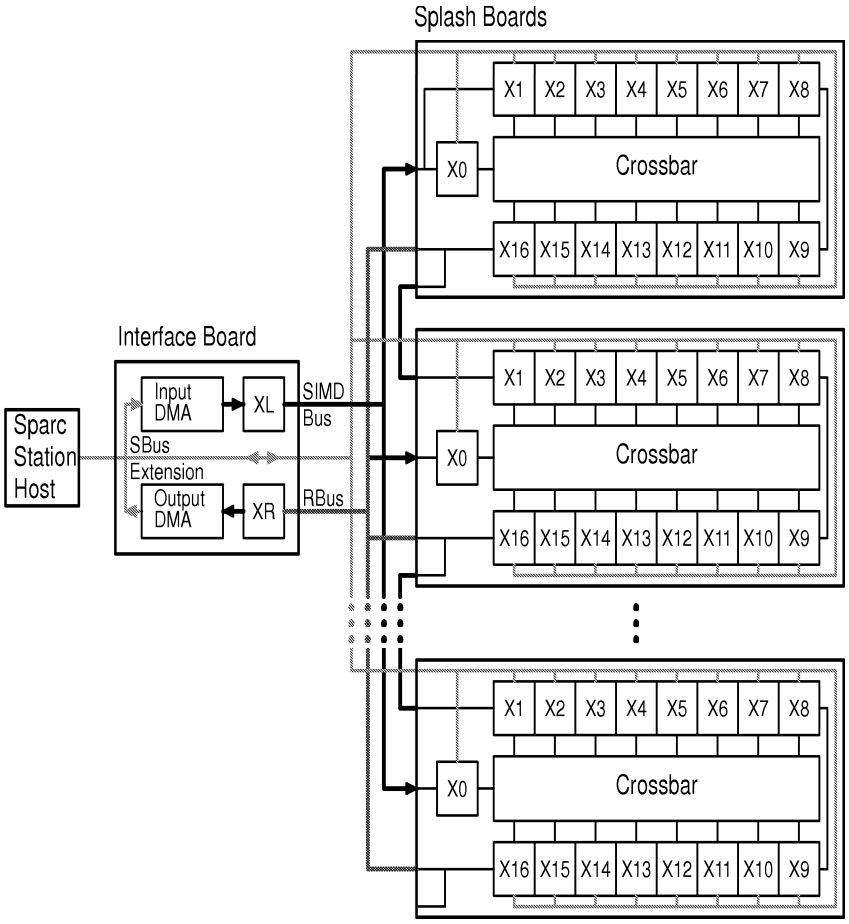


Figure 5: Splash 2 Architecture.

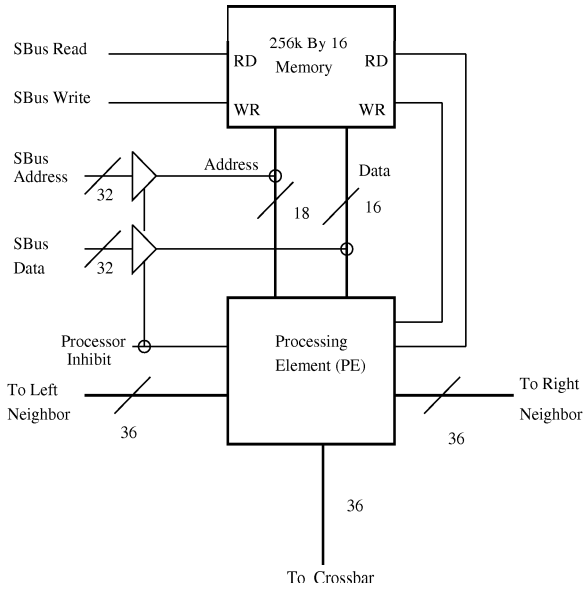


Figure 6: A Processing Element (PE) in Splash 2.

As Splash 2 is an attached slave processor, the host needs to take control of it. For this purpose a C-language interface is provided on the host. The host does the necessary disk I/O and uses Splash 2 system for the desired operations. The PEs and the host interface Xilinx bit stream files have to be generated using the utilities from the VHDL description. We get the speed estimates from the 'timing' utility for the Splash system clock speed.

4 Parallel Implementation

There are two kinds of parallelism which could be applied to the fingerprint matching problem, namely, at macro and at micro levels. The macro level involves splitting the fingerprint database over a set of processors each of which can carry out matching in its respective database to provide the results. Finally, the results of all the processors are merged at a central place to provide the answer to the query. At a micro level, each feature is matched against the set of reference features in parallel. In other words, imagine a series of processing elements carrying out the matching of features. They all receive a feature for checking if it could be paired with the feature the PE owns. If a query feature is matched to multiple reference features, then we will accept the reference feature with the closest match value. The process is repeated until all the features have been checked. In this paper, we describe a parallel algorithm suitable for micro-level parallelism. The type of architecture appropriate for micro-level parallelism is SIMD with a global

broadcast facility.

In principle, we can take the serial version of the algorithm and translate it into each PE. But this is not efficient. So, we parallelize the algorithm such that it utilizes the specific characteristics of the given architecture. The parallel version of the algorithm is given below:

1. Host operations

   - Query fingerprint

     The query fingerprint is read first and the following preprocessing is done:

     (a) The core point is assumed to be available. For each feature $\mathbf{f_i^q}$, i=1, 2, ... N, generate a 3-D "tolerance box". Enumerate a total of $(t_x * t_y * t_\theta)$ grid points in this box, where t_x is the tolerance in x, t_y is the tolerance in y and t_θ is tolerance in θ.

     (b) Allocate each feature to one PE in Splash 2. Repeat this cyclically, i.e., features 1-16 are allocated to PEs X_1 to X_{16}, features 17-32 are allocated to PEs X_1 to X_{16}, and so on.

     (c) Load the grid points in the tolerance boxes in step (a) into the memory of individual PEs.

   - Database fingerprints

     The host reads the database of fingerprints and sends the feature vectors for matching to the Splash 2 board.

     For each database fingerprint, the host performs the following operations.

     (a) Read the feature vectors.

     (b) Compute the common bounding box with the query fingerprint.

     (c) Send each of the feature vectors on the broadcast bus to all the PEs if it is within the bounding box of the query fingerprint.

   - Read result: For each database fingerprint, collect the number of paired features m_p^r, r=1, 2, ... K from Splash 2 system.

   - Score computation

     Same as in sequential method.

2. Operations on X_0

   Each feature vector received from the host is broadcast to all the PEs. If it can be paired, the global XOR bus will be driven low. Hence on low XOR, increment a counter.

   The host reads this counter value (m_p^r), and then resets it to 0.

3. Operations on each PE

   On receiving the broadcasted feature, compute its address in PE memory through a bit hashing function. If the data at the computed address is a '1', pull the global XOR bus low.

5 Performance of Sequential and Parallel Algorithms

The sequential algorithm executed on a Sun SPARCstation 10 performs at the rate of 70 matches per second (database and query fingerprints with approximately 65 features). The parallel implementation performs matching at the rate of $2.6 * 10^5$ matches per second. This is obtained from the 'timing' utility. The host interface part can run at 17.1 MHz and each PE can run at 33.8 MHz. Hence the overall application will run at the slower speed of the two, i.e., 17.1 MHz. Assuming 65 minutiae, on an average, in a database fingerprint, it will be matched at $2.6 * 10^5$ per second. The parallel implementation is independent of query fingerprint minutiae count. If we have multiple Splash 2 boards on the same host, each board could cater to a different query resulting in a higher throughput. The output of the timing utility, and simulation waveforms are shown in Figures 7-8.

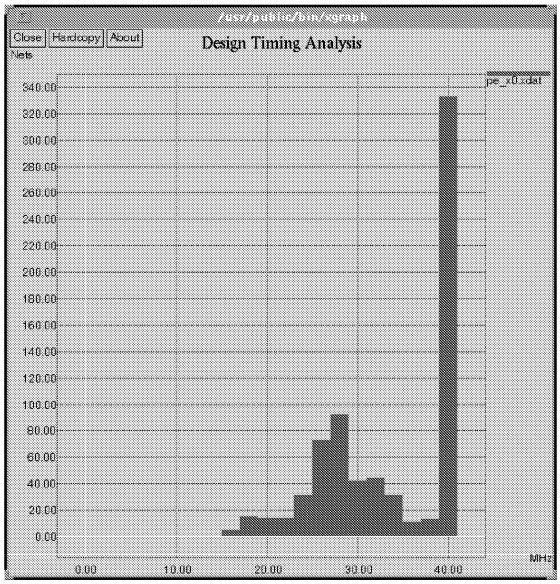
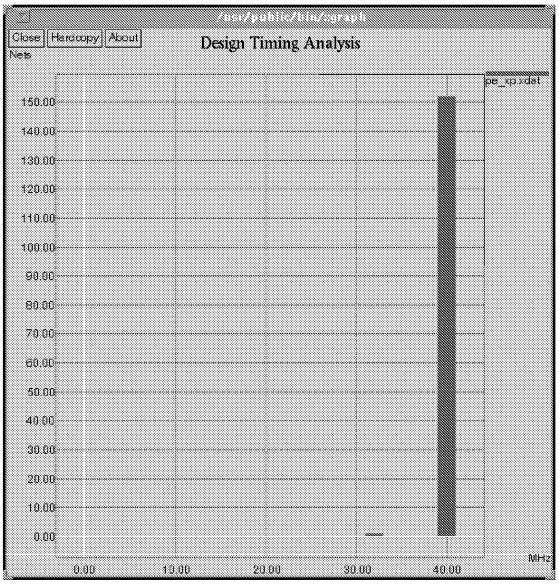


Figure 7: Timing Result for PE X_0.



Timing Result for PE X_i

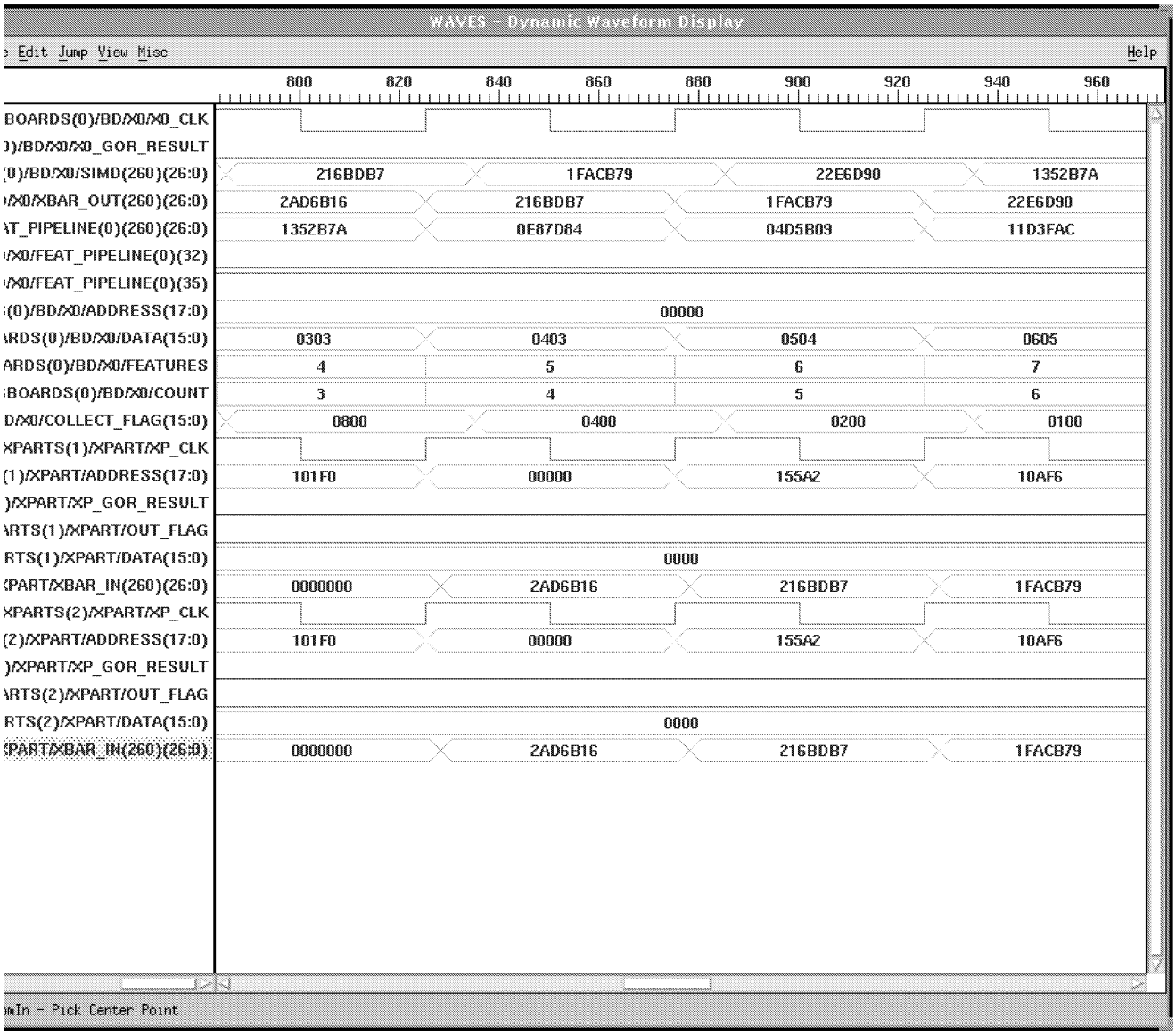


Figure 8: Simulation Waveforms for test data.

6 Conclusion and Future Work

The Splash 2 architecture is highly suitable for rolled fingerprint matching. The parallel algorithm has been designed to match the Splash 2 architecture, thereby resulting in optimum performance.

We are now in the process of building a small (≈ 100) fingerprint database to further test our algorithm. In the next phase of the project we plan to implement a minutiae detection algorithm and a latent fingerprint matching algorithm. Both these algorithms appear promising for implementing on Splash 2 architecture.

The speed can be further improved by replacing some of the soft macros on the host interface part (X_0) by hard macros. To sustain this matching rate, the data throughput should be at a rate of 0.26 million records per second from the fingerprint database. This may be a bottleneck for I/O subsystem.

Acknowledgment

This research was supported by a grant from the Supercomputing Research Center, Bowie, Maryland.

References

[1] Sir W. J. Herschel, The origin of fingerprinting, AMS Press, NewYork, 1974.

[2] D. A. Buell, A Splash 2 tutorial, Ver 1.2, SRC Tech Report SRC-TR-92-087, Feb. 1993.

[3] J. M. Arnold, Splash 2 Programmer's Manual, Ver 0.7, SRC, Dec. 1992.

[4] M. Gokhale, et al., "Building and Using a Highly Parallel Programmable Logic Array," IEEE Computer, Jan. 1991, pp. 81-89.

[5] J. M. Arnold, et al., "Splash 2," Proceedings of 4th Annual Symposium on Parallel Algorithms and Architectures, 1992, 316-322.

[6] J. M. Arnold and D. A. Buell, "VHDL programming on SPLASH 2", 3rd Intl. Workshop on Field Programmable Logic and its Applications, Sept. 1993, Oxford, England.

[7] J. M. Arnold, "The Splash 2 software environment", Proc. of IEEE Workshop on FPGAs as Custom Computing Machines, April 1993.

[8] J. H. Wegstein, "An automated Fingerprint Identification System", National Bureau of Standards Special Publication 500-89, Feb. 1982.

[9] H. C. Lee, R. E. Gaensslen (Ed.), Advances in Fingerprint Technology, Elsevier, New York, 1991.