VLSI System Design Using Asynchronous Wave Pipelines: A 0.35 μm CMOS 1.5 GHz Elliptic Curve Public Key Cryptosystem Chip

O. Hauck, A. Katouch*, and S. A. Huss
{hauck | katouch | huss}@vlsi.informatik.tu-darmstadt.de
Integrated Circuits and Systems Lab Departments of CS and EE
Darmstadt University of Technology Germany

Abstract

This paper presents VLSI system design using asynchronous wave pipelines (AWPs) with a public key crypto chip as an example. The design challenges imposed by the crypto chip include very wide data paths, bit-level wave pipelining, hierarchical control resulting in different frequency domains, and interfacing synchronous registers with asynchronous controllers and data paths. The timing analysis indicates that AWPs operate more safely than synchronous wave pipelines. At the circuit level, SRCMOS is shown to be superior to previously proposed logic styles for wave pipelining. The same circuit style applies for both data path and control. Following some mathematics and cryptography background, the architecture of the chip is detailed whose outstanding feature is a wave pipelined Massey-Omura finite field multiplier. Simulations from layout of key circuits running at a rate of 1.5 GHz in a 0.35 μm CMOS process demonstrate the feasibility of the AWP concept.

I. Introduction

Given the overhead of synchronous designs employing a global clock on the one side and the overhead of full handshake asynchronous circuits on the other side, we developed the AWP architecture as a design option for self-timed high-throughput, fine-grain pipelined systems [1, 2]. AWPs avoid the overhead related to acknowledge signals and sacrifice elasticity to gain higher speed. In addition, AWPs allow safe wave pipelining by removing the double-sided constraint on the cycle time which makes system design with synchronous wave pipelines so difficult. To this end, a tightly matched bundled delay element is necessary. AWPs are currently a custom methodology, dedicated tools are not yet available. The primary emphasis of AWPs is to minimize sequencing overhead—may it be caused by synchronous clocks or by handshake circuits—and to push CMOS performance to the limit. Issues like portability, speed independence, verification and synthesis are as yet of secondary concern.

The contribution of this paper, compared to [2], is twofold: first, in section II the AWP is analyzed and compared with the synchronous wave pipeline more thoroughly than in [2]. A new idea for a controller architecture suitable for AWPs is presented that can implement any FSM specification. On the circuit front, we have abandoned static pass transistor logic in favor of dynamic self-resetting logic. Section II.A describes the special SRCMOS circuits we are using for AWPs.

Second, as a new application sections III and IV describe an AWP 261-bit public key crypto chip based on the discrete logarithm problem in finite abelian groups over elliptic curves over $F_{2^{521}}$. Public key schemes are very computing-intensive and usually quite slow, therefore the speedup of such a system is of practical interest. It will become clear that the AWP is a good fit for this system, because in conventional implementations the critical part, a finite field multiplier, has high latency and implies a long cycle time. Pipelining this multiplier is believed to be very difficult with techniques other than the AWP due to the data path having as many as 783 bits at the widest slice. Any scheme that would have to drive a clock or precharge through the whole width of the data path would introduce significant overhead and require larger margins. The AWP, in contrast, needs latching only at the input and output of the multiplier. Simulations from the layout of the critical part show three waves in the logic at a throughput of 1.5 GHz. As the algorithm implies a control hierarchy of three levels, only the lowest level performing finite field arithmetic runs at full speed, while the mid level performing elliptic curve arithmetic and the top level implementing a double-and-add algorithm for multiplication of a base point $P_0$ on the elliptic curve by a key $k$ have lower activity by orders of magnitude. This is naturally accommodated by the asynchronous control so that every part of the system runs at adequate speed thereby saving power. Finally, section V gives a summary.

II. Asynchronous Wave Pipelines

1. General Framework

Consider the generic pipeline stage in Fig. 1, where a combinational logic block is surrounded by edge-triggered registers that are clocked by delayed versions of a global clock. The input clock is skewed by $\Delta_i$, the output clock by $\Delta_o$. By choosing the skew between output and input $\Delta = \Delta_o - \Delta_i$ appropriately, either a conventional synchronous pipeline, a synchronous wave pipeline, or an asynchronous wave pipeline results. Note that if the clock distribution is implemented as in Fig. 1, i.e., by tapping the clock and employing skew ele-
ments, then the clock is synchronous and global, whereas for \( \Delta > 0 \) a delay element can be used along the data path which delays the clock at the output side by \( \Delta \) with respect to the input. In the latter case the clock is asynchronous and local, corresponding to the request signal of the handshake protocol.

By transitivity, (4) implies:

\[
T_{clk} \geq (t_{\text{max}} - t_{\text{min}}) + t_{\text{setup}} + t_{\text{hold}} + 2 \cdot t_{\text{show}}
\]

Equation (5) shows that the cycle time is bounded by delay variation in the logic, register overhead and clock skew.

Similarly, all internal gate output nodes \( i \in \mathcal{G} \) have to be stable for some time \( t_{\text{stable}}(i) \) in order to prevent vanishing pulses:

\[
T_{clk} \geq (t_{\text{max}}(i) - t_{\text{min}}(i)) + t_{\text{stable}}(i) + t_{\text{show}} \quad \forall i \in \mathcal{G}
\]

The minimum cycle time is simply the minimum time satisfying both (4) and (6) and does not depend on \( k \). For each value of \( k \) there exists some \( \Delta \) that achieves the minimum cycle time. Furthermore, in a multiple stage system constraints (4) and (6) have to be satisfied for all stages. Accordingly, the worst-case stage determines the maximum frequency.

The internal node constraint (6) sets the upper limit to the raw combinational logic frequency and is independent from the clocking scheme. In contrast, there exist various combinations of \( k \) and \( \Delta \) satisfying (4) which refer to different clocking schemes. In the following, combinations of \( k \) and \( \Delta \) corresponding to conventional synchronous pipelines, synchronous wave pipelines and AWPs are compared with respect to cycle time, frequency range, and tolerance of parametric variations, first for feedforward structures and then for those including feedback.

2. Feedforward Structures

First, for \( k \neq 0 \) and \( k \neq 1 \), we can write (4) as

\[
\frac{t_{\text{max}} + t_{\text{d}} + t_{\text{setup}} + t_{\text{show}}}{k} \leq T_{clk} \leq \frac{t_{\text{min}} + t_{\text{d}} + t_{\text{setup}} + t_{\text{show}} - \Delta}{k}
\]

This case corresponds to the synchronous wave pipeline and exhibits a two-sided constraint on the cycle time, equivalent to a frequency range for valid operation. From (7) it follows that the allowable frequency range gets more narrow as \( k \) is increased and that the ranges are distinct for different values of \( k \). Furthermore, any imprecision in the value of \( \Delta \) additionally narrows the frequency range for valid operation. Traditionally, in synchronous wave pipelining one tried to keep \( \Delta \) small because it was believed that skew elements \( \Delta_i \) and \( \Delta_o \) with small values are easier implemented. This, in turn, due to (1), requires higher values of \( k \) to achieve the minimum cycle time which results in operation in a narrow frequency range. This property of synchronous wave pipelines is the main obstacle in the way to larger systems employing this technique, because for different parts in the system there may not even exist a single frequency where those parts would all work. This together with the fact that clock skew and variation in the delay elements not only reduce the maximum possible frequency but also further narrow the valid frequency ranges renders structures with \( k > 1 \) unsuitable for system design as timing gets more and more complex with increasing system size.

\[\text{Fig. 1. Generic pipeline stage.}\]

The following analysis assumes edge-triggered registers\(^1\). We define some parameters using the framework in [3].

\[\mathcal{G} \quad \text{set of all gate output nodes in logic}\]

\[t_{\text{min}}, t_{\text{max}} \quad \text{minimum and maximum logic delay}\]

\[t_{\text{min}}(i), t_{\text{max}}(i) \quad \text{minimum and maximum delay from input to internal node } i \in \mathcal{G}\]

\[t_{\text{stable}}(i) \quad \text{minimum time internal node } i \in \mathcal{G}\]

\[T_{clk} \quad \text{clock period or cycle time}\]

\[\Delta_i, \Delta_o \quad \text{intentional skew at input and output registers}\]

\[\Delta = \Delta_o - \Delta_i \quad \text{delay between input and output clock}\]

\[t_{\text{show}} \quad \text{uncontrolled clock skew at register}\]

\[t_{\text{setup}} \quad \text{set-up time of register}\]

\[t_{\text{hold}} \quad \text{hold time of register}\]

Output data is latched by the output clock at some time \( t \) relative to the global clock, where

\[t = k \cdot T_{clk} + \Delta_o
\]

The important parameter \( k \) is called global clock latency in [3] and equals the number of clock edges that arrive at the output register between launching a data wave at the input register and latching the same data wave at the output register.

In order for the output data being latched after the latest bits have arrived, \( t \) is bounded below by:

\[t \geq \Delta_i + t_{\text{max}} + t_{\text{d}} + t_{\text{setup}} + t_{\text{show}}
\]

The output data must be latched before the earliest bits of the next wave arrive which bounds \( t \) above by:

\[t \leq T_{clk} + \Delta_i + t_{\text{min}} + t_{\text{d}} - t_{\text{hold}} - t_{\text{show}}
\]

Combining (2) and (3) gives the following two-sided constraint on \( k, T_{clk} \) and \( \Delta \):

\[t_{\text{max}} + t_{\text{setup}} + t_{\text{show}} \leq kT_{clk} + \Delta - t_{\text{d}} \leq T_{clk} + t_{\text{min}} - t_{\text{hold}} - t_{\text{show}}
\]\n
\(^1\) A similar analysis applies to level-sensitive latches.
For $k = 1$, equation (4) reduces to
\[
T_{cl,k} \geq t_{\text{max}} + t_d + t_{\text{setup}} + t_{\text{show}} - \Delta 
\] (8)
\[
\Delta \leq t_{\text{min}} + t_d - t_{\text{hold}} - t_{\text{show}} 
\] (9)
If one takes the limit for $\Delta$ in (9) one gets again (5), but for $\Delta = 0$ we have the conventional synchronous pipeline with (8) bounding the clock cycle by worst-case logic delay plus overhead:
\[
T_{cl,k} \geq t_{\text{max}} + t_d + t_{\text{setup}} + t_{\text{show}} 
\] (10)
Finally, for $k = 0$ equation (4) reduces to
\[
T_{cl,k} \geq \Delta \geq t_{\text{max}} + t_d + t_{\text{setup}} + t_{\text{show}} 
\] (11)
(12)
Taking the limit for $\Delta$ in (12) gives the AWP. In this case the frequency is only bounded above and the minimum cycle time is bounded by differences in logic delay. This scheme is implemented as shown in Fig. 2.

![Fig. 2. Generic AWP stage.](image)

The global synchronous clock is replaced by a local asynchronous request signal which is delayed along the data path by a wave pipelined delay element with value $\Delta_{AWP} = t_{\text{max}} + t_d + t_{\text{setup}} + t_{\text{show}}$. There are several advantages of the AWP: first, the asynchronous request signal is more easily distributed due to its local nature. The load seen by the request line at any location depends only on the width of the data path. This load corresponds to one register only whereas the scheme in Fig. 1 loads the clock with the total number of registers in the system. Second, equations (11) and (12) actually refer to the implementation of Fig. 1 and can be simplified for the implementation in Fig. 2. The parameter $t_{\text{show}}$ in (11) and (12) is in half due to clock skew at the input register in Fig. 1 which is not present in Fig. 2 as the input request does not come from a global clock but is the output request of the previous stage. The only skew that has to be accounted for is the one which refers to the imprecision in the value of the delay element $\Delta_{AWP}$, and therefore we can write $t_{\text{show}}$ in (5) for $2 \cdot t_{\text{show}}$. The prize we have to pay in the AWP is that the delay element $\Delta_{AWP}$ takes more area than the small skew elements $\Delta_i$ and $\Delta_s$ of synchronous wave pipelines. As $\Delta_{AWP}$ is in the order of $t_{\text{max}}$, special emphasis must be given to the realisation of the delay element to control the latency mismatch between the request line and the logic in the presence of process, temperature, and voltage (PTV) variations, denoted by $t_{\text{show}}$. On the other hand, the delay $\Delta_{AWP}$ may be easier to implement than skew elements $\Delta_i$ and $\Delta_s$ because a copy of some logic path can serve as delay element. In contrast, for skew elements there is no model in the logic.

An important consequence of $k = 0$ is that a data wave is latched at the output by the same delayed request that launched that data wave at the input, in other words, data waves in the logic and control information in the request line propagate in a coherent manner. This property links AWP's to other asynchronous schemes. With respect to the protocol, AWP's are similar to Micropipelines except that they are lacking the acknowledge signal and thus elasticity. AWP's save power by removing many of the registers of conventional pipelines and by using only a minimum, local request infrastructure. Under absence of input data they are naturally idle. Furthermore, the cascaded delay elements distribute power consumption more uniformly over time. Finally, AWP's can be plugged together without complicating global timing issues.

3. Feedback Structures and FSMs

This section will analyze the implications of the presence of feedback in AWP structures and the design of FSMs. Consider the AWP with feedback in Fig. 3 where the latched output is fed back to the input register.

![Fig. 3. AWP with feedback.](image)

Assume data is arriving at the primary input at some constant rate. After some delay, data from the feedback path will arrive at the input register at the same rate but with different phase. The issue at the input register is to latch in the primary input with correct data from the feedback path. In order to avoid that unsettled data from the feedback path is latched in, the output from the logic is latched before being fed back. This ensures that only settled feedback data will arrive at the input register and is latched in with some input data. If $n$ waves are active in the logic, then data from primary input propagates through the logic and merges with the $n$-th next primary input data at the input register. This is true for all frequencies but one has to wait for one logic delay until a new input frequency has propagated through the logic.

As the input frequency cannot be assumed as perfectly constant the question arises as to how much variation in the input rate can be tolerated so that the number of cycles between inserting some data at the input and merging the logic results for that data with new input remains constant. This is important in order for the semantics of the circuit not to change. The answer to this question is that for every possible frequency in the logic there is a window of one cycle time where the feedback data can arrive at the input register in order to be latched with the same input data. For maximum toleration against frequency change feedback data should be centered in that window, because then data up to a half cycle...
too early or too late will be correctly latched. Fig. 4 illustrates the situation.

In order to make a feedback configuration especially robust for operation at some constant frequency, a delay element can be used in the feedback path.

Fig. 4. Waveforms of a feedback AWP with 2 waves showing primary input request and feedback data at input register.

The arrangement in Fig. 3 can be used for both data paths —as is the case in the shift-and-add accumulator in distributed arithmetic for DCT transform [2]— and state machines. The limitation is that machines of this sort can only transition between states under presence of primary input because only the request of the primary input can launch a new state into the logic. Autonomous state transition is not possible as without input activity and therefore request activity a new state cannot propagate through the input register. In [2] we proposed to partition the machine into autonomous and non-autonomous parts to overcome this problem. However, there is a more elegant solution. We introduce one additional output signal “X” which examines the current state and is “1” if and only if this state transitions unconditionally, i.e. without primary input, to the next state. This signal is designed so that all next state bits are stable before it and is then OR-ed with request signals from all primary inputs. In essence, autonomous states provide for their own request signal. This is illustrated in Fig. 5, where the logic and thus X are assumed to be pulsed.

Fig. 5. AWP controller architecture.

Note that if only one wave is active in the logic we have the usual state machine and the input rate is not bounded below. In this case the output register can be omitted.

Finally it is emphasized that this controller architecture allows general FSM specifications to be realized while avoiding all hazard problems as the feedback is broken by the input register. The controllers described in section IV are constructed using this scheme.

4. SRMOMS Circuits for AWPs

Fig. 6 shows the circuit style used in the multiplier of the crypto processor. It is a dual-rail cross-coupled dynamic logic with local precharge. Dual-rail is needed as the critical net is a large XOR-tree which is non-monotonic. One of the complementary NMOS trees pulls the precharge node down which is subsequently pulled high after one inverter pair delay. Therefore the outputs are pulsed and, in order to avoid crowbar current, the inputs to these gates have to be pulsed as well. Therefore the shown SRMOMS circuits can be easily cascaded. Pulldown on one side activates the keeper on the other side so that after a switching event both floating nodes are refreshed, the one on the switching side through precharge activated by the feedback inverter, while the complementary floating node is refreshed by the keeper. If the gate idles and neither side switches, both floating nodes are leaking similarly and will eventually activate the keeper for the complementary node thus preserving precharge integrity.

The NMOS trees are designed so that ideally there is only one path conducting to ground. The length of all possible paths in the NMOS trees of gates in the same sequential position in the combinational logic must be equalized in order for the data word to be coherent. Fanout load including interconnect has to be equalized for all gate outputs. These requirements are easily met in a XOR-tree, as there is always only one possible path in the NMOS tree and no padding devices are needed.

Our approach differs from that in [4] in that our design the precharge is fully local and does not use tapping a quasi-global reset timing chain. For further details on SRMOMS circuit design for AWPs rf. [5].

III. Elliptic Curve Public Key Cryptosystem

1. Background

Cryptography is becoming important given the transmission of increasing amounts of sensitive data in electronic form over networks. E.g., when sending one’s credit card number over the internet one would like to be sure that no other person can obtain the number by wiretapping. There are mainly two categories of cryptographic methods for enciphering and deciphering data: symmetric schemes use the same key for enciphering and deciphering, asymmetric schemes use different
keys. Symmetric schemes are also called block ciphers and typified by DES, asymmetric schemes are also called public key cryptosystems, typified by RSA. Block cipher chips are able to process data with Gb/s rates while public key crypto chips are orders of magnitude slower due to the algorithmic complexity and large key length. However, public key schemes are by far more secure, e.g. 1024-bit RSA is still safe while 56-bit DES is not. For the secure transmission of large amounts of data one combines the strengths of both schemes in that the key of the symmetric scheme is changed frequently and encrypted by the public key cryptosystem before sending. The frequency the key can be changed with depends on the speed of the public key cryptosystem. In summary, even though block ciphers will always be faster than public key cryptosystems, for better security it pays to improve both schemes.

In the following we will concentrate on a public key cryptosystem based on elliptic curves. Elliptic curve cryptography has emerged as an alternative to RSA because it is believed to give higher security per key bit, i.e. one can work with shorter keys [6]. Common to all public key schemes is a computationally hard problem on which the security of the cryptosystem is based. In RSA it is the factoring problem, meaning that it is easy to multiply two large primes but impossible with current techniques to factor a 1024-bit number which is a product of two large primes. The other problem whose intractability is exploited in this manner is the so-called discrete logarithm problem in large finite groups. This refers to the fact, that, given a large finite group \( G \) and group elements \( P_0, P \in G \) with \( P = k \cdot P_0 \) then it is impossible to compute \( k \) given only \( P_0 \) and \( P \). The reason for the name discrete logarithm stems from the fact that one notates the abstract group operation either additively, as before, or multiplicatively: \( P_0^k = P \). In the latter notation \( k \) is the logarithm of \( P \) to the base \( P_0 \).

We will not go into the details of the complete encryption and decryption process, suffice it to say that only data of the form \( kP \) goes over the wire where \( k \) is the secret information. Likewise the computation of \( kP \) is the most time-consuming one and therefore the crypto chip will implement the computation of \( kP \), given \( k \) and \( P \). The difficulty of finding \( k \) depends on the concrete finite group being used. In the original proposal the multiplicative group of the field \( F_p \) was used, however it is assumed that elliptic curve groups provide more security.

In the following section the basics of elliptic curves are introduced, together with the algorithm the chip implements. Previous implementations [7] use only 155 bits for the key, as opposed to our 261-bit key, and run at well under 100 MHz. We picked \( m = 261 \) as key size for the existence of an optimal normal basis and because from the theory this is more secure than RSA with 2048 bits.

2. Mathematics of Elliptic Curves and Algorithm

In the following we will only consider so-called non-supersingular elliptic curves as these provide the highest security. Non-supersingular elliptic curves are defined as the set of solutions of the cubic equation

\[
y^2 + xy = x^3 + ax^2 + b
\]  

(13)

together with the point at infinity \( \mathcal{O} \), where \( a, b \in F, b \neq 0 \) and \( F \) is any field. By defining an appropriate addition operation an elliptic curve becomes an abelian group. By varying the parameters \( a, b \) one has a source of finite abelian groups. Fig. 7 shows an example of an elliptic curve over the reals. Here a geometric interpretation of the addition can be given: the points \( P \) and \( Q \) are added in that the third intersection point with the curve of a straight line through them is mirrored at the \( x \) axis.

\[
y^2 = x^3 - 7x
\]

Fig. 7. Example of a supersingular elliptic curve over the real numbers with point addition.

For non-supersingular elliptic curves the basic operation of adding points \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \) is as follows:

\[
R = P + Q = (x_3, y_3)
\]  

(14)

if \( P \neq Q \) (addition):

\[
\phi = \frac{y_2 - y_1}{x_2 - x_1}
\]  

(15)

\[
x_3 = \phi^2 + \phi + x_1 + x_2 + a_2
\]  

(16)

\[
y_3 = \phi(x_1 + x_3) - y_1 + x_3
\]  

(17)

if \( P = Q \) (doubling):

\[
\phi = x + y
\]  

(18)

\[
x_3 = x^2 + \frac{b}{x_2}
\]  

(19)

\[
y_3 = x^2 + (\phi + 1)x_3
\]  

(20)

To avoid computing inverses, we change from affine to projective coordinates. If \( P = (x_1, y_1, z_1) \) and \( Q = (x_2, y_2, z_2) \) where \( P, Q \neq \mathcal{O} \) and \( P \neq -Q \) then \( R = P + Q = (x_3, y_3, z_3) \) is for \( P \neq Q \):

\[
x_3 = AD
\]  

(21)
\[ y_3 = CD + \lambda_i(Ax_1 + Ay_1) \quad (22) \]
\[ z_3 = A^3z_1z_2 \quad (23) \]
where \( A = x_2z_1 + x_1z_2, B = y_2z_1 + y_1z_2, C = A + B \) and
\[ D = A^2(A + az_1z_2) + z_1z_2BC. \]
We can fix \( z_2 \) to 1, as always
\[ P_0 = (x_0, y_0, 1) \]
will be added.
If \( P = Q \), then
\[ x_3 = AB \quad (24) \]
\[ y_3 = x_1^2A + B(x_1^2 + y_1z_1 + A) \quad (25) \]
\[ z_3 = A^3 \quad (26) \]
where \( A = x_1z_1, B = b_1^4 + x_1^4 \).

Thus, computing \( P + Q \) requires 13 field multiplications and 2\( P \) requires 7 multiplications. A full double and add requires 20 multiplications. As doubling is cheaper in terms of multiplication as addition, the use of projective coordinates benefits from a \( k \) with low hamming weight.

The multiplication of a elliptic curve point \( P_0 \) by some \( k > 1 \) is performed as repeated double and add of the base point \( P_0 \) using the above equations for \( x_3, y_3, z_3 \). To this end, addition and multiplication have to be done in the underlying field. In our case the field is \( F_{2^m} \) with \( m = 261 \). A field of characteristic two is used for a hardware implementation as addition reduces then to simply XOR-ing corresponding bits.

The elements of \( F_{2^{2m}} \) are represented in the so-called Optimal Normal Basis (ONB). In the following multiplication in ONB's is explained.

Let \( \alpha \) be an element of field \( F_{2^m} \), then the ONB can be formed as:
\[ \{ \alpha^{2^{m-1}}, \ldots, \alpha^2, \alpha, 1 \} \]
Any element in \( F_{2^m} \) can be represented in normal basis form as:
\[ A = a_{m-1}\alpha^{2^{m-1}} + \ldots + a_2\alpha^2 + a_1\alpha + a_0 \quad (27) \]
or
\[ A = \sum_{i=0}^{m-1} a_i\alpha^i \quad (28) \]

Squaring a number is just rotation of bits. \((\alpha^i)^2 = \alpha^{i+1}\) and from Fermat’s little theorem \( \alpha^{2^m} = \alpha \).

Multiplication of two numbers \( A, B \in F_{2^m} \) proceeds as follows:
\[ A = \sum_{i=0}^{m-1} a_i\alpha^i \]
\[ B = \sum_{j=0}^{m-1} b_j\alpha^j \]
\[ C = A \cdot B = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \alpha^{i+j} = \sum_{k=0}^{m-1} c_k\alpha^{2^k} \quad (31) \]
where
\[ \alpha^i \cdot \alpha^j = \sum_{k=0}^{m-1} a_i b_j \lambda_{ij} k \quad (32) \]
which can be modified to
\[ c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_{i+k} b_{j+k} \lambda_{ij} k \quad (33) \]
\[ ^2 \text{except for } i = 0, \text{ where only one value is obtained} \]

This requires calculation of only \( \lambda_{ij} \). An ONB has the minimum number of \( 2m - 1 \) non-zero terms in \( \lambda_{ij} \). This is also the number of the wiring connections from the registers to the multiplier circuit. We have
\[ \alpha^i \cdot \alpha^j = \alpha^{i+j} \text{ if } 2^i \neq 2^j \pmod{2m+1} \quad (34) \]
or\[ \alpha^i \cdot \alpha^j = \alpha^k \text{ if } 2^i \equiv 2^j \pmod{2m+1} \quad (35) \]
In case \( 2^i \neq 2^j \pmod{2m+1} \) at least one of the equations will have a solution
\[ 2^i + 2^j = 2^k \pmod{2m+1} \quad (36) \]
\[ 2^i + 2^j = -2^k \pmod{2m+1} \quad (37) \]
and at least one of the equations will have a solution
\[ 2^i - 2^j = 2^k \pmod{2m+1} \quad (38) \]
\[ 2^i - 2^j = -2^k \pmod{2m+1} \quad (39) \]
For the calculation of the \( \lambda \) matrix we set \( k = 0 \) and then find the solution to the following equations:
\[
\begin{array}{c|c|c}
 i & j_1 & j_2 \\
0 & 1 & - \\
1 & 0 & 36 \\
2 & 36 & 49 \\
3 & 128 & 72 \\
44 & 85 & 159 \\
5 & 104 & 45 \\
& & \\
260 & 260 & 35 \\
\end{array}
\]

For each \( i \) we get two \( ^2 \) values of \( j \) represented as \( j_1 \) and \( j_2 \).

For example the partial table for \( m = 261 \) is shown.

\[ c_k = \ldots \oplus b_{i+k}(a_{j_1+k} \oplus a_{j_2+k}) \oplus \ldots \quad (40) \]

where \( k = 0, \ldots, m - 1 \). For example, referring to the table for \( m = 261 \), we get:
\[ c_0 = a_0b_1 \oplus a_1(b_1 \oplus b_{36}) \oplus a_2(b_{36} \oplus b_{49}) \oplus \ldots \]
\[ c_1 = a_1b_2 \oplus a_2(b_2 \oplus b_{72}) \oplus a_3(b_{72} \oplus b_{128}) \oplus \ldots \]
\[ c_2 = a_2b_3 \oplus a_3(b_3 \oplus b_{159}) \oplus a_4(b_{159} \oplus b_{260}) \oplus \ldots \]
\[ c_{260} \]

This formula for calculating the \( c_k \)'s is quite regular and forms the basis for the application of the AWP. The multiplier is a large combinational net which in conventional architecutes computes the \( c_k \)'s one after another. Our main contribution here is to reduce the latency of the whole multiplication procedure by pipelining the computation of the \( c_k \)'s.
IV. Architecture and Implementation

The architecture of the chip is shown in Fig. 8. It consists of a register bank, an adder, and the multiplier. The upper part of the register bank consists of static registers which are serially loaded from off-chip. Registers X0 and Y0 hold the base point \( P_0 = (x_0, y_0) \). Registers a, b, and k contain the elliptic curve parameters and the multiplier, respectively. For the temporary values A, B, and D introduced above are registers allocated. For variable C we have not allocated a register as it stores only a sum, whereas an addition takes only one cycle. Finally, the result point is accumulated in projective coordinates in registers X, Y, and Z. All connections have width \( m = 261 \) if not labelled otherwise.

Fig. 8. Organization of the elliptic curve crypto chip.

The operation is illustrated in Fig. 9. Once all parameters are loaded, top level control implements the double-and-add algorithm for computing \( k \cdot P_0 \) by shifting register k left \( m \) times. At each step, an elliptic curve double point operation with \( (x, y, z) \) is performed and, if the leftmost bit of \( k \) is one, an elliptic curve add point operation \( (x, y, z) + (x_0, y_0, 1) \) follows. After all bits of \( k \) have been processed, the result \( k \cdot P_0 \) is in registers X, Y, and Z and can be serially downloaded from the chip.

Fig. 9. Control hierarchy for the crypto chip with different frequency domains.

Fig. 10. Top level control.

Fig. 11. Waveforms of top level control.
is used. The signal X is designed so that it has a transition in autonomous states and a pulse is derived from it (signal X!).

At the mid level of control, we have to compute doubling and adding of points on the elliptic curve. This in turn is achieved by sequentially computing the formulae given in section III.2 using the finite field base arithmetic at the lowest level. The operation is detailed in Fig. 14 and Fig. 13. Here the control logic is pulsed and has X equal to one in autonomous states and zero else.

![Detailed architecture of the multiplier](image)

Fig. 12. Detailed architecture of the multiplier.

![Waveforms of double control](image)

Fig. 13. Waveforms of double control.

An oscillator serves as a request source. The operand registers opA and opB are synchronous registers and clocked by the request. The addition operation is only a parallel XOR of corresponding bits. For multiplication, the operand registers are rotated once and then parallel loaded into the multiplier. The multiplier is an AWP, and therefore the request goes into the delay element. The multiplier core accepts \(2 \cdot m - 1 = 521\) inputs from opA and opB and reduces them in a huge XOR-tree with initial AND-XOR-stages according to equation (40) and Fig. 12 into one bit which goes serially into the result register out. The opB bits have a fanout of two at the multiplier input corresponding to the \(j_1\) and \(j_2\) values in III.2 so that the widest slice of the data path is \(3 \cdot m = 783\) bits wide. The index shuffling in the computation of the \(c_k\)'s in (40) results in wiring which is shown in Fig. 8 as a shaded area. A counter controls that all \(c_0\) to \(c_{m-1}\) are computed, after each step rotating the input registers.

![Initial AND-XOR stage of the multiplier](image)

Fig. 15. Initial AND-XOR stage of the multiplier (abx in Fig. 12).

The layout of the multiplier as in Fig. 16 is organized in three slices each having \(m\) inputs. The input is coming as levels from the registers and goes into a pseudo-NMOS stage, cf. Fig. 12 and Fig. 15. Pseudo-NMOS was tried to save devices over SRCMOS as the data path has width \(3 \cdot m = 783\)
here, but one could use as well SRCMOS throughout. Following the initial stage, the data wave is synchronized with the request in the wave latch stage in Fig. 17 and converted to pulses. Thereafter, the 87 outputs go into a tree of 3-XORs as in Fig. 18.

Fig. 19 shows some signals from layout simulation in closeup and demonstrates that the SRCMOS gate delays are very well balanced. The cycle time is 666 ps, which translates into 1.5 GHz throughput, as can be seen in Fig. 19.

Fig. 20 shows the latency of the multiplier core from inputs to the inputs of the last 3-XOR to be around 2 ns. Therefore, the number of waves active in the logic is 2 ns / 0.666 ns ≈ 3.

![Fig. 16. Layout of the multiplier measuring 0.85 by 0.69 sqmm and having 783 inputs and 1 output. As shown in the inset, some 300 µm long wires need repeaters to compensate wire delay variation.](image1)

![Fig. 17. Wave latch converting levels into pulses.](image2)

![Fig. 18. 3-XOR used in the reduction tree.](image3)
V. Conclusion

This paper has shown the application of AWP to a crypto chip. The critical part shows throughput of 1.5 GHz from layout. The chip has a hierarchical control structure comprising three layers resulting in different frequency domains. Synchronous registers and asynchronous controllers are used together with the AWP multiplier.

AWP has enabled fine-grain pipelining of a wide data path without adding much to the latency. In fact, throughput is determined only by the width of the pulses that the logic can propagate. From the description of the multiplier it has become clear that conventional pipelining techniques would have been difficult to use here.

The layout of the complete processor is under work. We expect to have first silicon in summer.

References


