PRACTICAL EVALUATION OF A RADIAL SOFT HASH ALGORITHM

F.-X Standaert[1,2], F. Lefebvre[3], G. Rouvroy[1,2], B. Macq[3], J.-J Quisquater[1,2], J.-D Legat[2]

[1]UCL Crypto Group, [2]Laboratoire de Microélectronique
[3]Laboratoire de Télécommunications et Télédétection
Université Catholique de Louvain

standaert,rouvroy,quisquater,legat@dice.ucl.ac.be
lefebvre,macq@tele.ucl.ac.be

ABSTRACT

This paper presents a practical evaluation of a RADIal Soft Hash algorithm, an image hashing technique used for image and visual content authentication. First we evaluate its collision resistance and susceptibility to geometrical deformations and image processing attacks. Second we investigate its actual efficiency on different platforms. Although software implementations allow managing sizes and throughput for most applications, they suffer from some troubles for real time applications such as video and for images in high definition. For these more constrained contexts, we propose hardware implementations. In practice, throughput up to 2^{18} (resp. 2^{26}) pixels/sec are managed in a single SW (resp. HW) implementation.

1. INTRODUCTION

Present networks allow millions of users to share their digital multimedia products and this allows the unlimited copy and transmission of almost every digital data. Therefore, the Digital Right Management issue constitutes a bottleneck for a large use of digital contents. The watermarking technique is a first attempt to solve copyright management. It intends to embed a hidden, robust digital signal in a multimedia content that allows the detection of an illegal copy and demand of legal responsibilities. However it is not a low cost process and the efficiency of such an approach is not yet unanimously recognized [6].

In recent writing about visual content authentication, the term image hashing has been introduced to refer to the computation of a content based image digest [3, 4, 13]. In accordance with this terminology, we also call hashing the extraction of a content-based image or video digest, but we make the distinction between cryptographic hashing and robust hashing. Hash functions are well-known in cryptography and are generally used for digital signatures. In essence, they summarize a message in a short and constant bit length digest, which uniquely identifies the original message. Cryptographic hashing has to be resistant to collision, and computationally non-invertible [10] (i.e it should be computationally impossible to construct a different file producing the same hash value). In cryptography, the output message digest dramatically changes when a single bit of the input message changes [10]. One says that cryptographic digests are discontinuous. Discontinuous hashes are useful to guarantee strong integrity and authenticity. However, in visual content management applications, continuous hash functions are preferred. A continuous hash function, also called robust hash function, alters the output message (or media) digest in proportion to the changes in the input message. When applied to image or video signals, such functions are designed to capture the essence of the visual content.

The purpose of robust image hashing is thus to define an image digest that satisfies two properties. On the one hand, similar to cryptographic message digest, the robust image digest characterizes the image in the sense that it uniquely identifies its content, i.e. the digests derived from a pair of visually distinct inputs have a low probability to be identical. On the other hand, the hashing process is robust in the sense that the digest is only slightly affected when the image changes due to compression or minor processing, i.e. visually indistinguishable images generate equal or similar digests. Conversely to cryptographic hashing, robust hashing is thus able to deal with visually non-significant changes of the content, and supports common manipulations like compression or reformating (e.g. spatial or temporal subsampling).

Because it defines a vector that identifies the image contents, robust hashing is an obvious solution for content identification and indexing. When used in combination with conventional cryptographic digital signature methods, robust hashing can also be used for integrity and authentication purposes [3, 9]. Finally, in watermarking, hashing enables the creation of payloads that depend on the media content, and which are thus resistant to the copy attack reported by Fridrich and al. in [4, 5].

In [7], a technique was proposed for copyright protection and video recognition (RASH). It is a new one-way function for images, based on the Radon transform, and adapted to the particularities of image and video signals. In this paper, we improve its authentication properties and define a new hash function (RADISH). We also evaluate its actual implementation efficiency. The rest of this paper is structured as follows. In section 2 and 3, we briefly describe the RASH and RADISH algorithms. Section 4 provides results that validate the collision resistance of the RADISH algorithm. Section 5 investigates its SW performances and sections 6, 7 discuss hardware implementation concerns. Performance evaluation on FPGAs are also provided. Finally, conclusions are in section 8.

2. RADON SOFT HASH ALGORITHM (RASH)

Reference [7] presented a new visual content descriptor for images. This method is based on Radon transform,$\mathscr{R}$:

$$\mathscr{R}(g,p,\theta)=\int_{-\infty}^{\infty} g(p.\cos\theta-q.sin\theta,p.\sin\theta+q.\cos\theta)\,dq \tag{1}$$

The transform properties allow obtaining good invariance against rotation, scaling and image processing attacks. To extend the Radon transform to discrete images, the line integral along $x.\cos\phi+y.\sin\phi=d$ is approximated by a summation of the pixels lying in the one-pixel-wide strip:

$$d-\frac{1}{2}\leq x.\cos\phi+y.\sin\phi\leq d+\frac{1}{2} \tag{2}$$

Reference [2] gives a quick and good approximation of this discrete Radon transform. However, the RAdon Soft Hash algorithm [7] suffers from some troubles due to a hight energy along the angle 45 and 135 (see figure 1). This is notably due to the fact that, in this first method, the pixel luminances along the line projection were summed.

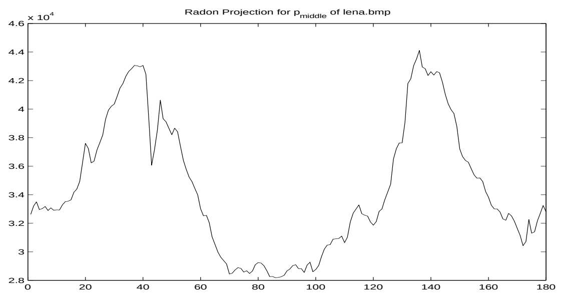


Figure 1: RASH feature vector for Lena.

In order to overcome this problem, we observed that the variance efficiently captures much better luminance discontinuities along the projection lines. In the image, these discontinuities correspond to edges that are orthogonal to the projection direction. Hence, the variance is expected to capture relevant information about the visual content of an image. This led us to define a new identification method, called RADIal Hashing. In the rest of the paper, we denote it as the RADISH transform.

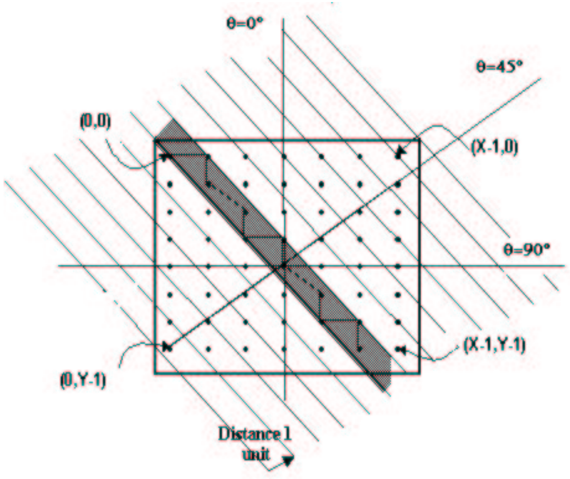


Figure 2: Image definitions.

3. RADIAL SOFT HASH ALGORITHM (RADISH)

Formally, we define the RADIal Hashing feature vector as follows. Let $\Gamma(\phi)$ denote the set of pixels (x,y) on the projection line corresponding to a given angle ϕ. Let (x',y') denote the coordinates of the central pixel. According to figure 2, $(x,y)\in\Gamma(\phi)$ if and only if:

$$-\frac{1}{2}\leq(x-x').cos\phi+(y-y').sin\phi\leq\frac{1}{2} \tag{3}$$

Let $I(x,y)$ denote the luminance value of the pixel (x,y), the RADISH feature vector $R[\phi]$, $0\leq\phi<180$, is then defined by:

$$R[\phi]=\frac{\sum_{(x,y)\in\Gamma(\phi)}I^2(x,y)}{\#\Gamma(\phi)}-\left(\frac{\sum_{(x,y)\in\Gamma(\phi)}I(x,y)}{\#\Gamma(\phi)}\right)^2 \tag{4}$$

Figure 3 present the RADISH feature vector obtained for Lena. The energy is now correctly spread in accordance

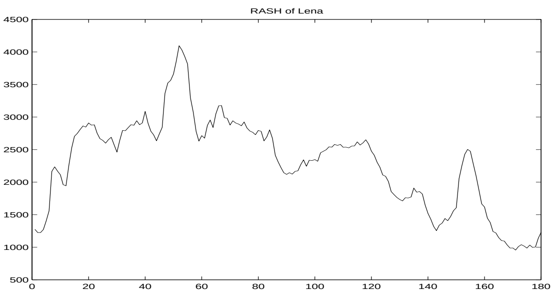


Figure 3: RADISH feature vector for Lena.

with the pixel luminance information and not according to the number of pixel along a certain direction.

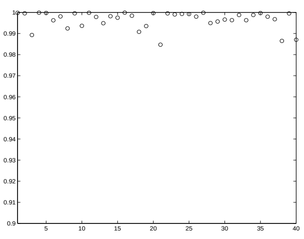
4. COLLISION RESISTANCE

To evaluate the robustness and collision resistance of our proposed hashing method, we experimented RADISH on 40 real-world images taken from the USC-SIPI database [12]. Note that those are preliminary results performed on a small data base. Further experiments are required, for example using consecutive frames in a movie or other transforms.

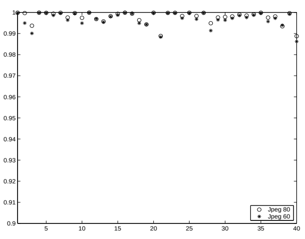
For each of the 40 images of the dataset, we considered 7 image processing attacks, generating 280 images, named processed images or intra images. The attacks are:

- **Filtering**: 3x3 Gaussian filtering with standard deviation of 0.5.
- **Compression**: JPEG compression with 80% and 60% quality factor.
- **Geometric**: scaling (factors = 1.2 and 0.8), rotation and rotation with centered cropping.

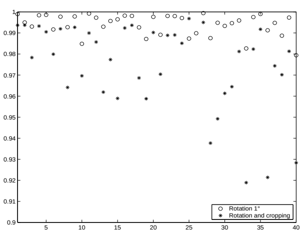
In figures 4, we observe that for all RADISH message digest, the peaks of cross-correlation (PCCs) between reference images and processed images are larger than 0.85.



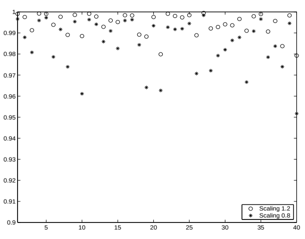
(a) Gaussian attack.



(b) Jpeg attack.



(c) Scaling attack.



(d) Rotation attack.

Figure 4: Peaks of cross correlation based on RADISH image digests for Intra images

To evaluate the risk of collision, we need to compare, for each original image from the dataset, the worst Intra matching with the best Inter matching. Given a reference original image, we classify the 280 processed images in Intra and Inter processed images, depending on whether they derived from the reference image or not.

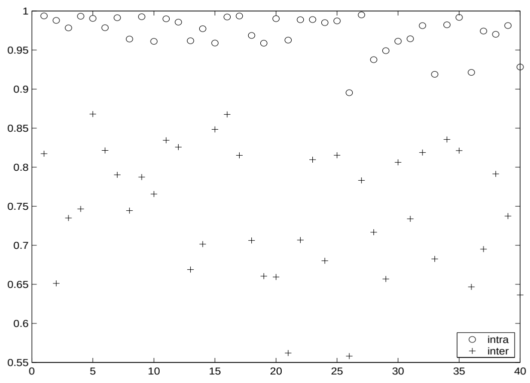


Figure 5: Collisions in RASDISH image digests.

Figure 5 presents the worst Intra PCCs and the best Inter PCCs for each image from the dataset. From this figure, we observe that all Intra PCC's are larger than 0.85, and that no Inter PCC lies under 0.85. We conclude that cross correlation is an efficient way to compare two RADISH digests, and that 0.85 is a good threshold to decide whether two images are visually similar or not.

5. SOFTWARE IMPLEMENTATIONS

A software implementation of the RADISH transform allowed us to evaluate its time-performances for different image sizes, from 512 $\times$ 512 to 4000 $\times$ 2000 pixels. We observed that the memory requirements of the algorithm do not significantly increase with the image size whereas its computational requirements linearly depend on it. Therefore, the throughput of the RADISH transform is independent from the image size. On a AMD Athlon, XP 1800 with 512 MBRAM, we found 2^{18} pixels/sec. In practice, the RADISH of a 512 $\times$ 512 image is computed in 1 second whereas a 4000 $\times$ 2000 image will need about 30 seconds. In the next section, we investigate the relevance of a hardware coprocessor for computing the RADISH transform.

6. HARDWARE IMPLEMENTATIONS

For hardware implementations, we assume that the pixels are received in a one by one serial way. That is, we first receive $(0,0)$, then $(1,0)$, $(2,0)$, ... , and finally (X,Y). The resources needed to implement the RADISH are divided as follows:

R1. Resources for computing condition (3). Those are not mandatory as this computation does not change with the input data and may therefore be performed offline.

R2. Squaring multipliers for the left term of equation (4).

R3. Averaging adders for the sums of Eq. (4) (i.e. $\sum I$, $\sum I^2$, $\#\Gamma(\phi)$).

R4. Registers or memory to store the averages of Eq. (4).

We may therefore build different implementation scenarios:

6.1 Serial-serial

This scenario refers to the situation where we compute condition (3) for one pixel (x,y) and one angle ϕ in one clock cycle. The resulting design has the lowest possible area requirements and a low throughput. As only one pixel is managed by clock cycle, only one multiplier is necessary for the squaring operation of equation (4). For the averaging, we need a memory with 180*3 addresses and three adders[1] to compute the different sums of equation (4). In terms of throughput, if the work frequency is f, we expect to have a throughput of about $\frac{f}{180}$ pixels/sec. It should not significantly improve software performances.

6.2 Serial-parallel

This scenario refers to the situation where we compute condition (3) for one pixel and all the 180 angles ϕ in one clock cycle. The resulting design needs to compute condition (3) 180 times in parallel (which, again, can be done offline and stored in a memory). However, as only one pixel is managed by clock cycle, we still only need one multiplier. For the averaging, we need the same number of memory addresses, but they have to be accessed in parallel so that we need 180*3 registers. 180 times more adders[2] are also necessary. As a consequence, the expected throughput becomes f pixels/sec.

6.3 Parallel-parallel

This scenario refers to the situation where we compute condition (3) for several (n) pixels and all the 180 angles ϕ in one clock cycle. Compared with the previous scenario, we have to multiply by n the number of times we compute equation (3) and the number of multipliers for the squaring operations of equation (4). As several pixels may influence the same angle in one clock cycle, we need either multi-operand adders or additional FIFO memories for the averaging of equation (4). The resulting design has an increased complexity and an expected throughput of $f.n$ pixels/sec.

6.4 Comparisons

The estimations for the different implementation scenarios are summarized in Table 1, where the symbol * indicates that additional resources are needed to deal with multiple pixels. (M) means that the storage is implemented in a single access memory and (R) means that the storage uses registers. Based on these estimations, the serial-parallel scenario appears to be an interesting combination of circuit size, throughput and simplicity.

[1]Due to the three sums that we have to compute, that is $\sum I^2$, $\sum I$ and $\#\Gamma$.

[2]Because some pixels are included in several angle lines, e.g. (x',y') is included in all of them (although in practice most pixels influence only one angle line).

	S-S	S-P	P-P
#R1 (may be offline)	1	180	180.n
#R2	1	1	n
#R3	3	180.3	180.3*
#R4	180.3 (M)	180.3 (R)	180.3*(R)
Throughput	$\frac{f}{180}$	f	$n.f$

Table 1: Area and throughput (pixels/sec) estimations for different implementation scenarios.

	Nbr LUTs	Nbr Flip flops	Nbr slices
Cond.(3)	84	19	47

Table 2: Implementation results for condition (3).

In the next section, we investigate its efficient implementation for a 512 x 512 pixels image with 8-bit luminance, with or without precomputation for condition (3).

7. EFFICIENT IMPLEMENTATION OF A SERIAL-PARALLEL ARCHITECTURE

This section describes the FPGA implementation of a serial-parallel architecture. We provide implementation results in the Xilinx Virtex-II technology [14]. Synthesis and implementation were performed with Xilinx ISE 6.1. The frequency is estimated after implementation and the hardware cost is evaluated by the number of LUTs, registers and slices. A short description of these FPGA components is given in appendix.

7.1 Computation of condition (3) (facultative)

Let X (resp. Y) be the number of pixels by line (resp. column) as suggested in Figure 2. For every $(x,y),\phi$, we want to compute the following condition:

$$-\frac{1}{2} \leq (x-x').cos\phi + (y-y').sin\phi \leq \frac{1}{2} \qquad (5)$$

As the pixels are provided in a one by one serial way, we observe that $(x-x').cos\phi + (y-y').sin\phi$ may be modified in only three different manners:

1. Initially, it is set to: $init_\phi = -x'.cos\phi - y'.sin\phi$.
2. For a new pixel, we add a constant value $a_\phi = cos\phi$.
3. For a new line, we add a constant value $b_\phi = -(X-1).cos\phi + sin\phi$.

If we store the values for a_ϕ, b_ϕ and $init_\phi$ in a memory, it is therefore possible to compute condition (3) with only one adder, one register and two comparisons (these comparisons are actually implemented as one adder and one substractor), as it is shown in Figure 6. The implementation results are in Table 2.

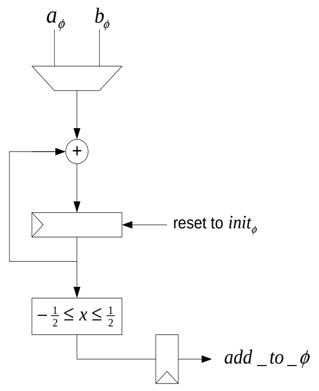


Figure 6: Computation of condition (3).

	Nbr LUTs	Nbr Flip flops	Nbr slices
Av. circuit (3)	94	54	54

Table 3: Implementation results for the Av. of (4).

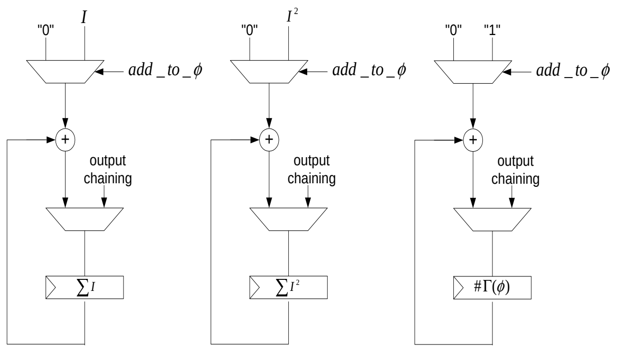


Figure 7: Averaging circuit (4).

7.2 Computation of the RADISH (4)

Thanks to the *add_to_*ϕ signals, we can compute the different sums of equation (4) for a fixed ϕ with three adders, three registers and three multiplexors. We decided to implement only $\sum_{(x,y)\in\Gamma(\phi)} I^2(x,y)$, $\sum_{(x,y)\in\Gamma(\phi)} I(x,y)$ and $\#\Gamma(\phi)$ in hardware. The final squaring, division and substraction can be more efficiently implemented in software. An efficient solution to do it would be to use the embedded processors available inside some recent FPGAs. Anyway, these operations are not critical in software and do not involve any need for hardware implementation. The design is represented in Figure 7 and its implementation results are in Table 3. Remark the use of output multiplexors allow chaining the different averaging registers in order to have a serial output of the 180.3 coefficients os the RADISH.

	Nbr LUTs	Nbr Flip flops	Nbr slices
full RADISH	28979	13174	16846

	Nbr LUTs	Nbr Flip flops	Nbr slices
RADISH + prec.	17012	9756	9770

	Frequency	Throughput
RADISH	75 Mhz	75.10^6 pixels/sec

Table 4: Complete implementation results.

7.3 Complete implementation

The complete implementation uses the previous cells 180 times in parallel. An additional multiplier is used to compute the current I^2. Some additional logic is required for the control signals. After implementation within a XILINX Virtex2-6000, we obtain the results of Table 4 for a 512 $\times$ 512 image. The hardware cost is presented with or without the precomputation and illustrates that removing the computation of equation (3) allows us to roughly divide the cost by two. For larger image sizes, the work frequency is only very slightly reduced (e.g. 74 Mhz for 4000 $\times$ 2000 images). It is due to the slight modification in the averaging adders size. As a consequence, we may assume that, as in the case of SW implementations, the throughput is independent from the image size.

8. CONCLUSION

We evaluated the performances of a new robust hash function and presented results to validate its authentication properties. Based on a simple parallelization of the algorithm, an FPGA implementation of the RADISH signature of an 8-bit luminance image has a throughput of 75.10^6 pixels/sec. Compared with software implementations, we compute the hash of a complete 512 x 512 image in 3.5 milliseconds in place of about 1 seconds. These performances allow the RADISH transform to be an efficient solution for visual content authentication in a wide range of applications. Associated with a frame detection, a software implementation of the RADISH allows a direct computation of small signatures of a video content. Combined with watermarking, its geometrical properties also permit the correction of certain critical deformations for watermark detection. Finally, hardware implementations allow dealing with high definition images and high throughput videos. Further research is required in a deeper evaluation of the collision properties of the hash function, using larger image databases.

Acknowledgements: The authors would like to thank the anonymous referees for their interesting comments.

REFERENCES

[1] P. Bas, J.M. Chassery, B. Macq, “Geometrically Invariant Watermarking Using Feature Point”, IEEE TRansactions on Image Processing, vol 11, no 9, pp 1014-1028, 2002.

[2] Martin L. Brady and Whanki Yong, “Fast Parallel Discrete Approximation Algorithms for the Radon Transform”, *SPAA*, 1992, pp.91-99.

[3] S. Bhattacharjee, M. Kutter, “Compression Tolerant Image Authentication”, in the proceedings of the 5th IEEE International Conference on Image Processing, pp 435-439, Chicago, USA, October 1998.

[4] J. Fridrich, M. Goljan, “Robust Hash Functions for Digital Watermarking,” in the proceedings of *ITTC 2000*, Las Vegas, USA, 2000.

[5] J. Fridrich, M. Goljan, “Visual Hash for Oblivious Watermarking”, in the proceedings of SPIE 2000, Security and Watermarking of Multimedia Content, pp 286-294, San Jose, CA, USA, 2000.

[6] C. Herley, “Why Watermarking is Nonsense?”, IEEE Signal Processing Magazine, no 5, pp 1011, September 2002.

[7] F. Lefebvre, B. Macq and J.-D. Legat, “RASH : RAdon Soft Hash algorithm”, in the proceedings of EUropeean SIgnal Processing COnference, 2002, Toulouse

[8] C. Lin, S. Chang, “Generating Robust Digital Signature for Image/Video Authentication”, in the proceedings of Multimedia and Security Workshop, ACM Multimedia, Bristol, UK, September 1998.

[9] C. Lin, S. Chang, “A Robust Image AUthentication Method Distinguishing JPEG Compression from Malicious Manipulation”, IEEE Transactions on Circuits and Systems for Video Technology, February 2001.

[10] A. Menezes, V. Oorschot, S. Vanstone, Handbook of Applied Cryptography”, CSC Press, 1998.

[11] J. Oostveen, T. Kalker, J. Haitsma, “Visual Hashing of Digital Video: applications and techniques”, SPIE Applications of Digital Image Processing, SAn Diego, USA, August 2001.

[12] “USC-SIPI image database”, available at http://sipi.usc.edu/services/database/Database.html

[13] R. Venkatesan, S.M. Koon, M.H. Jakubowski, P. Moulin, “Robust image hashing”, *Proceedings of the International Conference on Image Processing (ICIP)*, 2000.

[14] Xilinx: “Virtex-II Field Programmable Gate Arrays Data Sheet”, http://www.xilinx.com.

APPENDIX

All the implementation results provided in this paper were obtained using Xilinx Virtex-II devices [14]. In general, FPGAs may be viewed as a “sea” of programmable logic gates where the logic, but also the routing are user programmable. In this section, we briefly describe the logic resources in Virtex-II FPGAs.

The main component of the Xilinx Virtex-II devices is the Configurable Logic Block (CLB) that is made up of two slices, each one divided into two Logic Cells (LC). An LC includes a 4-input function generator, carry logic and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Figure 8 shows a simplified view of a single slice.

Virtex-II function generators are implemented as 4-input LUTs that can also provide a 16×1-bit synchronous RAM or a 16-bit shift register. In addition, the F5 multiplexer in each slice combines the LUT outputs. This combination provides a function generator that implements any 5-input function, a 4:1 multiplexer, a 32×1-bit synchronous RAM or selected functions of up to nine bits. Similarly, the F6 multiplexer combines the outputs of all four LUTs in the CLB by selecting one of the F5-multiplexer outputs. Finally, the arithmetic logic includes fast carry chains and additional logic gates (*e.g*. XORCY) to improve the efficiency of adder/multiplier implementations. Additional resources also allow combining different slices.

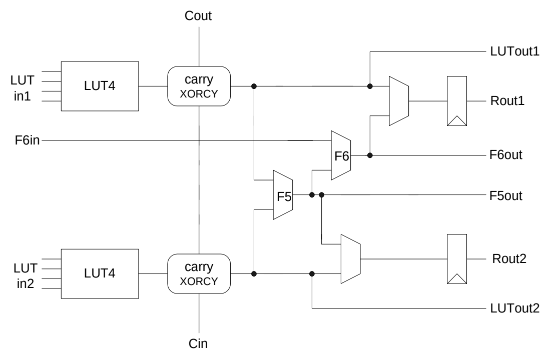


Fig. 9. The Virtex-II slice.

The storage elements can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches. The D inputs can be driven either by the function generators within the slice or directly from slice inputs, bypassing function generators.

Finally, Virtex-II FPGAs incorporate several large RAM Blocks (RAMB). These ones complement the distributed LUT implementations of RAMs. Every block is a fully synchronous dual-ported RAM with independent control signals for each port. The data widths of the two ports can be configured independently.