HIGH LEVEL SYNTHESIS TO REDUCE ENERGY CONSUMPTION OF SIGNAL AND IMAGE PROCESSING OPERATORS

H. Ye\textsuperscript{1}, L. Lacassagne\textsuperscript{2}, J. Falcou\textsuperscript{2}, D. Etiemble\textsuperscript{2}, L. Cabaret\textsuperscript{3} and O. Florent\textsuperscript{1}

\textsuperscript{1} ST Microelectronics, F-38019 Grenoble, France
\textsuperscript{2}LRI: Laboratoire de Recherche en Informatique, Univ. Paris-Sud, F-91405 Orsay, France
\textsuperscript{3}ECP/LISA: Ecole Centrale de Paris, F-92295 Châtenay-Malabry, France

ABSTRACT

High Level Synthesis for Systems on Chip is a challenging way to cut off development time, while assuming a good level of performance. But the HLS tools are limited by the abstraction level of the description to perform some high level transforms. This paper evaluates the impact of such high level transforms for ASICs. We have evaluated recursive and non recursive filters for signal processing an morphological filters for image processing. We show that the impact of HLTs to reduce energy consumption is high: from $\times 3.4$ for one 1D filter up to $\times 5.6$ for cascaded 1D filters and about $\times 3.5$ for morphological 2D filters.

Index Terms— High Level Synthesis, High Level Transforms, algorithm transforms, software optimizations, ASIC, power consumption, energy optimization, signal processing, image processing.

1. INTRODUCTION

High Level Synthesis (HLS) for Systems on Chip is a challenging way to cut off development time while assuming a good level of performance. The latest version of HLS tools integrates software optimizations from the optimizing compiler area \textsuperscript{1}like loop-unrolling, software pipelining and using the polyhedral model to improve loop scheduling. To further improve current performance, tools should integrate the semantic of an application domain \textsuperscript{2}and the related algorithm transforms \textsuperscript{3}.

This paper evaluates the impact of such algorithm transforms or high level transforms (HLT) for ASIC. More and more commercial or academic HLS tools are available like LegUp \textsuperscript{4}or Gaut \textsuperscript{5}. We have chosen Catapult-C as it is the tool used by ST Microelectronics for its synthesis farm. Finite Impulse Response and Infinite Impulse Response filters have been chosen as being ubiquitous in signal processing, while morphological filters are also largely used in the image processing area. The first section presents Catapult-C and how to explore the design space configurations. The next sections describe a set of optimizations and their impact on FIR filters, IIR filters and morphological filters as well implementation details on the code generation process based on preprocessor meta-programming.

2. HIGH LEVEL SYNTHESIS TOOLS AND OPTIMIZATIONS

The optimizations can be classified according to three categories: HLT, software optimizations and hardware optimizations. HLT are algorithmic transforms based on optimizations belonging to an application domain – like image and signal processing – and are related to operator properties to reduce the algorithm complexity. Examples of such transforms are filter separation and factorization. The usual software optimizations like loop unrolling, register rotation and software pipelining are present in Catapult-C: it is able to unroll or pipeline loops. Moreover, it is able to detect a loop that perform register rotation and transform the loop into a set of register-to-register copy within a register file.

The typical way of using such a tool is to provide a C or C++ source code and to set the clock frequency to be used. If almost all parameters are automatically explored by the tool, at least one parameter can be set by the user: the initiation interval (ii). That is the number of clock cycles between the start of two iterations of a loop. Let us consider the following computation $t = a + b + c + d$ (from \textsuperscript{6}) with usual 2-input adders and assume that the duration of one addition is one cycle. In the first case, one wants one output written every 3 cycles (Fig. 1, top with ii=3). In that case, there is no overlap between the operations and only one adder is needed. To get one output every 2 cycles (Fig. 1, middle with ii=2), two adders are needed as two additions occur on cycle 3: the first one computes $t_3 = t_2 + d$ from the first lane, and the second computes $t_1 = a + b$ from the second lane. To get one output every cycle (Fig. 1, bottom with ii=1, three adders are needed on cycle 3: the first one computes $t_3 = t_2 + d$, the second one computes $t_2 = t_1 + c$ and the third one computes $t_1 = a + b$.

Notice that the electronic instance of software pipelining, which is the most important optimization for VLIW processors. Depending on ii and the algorithm structure, one can have a direct impact on the size and performance of the...
circuit: with smaller \( ii \), the circuit is faster and bigger; with larger \( ii \), the circuit is slower and smaller.

For benchmarking, the ST 65-nm CMOS library was used with Catapult-C. The evaluation of the power consumption and the area was done with Synopsys Design Compiler – before place and route – without activating the capabilities of Catapult-C to reduce the total power consumption generating local/global clock gating glue as we assume that the ASIC is always running. In that case, the energy is the product of the execution time by the (static + dynamic) power.

In the following, the small internal arrays are stored into register, while big external arrays are stored into memory. There are one memory per array, except for banked-memory where there are 3 (or 5) memory bank to enable multiple accesses. We assume a streaming behavior: a set of data is transferred into memory before an operator is called to process it. As operators are pipelined, the latency is equal to the value of \( ii \).

3. FINITE IMPULSE RESPONSE FILTER

The Finite Impulse Response (FIR) filters are very common in signal processing. If data are represented by floating-point numbers, there are no special issue to address and the implementation is straightforward. The only issues are absorption and cancellation when the coefficients magnitude is very high. In our case, we consider integer data - typically - 8-bit and \( Q_8 \) computations. That implies three points: 1) the coefficients are multiplied by \( 2^8 \) before rounding, 2) the result of the computation is divided by \( 2^8 \) – shifted by 8 – and 3) in order to have rounded computations instead of truncated ones, we must add half the value of the division, that is \( 2^7 \). So considering a general 3-tap filter (Eq. 1), the associated implementation is given by the algorithm 1 with the rounding value \( r = 128 \). Notice that in all our examples, we do not provide the prolog or the epilog of a loop, in order to have small and compact examples (The Duff’s device [7] can be used to remove the epilog but leads to a more tricky code).

\[
y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) \quad (1)
\]

Algorithm 1: fixed-point FIR3 filter – Reg version

```plaintext
1. for \( i = 2 \) to \( n - 1 \) do
2. \( x_0 \leftarrow X[i-0], x_1 \leftarrow X[i-1], x_2 \leftarrow X[i-2] \)
3. \( y \leftarrow (b_0 \times x_0 + b_1 \times x_1 + b_2 \times x_2 + r)/256 \)
4. \( Y[i] \leftarrow y \)
```

3.1. FIR optimization

For FIR filters, there are two points to address. The first one is to compare the impact of the hardware and software optimizations on the synthesis of one filter. The second one is the optimization of cascaded FIR filters.

The usual drawback of such a filter is its low complexity: there is one MAC (Multiply-Accumulate) operation for every LOAD. So the FIRs are generally memory bounded (except on VLIW DSPs that are specially designed for signal processing [8] and are able to perform two memory accesses per cycle). So, optimizing means reducing the LOAD duration (the HW optimization) or the number of LOADs (the SW one). We have evaluated four kinds of memory: 1) the Single Port memory with 1 READ or 1 WRITE per cycle, 2) the SP RW memory with 1 READ and 1 WRITE per cycle, 3) the Double Port memory: two SP memories, with 2 READs per cycle and 4) the banked memory with 3 (or 5) interleaved SP memory banks, allowing 3 (or 5) accesses per cycle. A small finite state machine automaton selecting the correct memory bank during the filtering has been designed to handle circular addressing. For the software part, we can perform Register Rotation or Loop Unrolling (Algo. 2 and 3). We omit the prolog and epilog parts to keep the algorithms as small as possible.

Algorithm 2: FIR3 filter – Rot version

```plaintext
1. \( x_2 \leftarrow X[0], x_1 \leftarrow X[1] \)
2. for \( i = 2 \) to \( n - 1 \) do
3. \( x_0 \leftarrow X[i-0] \)
4. \( y \leftarrow (b_0 \times x_0 + b_1 \times x_1 + b_2 \times x_2 + r)/256 \)
5. \( Y[i] \leftarrow y \)
6. \( x_2 \leftarrow x_1, x_1 \leftarrow x_0 \) [Registers Rotation]
```
3.2. Results and analysis

Let us call best A, the configuration that minimizes the areas and best E the configuration that minimizes the energy.

The first part of the table 1 provides the average area and energy of all the best configurations for synthesis frequency varying from 200 to 800 MHz by step of 200 MHz.

As area and energy figures are very close for a given ii, they have been replaced by their average in order to provide all the results within only one table. The table also provides the initiation interval (ii). The best HW optimization is the 3 × SP version with a banked memory. The energy is divided by ×3.43 versus the basic SP version and the area increases by 78 %. The best SW optimization is SP+Rot: the energy is divided by ×5.87 with only a 19% area increase. The reason why these two configurations are by far the most efficient is that a computation can be launched every cycle (ii=1). Notice that Loop-Unrolling is not efficient with Catapult-C: the DP+LU consumes more energy than DP+Reg. That is the reason why we did not evaluate the 3 × SP+LU version.

If we now focus on the efficiency of two cascaded filters, there are three cases:
1. two independent filters (Algo. 4), with a temporary memory T of the same size than the input,
2. two pipelined filters with a 1-point FIFO for the Rot version and a 3-point FIFO for the Reg version (Algo. 5),
3. one single filter that is the fusion of the previous filters.

From a software point of view, the pipelined version corresponds to a loop-fusion but with two separates filters, while the fused version correspond to the filter fusion.

The table 1 summarizes all the results. Notice that in the case of the cascaded filters, the area does not consider the temporary memory area and its power consumption. As a matter of fact, a 65-nm 1024 × 8-bit SP memory has a power consumption of 14 pJ/point with an area close to 15000 µm². So the cascaded filters results are just provided as the

<table>
<thead>
<tr>
<th>memory optimization</th>
<th>SP Reg</th>
<th>SR RW Reg</th>
<th>DP Reg</th>
<th>3×SP Reg</th>
<th>SP Rot</th>
<th>SP LU</th>
<th>SP RW LU</th>
<th>DP LU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1× FIR3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>area(best A)</td>
<td>4237</td>
<td>4266</td>
<td>4562</td>
<td>4949</td>
<td>4458</td>
<td>6120</td>
<td>6240</td>
<td>7027</td>
</tr>
<tr>
<td>area(best E)</td>
<td>4671</td>
<td>7547</td>
<td>5957</td>
<td>7357</td>
<td>5047</td>
<td>10602</td>
<td>11083</td>
<td>12594</td>
</tr>
<tr>
<td>energy(best A)</td>
<td>11.68</td>
<td>12.08</td>
<td>12.54</td>
<td>13.80</td>
<td>11.92</td>
<td>28.24</td>
<td>28.49</td>
<td>26.18</td>
</tr>
<tr>
<td>s(i)(best E)</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

| two cascaded FIR3    |        |           |        |          |        |       |          |       |
| area(best A)         | 6393   | 6842      | 6542   | 7498     | 6574   | 9913  | 10155    | 11155 |
| area(best E)         | 8781   | 9206      | 10797  | 10360    | 7370   | 15082 | 16895    | 16273 |
| energy(best A)       | 12.88  | 13.11     | 13.95  | 12.86    | 8.79   | 24.18 | 25.05    | 19.93 |
| energy(best E)       | 7.56   | 5.87      | 4.34   | 6.69     | 2.82   | 15.79 | 16.45    | 10.89 |
| s(i)(best E)         | 3      | 3         | 2      | 1        | 1      | 3     | 3        | 2     |

| two pipelined FIR3   |        |           |        |          |        |       |          |       |
| area(best A)         | 5888   | 5943      | 6207   | 6453     | 5715   | 9625  | 10039    | 10400 |
| area(best E)         | 7619   | 7547      | 8639   | 12385    | 9317   | 18329 | 19086    | 20726 |
| energy(best A)       | 20.05  | 21.78     | 21.33  | 23.28    | 23.11  | 64.54 | 65.56    | 66.98 |
| energy(best E)       | 9.98   | 10.03     | 8.11   | 5.55     | 3.65   | 21.31 | 22.08    | 17.42 |
| s(i)(best E)         | 3      | 3         | 2      | 1        | 1      | 3     | 3        | 2     |

<table>
<thead>
<tr>
<th>memory optimization</th>
<th>SP Reg</th>
<th>SR RW Reg</th>
<th>DP Reg</th>
<th>5×SP Reg</th>
<th>SP Rot</th>
<th>SP LU</th>
<th>SP RW LU</th>
<th>DP LU</th>
</tr>
</thead>
<tbody>
<tr>
<td>two fused FIR3 = one FIR5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>area(best A)</td>
<td>5563</td>
<td>5619</td>
<td>5683</td>
<td>7121</td>
<td>5990</td>
<td>17968</td>
<td>17563</td>
<td>17330</td>
</tr>
<tr>
<td>area(best E)</td>
<td>6056</td>
<td>6198</td>
<td>7670</td>
<td>12513</td>
<td>8189</td>
<td>26913</td>
<td>28100</td>
<td>30441</td>
</tr>
<tr>
<td>energy(best A)</td>
<td>22.11</td>
<td>22.72</td>
<td>19.89</td>
<td>27.23</td>
<td>22.01</td>
<td>107.03</td>
<td>107.13</td>
<td>118.48</td>
</tr>
<tr>
<td>energy(best E)</td>
<td>13.74</td>
<td>14.05</td>
<td>10.52</td>
<td>5.59</td>
<td>3.19</td>
<td>48.21</td>
<td>50.98</td>
<td>34.14</td>
</tr>
<tr>
<td>s(i)(best E)</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1. FIR average area and energy

Algorithm 3: fixed-point FIR3 filter – LU version

1  \( x_2 \leftarrow X[0], x_1 \leftarrow X[1] \)
2  for \( i = 2 \) to \( n - 1 \) do
3      \( x_0 \leftarrow X[i - 0] \)
4      \( y_0 \leftarrow (b_0 \times x_0 + b_1 \times x_1 + b_2 \times x_2 + \tau)/256 \)
5      \( Y[i + 0] \leftarrow y_0 \)
6      \( x_2 \leftarrow X[i + 1] \)
7      \( y_1 \leftarrow (b_0 \times x_2 + b_1 \times x_0 + b_2 \times x_1 + \tau)/256 \)
8      \( Y[i + 1] \leftarrow y_1 \)
9      \( x_1 \leftarrow X[i + 2] \)
10     \( y_2 \leftarrow (b_0 \times x_1 + b_1 \times x_2 + b_2 \times x_0 + \tau)/256 \)
11     \( Y[i + 2] \leftarrow y_2 \)
first step of the optimization process, keeping in mind that that a designer will start with the pipelined version with a small FIFO. Compared to SP, the RW (respectively DP) memory has a surface and a power consumption that are \( \times 1.5 \) and \( \times 1.3 \) (respectively \( \times 1.8 \) and \( \times 1.6 \)) bigger than SP ones.

Regarding the pipelined filter configurations, the energy of the \( \text{Rot} \) version is \( \times 5.49 \) smaller than the energy of \( \text{Reg} \) version. The figure is even higher for the fused filters configuration: \( 3.19 \) \( \mu \text{J} \), that is \( \times 6.93 \) less than the \( \text{SP+Reg} \) version, while the area is quite the same (0.45% smaller). Again, only \( 3 \times \text{SP+Reg} \) and \( \text{SP+Rot} \) versions lead to an \textit{initiation interval} of 1 cycle per point.

---

Algorithm 4: 2 FIR3 filters, with temporary memory \( T \)

1. \textbf{for} \( i = 0 \) to \( n - 1 \) \textbf{do}
2. \hspace{1em} \( x \leftarrow X[i] \), \( y_1 \leftarrow F_1(x) \), \( T[i] \leftarrow y_1 \)
3. \textbf{for} \( i = 0 \) to \( n - 1 \) \textbf{do}
4. \hspace{1em} \( x \leftarrow T[i] \), \( y_2 \leftarrow F_2(x) \), \( Y[i] \leftarrow y_2 \)

Algorithm 5: two pipelined FIR3 filters

1. \textbf{for} \( i = 0 \) to \( n - 1 \) \textbf{do}
2. \hspace{1em} \( x \leftarrow X[i] \), \( y_1 \leftarrow F_1(x) \), \( y_2 \leftarrow F_2(y_1) \), \( Y[i] \leftarrow y_2 \)

Algorithm 6: two fused FIR3 filters

1. \textbf{for} \( i = 0 \) to \( n - 1 \) \textbf{do}
2. \hspace{1em} \( x \leftarrow X[i] \), \( y \leftarrow F_2(F_1(x)) \), \( Y[i] \leftarrow y \)

3.3. Meta-programming

In order to provide a high level interface for these transforms, we implemented an IDL (\textit{Interface Description Language}) like interface for the FIR function definitions using preprocessor meta-programming. Meta-programming is a set of software techniques that bring the benefits of automation to software developments. Those techniques rely on various languages specific features that enable developers to manipulate programs fragments as data at various stage of the compilation process. If the most famous meta-programming technique is template meta-programming (as defined in [9]), preprocessor or macro-based meta-programming fills a very important niche. Based on the seminal work [10], preprocessor meta-programming allows the developers to manipulate so-called preprocessor data structure and preprocessor control flows to generates arbitrarily complex codes for function interfaces or repetitive, token based iterations. Such structures include arrays, tuples and sequences. Control flows can be defined as preprocessor loops or iterations over structures. The advantages of such techniques over handwritten, direct macro expansion is the higher level of abstractions and the facility to compose macro generators. Libraries like Boost.ConceptCheck [11] or Boost.Local use such facilities to emulates languages features with a high degree of interface compliance.

In our case, preprocessor meta-programming was used to be able to comply with Catapult-C support of current C and C++ standards. As conformance to the C preprocessor standard is fairly common among existing tools, this strategy is portable across various tools outside of Catapult-C. Template meta-programming has been checked but was not completely supported by the Catapult-C compiler. In the following listings (Lst. 1 & 2), \texttt{ac\_channel} is a Catapult-C C++ class for the stream I/O and \texttt{array} stands for a C99 variable length array.

Listing 1. FIR3 with C99 array

\begin{verbatim}
FIR(((array(sint8,N),X),(array(sint8,M),H)),
(array(sint8,N),Y))
\end{verbatim}

Listing 2. FIR3 with \texttt{ac\_channel} I/O

\begin{verbatim}
FIR(((\texttt{ac\_channel}(uint8),X),(array(sint8,M),H)),
(\texttt{ac\_channel}(uint8),Y)))
\end{verbatim}

The macros expand in the following stages: a declaration of variables and a static array RF, a loop containing the load of a point (from an C99 array of from a stream \texttt{ac\_channel}), then a \texttt{register-rotation} RF with the memorization of the last input inside, the convolution and the output. We have observed that Catapult-C exactly generates a Register File with a rotation for the first loop.

---

**Fig. 2.** producer-consumer model of FIR3 \texttt{Reg} & \texttt{Rot} versions

**Fig. 3.** three versions of two cascaded FIR3: independent filters with temporary \( T \) memory (top), pipelined filters with a 1-point FIFO (middle), fused filters (bottom)
4. INFINITE IMPULSE RESPONSE FILTER

The recursive filters (Eq. 2) are difficult to optimize by a compiler. The reason is the dependency between the current output \(y(n)\) and the previous outputs that are also the current inputs \(y(n-1)\) and \(y(n-2)\). Let us focus on the IIR12 filters that are used for edge detection. This filter (Eq. 2) is a smoother filter obtained after a simplification [12, 8] of the well-known Canny-Deriche filter [13, 14]

\[
y(n) = (1-\gamma^2) x(n) + 2\gamma y(n-1) - \gamma^2 y(n-2)
\]

Algorithm 7: IIR12 filter – Normal form

1. for \(i = 2\) to \(n - 1\) do
2. \(x_0 \leftarrow X[i-0], y_1 \leftarrow Y[i-1], y_2 \leftarrow Y[i-2]\)
3. \(y_0 \leftarrow (b_0 \times x_0 + b_1 \times y_1 + a_2 \times y_2 + r) / 256\)
4. \(Y[i] \leftarrow y_0\)

4.1. IIR optimization

Thanks to the coefficient expression, the Factor form (Eq. 3) is the factorization of the previous filter with powers of \(\gamma\). It replaces one multiplication by two additions. Depending on their respective size / power consumption and latency, it could lead to smaller / lower power or faster designs.

\[
y(n) = x(n) + 2\gamma [y(n-1) - x(n)] - \gamma^2 [y(n-2) - x(n)]
\]

Algorithm 8: IIR12 filter – Factor form

1. for \(i = 2\) to \(n - 1\) do
2. \(x_0 \leftarrow X[i-0], y_1 \leftarrow Y[i-1], y_2 \leftarrow Y[i-2]\)
3. \(y_0 \leftarrow (256 \times x_0 + a_1 (y_1 - x_0) + a_2 (y_2 - x_0) + r) / 256\)
4. \(Y[i] \leftarrow y_0\)

Concerning the dependency problem, the shortest and strongest dependency \(y(n-1)\) can be removed by replacing the expression of \(y(n-1)\) into \(y(n-2)\) (Eq. 4). This form is called Delay as the dependency is delayed to \(y(n-2)\) and \(y(n-3)\).

\[
y(n) = (1-\gamma^2) x(n) + 2\gamma (1-\gamma^2) x(n-1) + 3\gamma^2 y(n-2) - 2\gamma^3 y(n-3)
\]

Algorithm 9: IIR23 filter – Delay form

1. for \(i = 2\) to \(n - 1\) do
2. \(x_0 \leftarrow X[i-0], x_1 \leftarrow X[i-1]\)
3. \(y_2 \leftarrow Y[i-2], y_3 \leftarrow Y[i-3]\)
4. \(y_0 \leftarrow (b_0 \times x_0 + b_1 \times x_1 + a_2 \times y_2 + a_3 \times y_3 + r) / 256\)
5. \(Y[i] \leftarrow y_0\)

Table 2. IIR filters complexity

```
<table>
<thead>
<tr>
<th>version</th>
<th>MUL</th>
<th>ADD</th>
<th>SHIFT</th>
<th>LOAD</th>
<th>STORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal form</td>
<td>3</td>
<td>2+1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Factor form</td>
<td>2</td>
<td>4+1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Delay form</td>
<td>4</td>
<td>3+1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
```

4.2. IIR Results and analysis

Table 3 presents the results of the three forms in term of area, power and energy. We can observe that energy consumption increases with \(ii\), while area and power decrease. The reason is that Catapult-C optimizes for area and power, so the smallest area (best \(A\) configuration) and lowest power is obtained when \(ii\) is not constrained (noted auto \(ii\)). In that case, Catapult-C chooses \(ii \in \{5,6,7\}\).

The smallest energy consumption is again obtained for \(ii = 1\). If we compare Normal and Factor forms, \(ii = 1\) and/or Factor provide energy reductions of 753/179 = \(\times 4.2\) at 200 MHz and 1171/211 = \(\times 5.6\) at 400 MHz. If we
### Table 3. IIR area, power and energy for synthesis frequency in [200..800] with 200 MHz step

<table>
<thead>
<tr>
<th>Freq (MHz)</th>
<th>Normal form</th>
<th>Factor form</th>
<th>Delay form</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>3780</td>
<td>3931</td>
<td>4469</td>
</tr>
<tr>
<td>400</td>
<td>3635</td>
<td>3984</td>
<td>4399</td>
</tr>
<tr>
<td>600</td>
<td>5517</td>
<td>6769</td>
<td>5517</td>
</tr>
<tr>
<td>800</td>
<td>9963</td>
<td>7817</td>
<td>5517</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Area (μm²)</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
<th>auto i</th>
<th>/best A</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 1</td>
<td>352</td>
<td>583</td>
<td>352</td>
<td>583</td>
<td>352</td>
<td>583</td>
<td>352</td>
<td>583</td>
<td>352</td>
<td>583</td>
<td>352</td>
<td>583</td>
<td>352</td>
<td>583</td>
<td>352</td>
<td>583</td>
<td>352</td>
<td>583</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i = 2</td>
<td>613</td>
<td>710</td>
<td>613</td>
<td>710</td>
<td>613</td>
<td>710</td>
<td>613</td>
<td>710</td>
<td>613</td>
<td>710</td>
<td>613</td>
<td>710</td>
<td>613</td>
<td>710</td>
<td>613</td>
<td>710</td>
<td>613</td>
<td>710</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i = 3</td>
<td>325</td>
<td>551</td>
<td>325</td>
<td>551</td>
<td>325</td>
<td>551</td>
<td>325</td>
<td>551</td>
<td>325</td>
<td>551</td>
<td>325</td>
<td>551</td>
<td>325</td>
<td>551</td>
<td>325</td>
<td>551</td>
<td>325</td>
<td>551</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i = 4</td>
<td>583</td>
<td>1085</td>
<td>583</td>
<td>1085</td>
<td>583</td>
<td>1085</td>
<td>583</td>
<td>1085</td>
<td>583</td>
<td>1085</td>
<td>583</td>
<td>1085</td>
<td>583</td>
<td>1085</td>
<td>583</td>
<td>1085</td>
<td>583</td>
<td>1085</td>
<td></td>
<td></td>
</tr>
<tr>
<td>best E</td>
<td>1×40</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td>1×10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>best A</td>
<td>1×138</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td>1×122</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Algorithm 10: 3 × 3 morphological filter (Eq.5) - Reg version

1. for \( i = 1 \) to \( n - 1 \) do
2.   for \( j = 1 \) to \( n - 1 \) do
3.     \( a_0 \leftarrow X[i-1][j-1], a_1 \leftarrow X[i-1][j], a_2 \leftarrow X[i][j+1], \)
4.     \( Y[i][j] \leftarrow r \)
5.     \( r \leftarrow a_0 \oplus a_1 \oplus c_1 \)

\( r \) the radius of the kernel: for a \( k \times k \) kernel, \( k = 2r + 1 \).

### 5. MORPHOLOGICAL FILTERING

The morphological operators [15] exist in two flavors: the binary ones and the grey level ones. In the following, we assume binary ones with a \( 3 \times 3 \) structuring element. The filtering operators like opening, closing and alternating sequential filters are all based on two basic operators: the erosion and the dilation. These two morphological operators rely on the use of \( \min \) and \( \max \) computations over the structuring element for gray images. These operators are respectively replaced by the boolean operators AND and OR for binary images. As they are all idempotent, let us define the \( \oplus \) operator that is one of the four basic operators. The basic implementation of such an operator is given in algorithm 10.

The usual problem to process the array edges is addressed by the use of Iflihe arrays [16] based on offset addressing that allows the programmer to allocate images with negative indexes like \([0 - r : (n - 1) + r] \times [0 - r : (n - 1) + r] \), with

\[
SE_{3 \times 3} = \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix} \times \begin{bmatrix}
1 \\
1 \\
1 \\
\end{bmatrix} = \begin{bmatrix}
1 \\
1 \\
1 \\
\end{bmatrix} \times [1 1 1] (5)
\]
Algorithm 11: 1-pass implementation of the $3 \times 3$ morphological filter with Register Rotation, Rot version

1 for $i = 1$ to $n - 1$
2 \hspace{1em} $j \leftarrow 1$ [preload the first two columns of each line]
3 \hspace{1em} $a_0 \leftarrow X(i - 1, j - 1), b_0 \leftarrow X(i - 1, j)$
4 \hspace{1em} $a_1 \leftarrow X(i, j - 1), b_1 \leftarrow X(i, j)$
5 \hspace{1em} $a_2 \leftarrow X(i + 1, j - 1), b_2 \leftarrow X(i + 1, j)$
6 for $j = 1$ to $n - 1$ step 3 do
7 \hspace{2em} $c_0 \leftarrow X(i - 1, j + 1)$
8 \hspace{2em} $c_1 \leftarrow X(i, j + 1)$
9 \hspace{2em} $c_2 \leftarrow X(i + 1, j + 1)$
10 \hspace{2em} $r \leftarrow a_0 \oplus b_0 \oplus c_0 \oplus a_1 \oplus b_1 \oplus c_1 \oplus a_2 \oplus b_2 \oplus c_2$
11 \hspace{2em} $Y(i, j) \leftarrow r$
12 \hspace{2em} $a_0 \leftarrow b_0, b_0 \leftarrow c_0$ [RR of the first line]
13 \hspace{2em} $a_1 \leftarrow b_1, b_1 \leftarrow c_1$ [RR of the second line]
14 \hspace{2em} $a_2 \leftarrow b_2, b_2 \leftarrow c_2$ [RR of the third line]

By introducing another optimization, we can both factorize the computations and reduce the number of memory accesses. The two passes of the 1D-filter on the image can be combined within a single pass. First, the result of the first 1D-filter is stored in a register. This transformation is called a reduction. In our case, it is a column-wise reduction: instead of memorizing 6 pixels (Algo. 11, lines 4-6), we compute the reduced value by column (Algo. 12, lines 5 & 6). Then the second operator is directly applied to the reduced values (Algo. 12, line 12). In that version, there are only 3 LOADs and 4 OPs. All the complexity figures are summarized in table 4 where MV stands for a move (to copy one register to another one) and AI represents the arithmetic intensity (ratio between arithmetic operators and memory accesses).

Algorithm 12: 1-pass implementation of the two separated 1D operators with reduction, Red version

1 for $i = 1$ to $n - 1$
2 \hspace{1em} $a_0 \leftarrow X(i - 1, j - 1), b_0 \leftarrow X(i - 1, j)$
3 \hspace{1em} $a_1 \leftarrow X(i, j - 1), b_1 \leftarrow X(i, j)$
4 \hspace{1em} $a_2 \leftarrow X(i + 1, j - 1), b_2 \leftarrow X(i + 1, j)$
5 \hspace{1em} $r_a \leftarrow a_0 \oplus a_1 \oplus a_2$ [reduction of the first column]
6 \hspace{1em} $r_b \leftarrow b_0 \oplus b_1 \oplus b_2$ [reduction of the second column]
7 for $j = 1$ to $n - 1$ do
8 \hspace{2em} $c_0 \leftarrow X(i - 1, j + 1)$
9 \hspace{2em} $c_1 \leftarrow X(i, j + 1)$
10 \hspace{2em} $c_2 \leftarrow X(i + 1, j + 1)$
11 \hspace{2em} $r_c \leftarrow c_0 \oplus c_1 \oplus c_2$ [reduction of the third column]
12 \hspace{2em} $r \leftarrow r_a \oplus r_b \oplus r_c$ [applying the horizontal operator]
13 \hspace{2em} $Y(i, j + 0) \leftarrow r$
14 \hspace{2em} $r_a \leftarrow r_b$ [rotation of the reduced registers]
15 \hspace{2em} $r_b \leftarrow r_c$

<table>
<thead>
<tr>
<th>version</th>
<th>OP</th>
<th>LD + ST</th>
<th>MV</th>
<th>AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg (1-pass of 2D-op)</td>
<td>8</td>
<td>9 + 1 = 10</td>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>Rot (1-pass of 2D-op)</td>
<td>8</td>
<td>3 + 1 = 4</td>
<td>6</td>
<td>2.0</td>
</tr>
<tr>
<td>Red (1-pass of 1D-op)</td>
<td>4</td>
<td>3 + 1 = 4</td>
<td>2</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 4. Morphological operator complexity and arithmetic intensity

5.2. Results and analysis

For the morphological operators, HLTs have a major impact on the efficiency. Let us call "bestE" and "bestA" the configurations associated to the smallest energy consumption, and the smallest area (Tab. 5 & 6). Let us also call auto $ii$ the combinatorial version. As the basic version (Reg) requires 9 LOADs (Tab. 5), we need 9 cycles to perform all the LOADs with a single-port RAM and $[9/2] = 5$ cycles with a dual-port RAM. For the same reason, the minimum number of cycles for Rot and Red versions (3 LOADs) is 2 cycles. That is very important, as for all the explored configurations, the best $E$ was reached for the smallest $ii$. The gap due to HLTs ranges from $2.85$ to $3.05$. Moreover, with a single-port RAM, the gap between Reg and Red versions would be even greater, as the energy increases with the $ii$. Finally, if we compare the configuration of the smallest area without HLT to the best Red, the gain ranges from $3.3$ to $3.8$.

<table>
<thead>
<tr>
<th>freq (MHz)</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>bestA Reg (auto $ii$)</td>
<td>6.45</td>
<td>6.67</td>
<td>7.44</td>
<td>7.79</td>
</tr>
<tr>
<td>bestE Reg (ii=5)</td>
<td>5.49</td>
<td>5.76</td>
<td>6.44</td>
<td>5.87</td>
</tr>
<tr>
<td>bestE Rot (ii=2)</td>
<td>2.47</td>
<td>2.78</td>
<td>3.14</td>
<td>2.95</td>
</tr>
<tr>
<td>bestE Red (ii=2)</td>
<td>1.80</td>
<td>2.02</td>
<td>2.23</td>
<td>2.05</td>
</tr>
<tr>
<td>bestE Reg / bestE Red</td>
<td>$\times 3.05$</td>
<td>$\times 2.85$</td>
<td>$\times 2.86$</td>
<td>$\times 2.86$</td>
</tr>
<tr>
<td>bestS Reg / bestE Red</td>
<td>$\times 3.58$</td>
<td>$\times 3.30$</td>
<td>$\times 3.31$</td>
<td>$\times 3.80$</td>
</tr>
</tbody>
</table>

Table 5. Energy (pj/pixel) of the morphological operator on a 65-nm ASIC with best $ii$ for Reg, Rot and Red versions

<table>
<thead>
<tr>
<th>freq (MHz)</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>bestA Reg (auto $ii$)</td>
<td>2893</td>
<td>2893</td>
<td>2893</td>
<td>2896</td>
</tr>
<tr>
<td>bestE Reg (ii=5)</td>
<td>3206</td>
<td>3208</td>
<td>3206</td>
<td>3030</td>
</tr>
<tr>
<td>ratio bestE / bestA</td>
<td>1.11</td>
<td>1.11</td>
<td>1.11</td>
<td>1.01</td>
</tr>
<tr>
<td>bestE Rot (ii=4)</td>
<td>2905</td>
<td>2908</td>
<td>2923</td>
<td>2847</td>
</tr>
<tr>
<td>bestE Rot (ii=2)</td>
<td>3534</td>
<td>3534</td>
<td>3563</td>
<td>3443</td>
</tr>
<tr>
<td>ratio bestE / bestA</td>
<td>1.22</td>
<td>1.22</td>
<td>1.22</td>
<td>1.21</td>
</tr>
<tr>
<td>bestA Red (ii=4)</td>
<td>2374</td>
<td>2378</td>
<td>2400</td>
<td>2408</td>
</tr>
<tr>
<td>bestA Red (ii=2)</td>
<td>2685</td>
<td>2685</td>
<td>2714</td>
<td>2616</td>
</tr>
<tr>
<td>ratio bestA / bestE</td>
<td>1.13</td>
<td>1.13</td>
<td>1.13</td>
<td>1.09</td>
</tr>
<tr>
<td>bestA Reg / bestE Red</td>
<td>1.08</td>
<td>1.08</td>
<td>1.07</td>
<td>1.14</td>
</tr>
</tbody>
</table>

Table 6. Area ($\mu m^2$) of the morphological operator on 65 nm ASIC with best $ii$ for Reg, Rot and Red versions: the smallest area and the area associated to the smallest energy
We can perform the same analysis for the area (Tab. 6). For each level of HLT optimization (Reg, Rot and Red), there is an area increase close to 11%, 22% and 13%. But if we compare the smallest area without HLT to the area associated of the smallest Red energy version, we can see that smallest Red energy version has a smaller area than the configuration without optimization (smallest Reg area). The impact of HLTs on chip area is limited.

6. CONCLUSION AND FUTURE WORKS

We have shown that we can enhance the performance of a HLS tool like Catapult-C by performing a software optimization like register rotation (by hand or by macro meta-programming). Moreover this software optimization is more efficient than hardware optimizations such as switching to dual-port or interleaved memories for filtering.

We have shown that high level transforms (HLT) are very efficient. The principle is to transform the code – and the associated hardware – in order to get the lowest possible latency represented by the value of the initiation interval ii. Smaller ii lead to reduced power and energy consumptions and faster execution times.

For FIR with fusion of cascaded filters, the energy consumption is divided by ×5.87. For IIR, the Delay form allows to reach higher frequencies for synthesizing ASICs without needing the more recent technology. For 2D morphological filters, reduction is the key to reduce complexity and memory accesses, and thus decrease ii. We get a faster and smaller design with an energy consumption divided by ×3.5

Usually one must choose between speed and low power consumption. With the combination of HLT and HLS, no choice is needed: the ASIC is both faster and greener!

In future works, we will implement the high level transforms through algorithmic skeletons [17, 18, 19] and C++ Meta-programming [9] to make the whole process (algorithm transformation, operator fusion and synthesis) fully automatic within the reach of the underlying tools as it will require a strict conformance to C++ standard. We will target color algorithms as they are more computation intensive [20] than usual black & white algorithms.

7. REFERENCES


