A CONSTANT-GEOMETRY SEMISYSTOLIC ARCHITECTURE FOR THE FAST HARTLEY TRANSFORM*

E.L. Zapata, F. Argüello and J.D. Bruguera
15706 Santiago. SPAIN

In this article we present a parallel architecture for the calculation of the fast Hartley transform (FHT) radix 2 which is adequate for its implementation in VLSI technology. As a first step, a constant geometry (frequency decimation) algorithm for computing the FHT has been developed. The circuit we propose is characterized by its modular design and its interconnection regularity. It can be considered as semi-systolic. It permits the computation of arbitrarily sized FHTs as a consequence of data recirculation over the processing units in all the stages of the transform. Each calculation stage requires N/4Q cycles where N and Q are the length of the input real sequence and the number of processors (N=2^Q, Q=2^p), respectively. The system we propose calculates the FHT in n stages, therefore, the total calculation time is (Nlog2N)/4Q cycles.

1. The fast Hartley transform

In 1942 [1] Hartley was the first to propose the transform named after him, and it has been virtually ignored until its reintroduction by Bracewell [2-3] in 1984, who developed for this transform a similar algorithm (decimation in time) to that of Cooley-Tukey's FFT (Fast Fourier Transform) [4-6].

The Hartley transform starts with a set of samples from a signal and uses a transformation term the real variable cas(y) = cos(y) + sin(y). As it was defined by Hartley, the discrete transform H(f) of a set of N samples x(t):

\[ H(f) = \frac{1}{N} \sum_{t=0}^{N-1} x(t) \text{cas}(2\pi f t) \]  

The fast Hartley transform (FHT) algorithm computes the transform of N=2^p samples in a time proportional to Nlog2N. It is necessary to perform a bit reversal operation to obtain the results in the initial order. The bit reversal permutation can be applied before, after or in situ for the Hartley transform. We will implement a decimation in frequency approach with the permutation process after the transform. Note that the bit reversal permutation proposed by Bunneman [7] is not appropriate to make a constant geometry architecture.

In a frequency decimation algorithm, each sequence of N/2 data of stage i-1 (where N=2^i+1, i=1,...,n) is converted for stage i as follows:

\[ x'(t) = x^i(t) + x^i(N/2+t) \quad t = 0,...,N/2-1 \]
\[ x'(N/2+t) = [x^i(t) \cdot x^i(N/2+t)]\text{cos}(2\pi t/N) + [x^i(t) \cdot x^i(N/2+t)]\text{sin}(2\pi t/N) \quad t = 0,...,N/2-1 \]  

These expressions use four data items \( x^i(t), x^i(N/2+t), x^i(N/2+2t) \) and \( x^i(N-t) \). Due to the symmetry properties of the sine and cosine functions, these expressions can be rewritten in the following way:

\[ x(t) = x^i(t) + x^i(N/2+t) \quad t = 0,...,N/4-1 \]
\[ x(N/2+t) = [x^i(t) \cdot x^i(N/2+t)]\text{cos}(2\pi t/N) + [x^i(t) \cdot x^i(N/2+t)]\text{sin}(2\pi t/N) \quad t = 0,...,N/4-1 \]
\[ x(N/4+t) = x^i(N/2+t) + x^i(N/4+t) \quad t = 1,...,N/4 \]
\[ x(N/4-t) = x^i(N/2-t) \cdot x^i(N/4+t)\text{cos}(2\pi t/N) + x^i(t) \cdot x^i(N/2+t)\text{sin}(2\pi t/N) \quad t = 1,...,N/4 \]

In the case of the FFT [8], many topologies are possible for mapping the FHT onto parallel systems. Marcheti et al. [9] suggest a dynamic systolic architecture based on the concept of systolic elevator and use \( 2^k \cdot (2^m) \cdot (2^k) \cdot (2^m) \cdot 1 \) CORDIC processors (\( N=2^p \) is the length of the input sequence). Other systolic approach is developed by Chakrabarti and Jala [10], that map one dimensional transforms into two dimensional systolic arrays. Boussakta and Holt [11-12] analyze the calculation of the discrete Hartley transform using a VLSI chip based on the Fermat number transform. Zapata et al. [13] have recently developed an algorithm for computing the FHT in SIMD hypercuber computers.

Another approach, the one we intend to follow, is the construction of a constant geometry algorithm. Algorithms of this type have been developed for the FFT [14-15] which permit the repetition of a column of PEs (processing elements) throughout the array. This facilitates that a standard part of the circuit is used throughout the whole transform, and that the communications are constant.

For implementing a constant geometry algorithm, we must insure that the spacing between butterfly elements in the sequences is constant throughout all the stages of the transform. This is achieved processing the butterflies in an appropriate order and recombinig the data after each processing stage. According to equations (3), the elements that multiply the trigonometric terms are not symmetrical. This introduces some computation problems because the processing of asymmetrical terms is hard to implement.

In a constant geometry algorithm (frequency decimation) the data will not be processed in their natural order, but in one that takes into account the flow of data in the successive stages of processing. The
following procedure that we propose shuffles the sequence of data. This procedure guarantees a geometry which is regular and constant in its communications.

```c
void shuffle
{
  s = 2;
  while (s < N/4) {
    z = 2*s;
    for (j = 1; j < s; j++)
      {a = x[z+j]; x[z+j] = x[z+j+s]; x[z+j+s] = a;}
    z = z + 4*s;
    while (z < N);
    s = 2*s;
  }
}
```

The objective of the program is to exchange points x(t) and x(t+1) so as to homogenize the distribution of the two types of elements that appear in the process: those which multiply the cosine coefficients and those which multiply the sine coefficients. As an example, table I shows the pairs (t, b(t)), (natural index and shuffled index) for a sequence of 04 data items.

<table>
<thead>
<tr>
<th>k</th>
<th>b(k)</th>
<th>k</th>
<th>b(k)</th>
<th>k</th>
<th>b(k)</th>
<th>k</th>
<th>b(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>16</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>17</td>
<td>25</td>
<td>33</td>
<td>33</td>
<td>49</td>
<td>57</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>18</td>
<td>26</td>
<td>34</td>
<td>34</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>19</td>
<td>27</td>
<td>35</td>
<td>35</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>20</td>
<td>28</td>
<td>36</td>
<td>36</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>21</td>
<td>31</td>
<td>37</td>
<td>39</td>
<td>53</td>
<td>63</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>22</td>
<td>30</td>
<td>38</td>
<td>38</td>
<td>54</td>
<td>62</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>23</td>
<td>29</td>
<td>39</td>
<td>37</td>
<td>55</td>
<td>61</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>24</td>
<td>24</td>
<td>40</td>
<td>40</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>25</td>
<td>21</td>
<td>41</td>
<td>45</td>
<td>57</td>
<td>53</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>26</td>
<td>22</td>
<td>42</td>
<td>46</td>
<td>58</td>
<td>54</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td>27</td>
<td>23</td>
<td>43</td>
<td>47</td>
<td>59</td>
<td>55</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>28</td>
<td>20</td>
<td>44</td>
<td>44</td>
<td>60</td>
<td>52</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
<td>29</td>
<td>19</td>
<td>45</td>
<td>43</td>
<td>61</td>
<td>51</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>30</td>
<td>18</td>
<td>46</td>
<td>45</td>
<td>62</td>
<td>50</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>31</td>
<td>17</td>
<td>47</td>
<td>41</td>
<td>53</td>
<td>49</td>
</tr>
</tbody>
</table>

Table I. Operation of the procedure Shuffle for a sequence of 64 data, where k is the original index and b(k) is the shuffled index.

### 3. Parallel architecture

In the design of the parallel architecture we have considered a limited number of PEs, which makes its integration feasible. Nevertheless, we intend to make it as general as possible, not depending on the size of the input sequence. This implies the introduction of specific hardware, for example FIFOs, the length of which will depend on the size of the problem.

We propose an efficient parallel circuit adequate for its VLSI implementation. The number of PEs will be a power of two (\( Q = 2^r \)) and each one of them will compute a four point butterfly. The PEs will operate in parallel, each one of them computing one of the \( n = \log_2 N \) stages that make up the transform. While the computation of the butterflies is performed in the PEs, the interconnection network will recirculate the calculated elements and deliver them to the PEs as data, in an appropriate order for them to compute the butterflies corresponding to a new stage of the transform. The results will come out in a bit reversal order as corresponds to a frequency decimation FFT.
Data must be inserted between paths 0 and 2 and paths 1 and must be shuffled before we make them enter the same PE. There are two even or 2 and

A2) connected to the input paths of the p-th PE are numbered 0 and 1 if p is actions that must be performed: change the order of the input of data (process B) and exchange data between the paths (processes C1 and

AI) If we are in the initial cycle

cases can be distinguished:

PE is connected to two others. Two of the input paths off the p-th PE originate in the p/2-th PE and the other two in the {Q/2+p/2}-th PE.

terms sine and cosine

C2) Cl)

Cl)

The procedure butterfly(i) is the computation of stage i of the transform. The data enter the PES in parallel (process A) and recombine subsequences within the same PE (process C).

The general structure of the FHT. The interconnections of the circuit are constructed from these processes. Process A recombines subsequences from different PEs and gives an interconnection rule between the modules of the circuit. Processes B and C recombine subsequences within the same PE and control the movement of data within each module.

As we have indicated before, we will suppose that each PE has a local memory where the trigonometric coefficients are found. In a sequential algorithm, the number of sines and cosines is N/4. In a parallel algorithm it is not possible to distribute these coefficients among the PEs, as they are not used by more than one PE. It is necessary to introduce a certain amount of redundancy.

3.2. Structure of the circuit we propose

The parallel circuit we propose is illustrated in figure 3. The general structure of the circuit with its external feedback loops, input/output network and connections of the different PEs (4 in this case) is shown there. Each of these PEs (figure 4a) include the internal feedback network, the input multiplexors (MUXO), which select the entry path for the data in the PE, the processing section (PS), the routing circuit (RC) and a switch (SW). The structure of the PS and RC is also modular, it can be decomposed into two identical subcircuits: submodules X and R in the figure.

![Fig. 3. Structure of the parallel circuit we propose for Q=4.](image-url)
The operation they have to perform is slightly different in the two submodules X of the processing section. The normal operation for the bottom submodule is \( x = c + d \) and \( d' = -(c-d)k(x) + (a-b)sin(\phi) \). \( \phi = 0 \)
is a singularity in which \( x = c + d \) and \( d' = c - d \) (instead of \( d' = d - c \)), as we can
deduce from equations (3). A control input, coming from a zero
detector is included to detect this situation \((\sin(\phi) = 0\), see figure 1). The
operation of the top submodule is similar, but in this case there are no
singularities and the control input is always zero.

The routing circuit (RC) performs the data movement processes
A1, A2 (together with the external recirculation network) and B. Figure 4a
shows the RC circuit, made up by two identical submodules (R). Figure 4b
shows the internal structure of submodule R with more detail. We have
 included two FIFO queues to prevent the loss of data (rewriting) in a
 process such as B. They will alternate throughout the transform. During
 one stage, one queue is loaded (stores data for the next stage) and the
 other is unloaded (reads data from the previous stage).

MUX1 changes the data coming from the two input paths from
 parallel to serial. These data items are introduced into the FIFOs, which
 are after is, at a speed of two words per cycle. This speed is independent
 of the process A or B we are carrying out (both require the same loading
 sequence).

\[
\begin{align*}
&c_{w_1} \quad c_{w_1}^1 \quad d_{w_1}^1 \quad d_{w_1}^2 \\
&d_{w_1} \quad d_{w_1}^1 \quad c_{w_1}^1 \quad c_{w_1}^2
\end{align*}
\]

The signals CK0 and CK1, together with the multiplexors MUX2
and MUX3 control the loading and unloading of the FIFO queues. Each
queue is made up by two W-1 word FIFOs plus an additional latch (H),
the unions of which coincide with the outputs of the queue (see figure 4b).

As we have mentioned previously, processes A and B are different
in the unloading of the FIFO queues. In a process type B, during an
 internal recirculation, the first half of the data stored in the FIFO queue
 will be directed to path \( c' \) and the second half to path \( d' \). The FIFO
 queue will move at a speed of one word per cycle generating the following
 output sequence,

\[
\begin{align*}
&d_{w_1}^1 \quad c_{w_1}^1 \quad d_{w_1}^1 \quad c_{w_1}^1 \quad d_{w_1}^2 \quad c_{w_1}^2
\end{align*}
\]

Fig. 4. PE internal structure
a) Structure of the PE modules.
b) Structure of the subcircuit R of the routing circuit.
c) Structure and operation of the switch (SW).

Fig. 5. Data flow for a FFT with 64 points in the circuit we propose \((Q = 4\) PEs).
This unfolding of the sequence stored in a FIFO queue is accomplished by means of the partition of the queue into two segments with W-1.

With a process type A (external recirculation) we recover the parallelism (change from serial to parallel) suppressed by MUXI (sequence (9)) and the data items come from module R in the same order that they entered it. The movement of the FIFO queue is very simple. It will suffice with increasing the number of communications. There are only 4Q connections with a length of W-1.

The last module appearing in figure 4a is the switch (SW) which returns the bit y of the integer associated with the least significant bits of x. This unfolding of the sequence stored in a FIFO queue is accomplished by means of the partition of the queue into two segments with W-1.

The total number of operating cycles is nW and, therefore, one possibility is to introduce the nW angles into each PE. In this case, the addressing of the table is very simple. It will suffice with increasing the memory address by one unit each cycle. Another alternative which minimizes the redundance associated to the table implies a (q + l)W integer constants (m) which determine the angles (+=2πj):

\[ \text{for } p=0; p<Q; p++ \}
\[ \text{for } j=0; j<W; j++ \}
\[ m[p][(j+p)*W+j]=a[(j+(p>q)*W)<c+1]; \]

where a is a vector with N/4 integer constants corresponding to all the possible angles of the transform and m is a two-dimensional array whose p-th line contains the (q + 1)W elements of the vector a needed by the p-th PE. This solution, that reduces the redundance, requires the addressing process of this table to be the following:

\[ \text{if } i \leq q \]
\[ m[p][(i-1)*W+j]= \]
\[ \text{otherwise } m[p][q*W+j-Isn(j,i-q-l)], \]

where i represents the stage of calculation of the transform, j the computation cycle within this stage and p-th PE. bit(x,y) is the function that returns the bit y of the integer x, and the function Isn(x,y) returns the integer associated with the least significant bits of x.

One final aspect we want to mention about the architecture we propose is the high level of regularity it possesses both at the system level, all the processing modules are identical, as in their internal structure. These characteristics make its implementation in VLSI technology feasible and relatively simple.

### References


