18-643 Lecture 12: 
Spiral: Domain-Specific HLS

James C. Hoe 
Department of ECE 
Carnegie Mellon University

Housekeeping

• Your goal today: see an example of really high-level synthesis (this lecture not on Midterm)
• Notices
  – Handout #4: lab 2, due noon, 10/6
  – Handout #5: lab 3, due noon, 10/20
  – 2.5 weeks to project proposal
  – 1.5 week to midterm
• Readings
Conflict btwn High-Level and Generality

- high-level: tool knows better than you
- HLS: tool decides what you can say and what you mean
- RTL synthesis: general-purpose but special handling of structures like FSM, arith, etc.
- place-and-route: works the same no matter what design

Spiral DFTgen: how high can you go?

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>range</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem specification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transform size</td>
<td>4,64,128</td>
<td>4–32/64</td>
<td>Number of samples [1]</td>
</tr>
<tr>
<td>direction</td>
<td>forward</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data type</td>
<td>fixed point</td>
<td>4–32 bits</td>
<td></td>
</tr>
<tr>
<td>data ordering</td>
<td>natural</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRAM budget</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permutation method</td>
<td>ועד[2]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

HLS: tool decides what you can say and what you mean.
**Design Space and Quality of Result**

DFT 1024 (16 bit fixed point) on Xilinx Virtex-6 FPGA

![Graph showing performance vs. area and slices](image)

**SPIRAL Framework**

DSP transform (user specified)

- Algorithm Level
  - Algorithm Generation
  - Algorithm Optimization
- Implementation Level
  - Implementation
  - Code Optimization
- Evaluation Level
  - Compilation
  - Performance Evaluation

**Principle 1**: Domain knowledge in the system

**Principle 2**: Optimization at a high level of abstraction
Very-High-Level Description

Linear Transforms

• Linear transform is a matrix-vector multiplication
  – computing by definition takes $O(N^2)$ operations
  – the matrix has structure
• E.g. discrete Fourier transform: $y = DFT_N \cdot x$

$$
\begin{bmatrix}
y_0 \\
y_1 \\
\vdots \\
y_{N-1}
\end{bmatrix} =
\begin{bmatrix}
x_0 \\
x_1 \\
\vdots \\
x_{N-1}
\end{bmatrix} \cdot
e^{-i2\pi jk/N}
$$

$e^{-i2\pi/8}$ e.g., $8^{th}$ roots of unit
“Fast” Algorithms

- “Fast” algorithm factors the matrix into a sequence of structured, sparse matrices
  
  cheaper sparse multiplies ⇒ \(O(N \log(N))\) operations

- E.g. Cooley-Tukey Factorization of \(DFT_4\)

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & i & -1 & -i \\
1 & -1 & 1 & -1 \\
1 & -i & -1 & i
\end{bmatrix} = \begin{bmatrix}
1 & 1 & \cdots & 1 \\
1 & 1 & \cdots & 1 \\
1 & -1 & \cdots & 1 \\
1 & -i & \cdots & 1
\end{bmatrix} DFT_2 \otimes I_2 \otimes DFT_2 = L_2^4
\]

- Matrix formula representation

\[
DFT_4 = (DFT_2 \otimes I_2)DFT_2^4 (I_2 \otimes DFT_2)L_2^4
\]

---

Factorization Rules

E.g. Cooley-Tukey

\[
DFT_{n \cdot m} = (DFT_n \otimes I_m) D_{n \cdot m} \otimes (I_n \otimes DFT_m) L_{n \cdot m}
\]

- \(DFT_2\) is

\[
\begin{bmatrix}
1 & 1 \\
1 & -1
\end{bmatrix}
\]

- \(D\) is a diagonal matrix of twiddle factors

- \(L\) is a stride permutation matrix

- \(A \otimes B = [a_{i,j}B]\) is the tensor (or kronecker) product

E.g., \(I_n \otimes B \Rightarrow \begin{bmatrix}
B & B & \cdots & B \\
0 & B & \cdots & B \\
\cdots & \cdots & \cdots & \cdots
\end{bmatrix} \quad A \otimes I_n \Rightarrow \begin{bmatrix}
a_{0,0} & 0 & a_{1,0} & 0 & \cdots \\
0 & a_{0,0} & 0 & a_{1,0} & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\]
Fast Fourier Transform Algorithms

- Recursively factorize by Cooley-Tukey rule until only leaf cases remain (e.g. $DFT_r$ for radix-$r$)

$$DFT_8 = (DFT_2 \otimes I_4)D_2^8(I_2 \otimes DFT_4)L_2^8$$

$$= (DFT_2 \otimes I_4)D_2^8(I_2 \otimes (DFT_2 \otimes I_2)D_2^4(I_2 \otimes DFT_2)L_2^4)L_2^8$$

- Exponential number of alternatives

- Each ruletree corresponds a different algorithm
- All cost $O(N \log(N))$

---

Describing a Design Space vs a Point

$$DCT_2^{(II)} \rightarrow \text{diag}(1,1/\sqrt{2}) \cdot F_2$$
$$DCT_n^{(II)} \rightarrow P \cdot (DCT_{n/2}^{(II)} \otimes DCT_{n/2}^{(IV)}) \cdot (I_{n/2} \otimes F_2)^I$$
$$DCT_n^{(IV)} \rightarrow S \cdot DCT_n^{(II)} \cdot D$$
$$DCT_n^{(IV)} \rightarrow M_1 \cdots M_r$$
$$DFT_n \rightarrow B \cdot (DCT_{n/2}^{(I)} \otimes DST_{n/2}^{(I)}) \cdot C$$
$$DFT_{nm} \rightarrow (DFT_n \otimes I_m) \cdot D \cdot (I_n \otimes DFT_m) \cdot P$$
$$F_n(h) \rightarrow (I_{n/2} \otimes^k I_{d+1}) \cdot (I_{n/2} \otimes F_d(h))$$
$$F_n(h) \rightarrow \text{Circ}((\tilde{h}) \cdot E)$$
$$DWT_n(W) \rightarrow (DWT_{n/2}(W) \otimes I_{n/2}) \cdot P \cdot (I_{n/2} \otimes_k W) \cdot E$$
$$WHT_{2^n} \rightarrow \prod_{i=1}^{n} (I_{2^n \cdots_{i+1}} \otimes WHT_{2^n}^{(n)} \otimes I_{2^n \cdots_{i+1}})$$

Done once per transform by an expert and the tool becomes the expert
Very-High-Level Synthesis

Formula to HW

- Given \( y = M \cdot x \) where \( M \) is:
  - \( M = A \cdot B \) apply \( B \) then \( A \)
  - \( M = I_n \otimes A \) apply \( A \), \( n \) times in parallel
  - \( M \) is a permutation permute \( x \)
  - \( M \) is a diagonal scale \( x \)

\[
y = (A \cdot B) \cdot x = A \cdot (B \cdot x)
\]

\[
y = (I_2 \otimes A) \cdot x
\]

\[
y_0 = A \cdot y_0
\]

\[
x_0 \xrightarrow{A} y_0
\]

\[
x_1 \xrightarrow{y_1}
\]

\[
x_2 \xrightarrow{y_2}
\]

\[
x_3 \xrightarrow{y_3}
\]

\[
x_0 \xrightarrow{y_0}
\]

\[
x_1 \xrightarrow{y_1}
\]

\[
x_2 \xrightarrow{y_2}
\]

\[
x_3 \xrightarrow{y_3}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot x
\]

\[
\begin{bmatrix}
7 & 0 & 0 & 0 \\
0 & 8 & 0 & 0 \\
0 & 0 & 2 & 0 \\
0 & 0 & 0 & 4
\end{bmatrix} \cdot x
\]
DFT\textsubscript{8} Example

\[ DFT_8 = (DFT_2 \otimes I_4)D_2^8(I_2 \otimes ((DFT_2 \otimes I_2)D_2^4(I_2 \otimes DFT_2)L_2^4)L_2^8) \]

(formula is applied from right to left)

Pease DFT\textsubscript{8} Example

\[ DFT_8 = R_8 \cdot T_2(I_4 \otimes F_2)L_4^8 \cdot T_1(I_4 \otimes F_2)L_4^8 \cdot T_0(I_4 \otimes F_2)L_4^8 \]
How about good HW?

- Formulas map naturally to combinational dataflow, but this is neither good nor realistic

  What if I want DFT\textsubscript{16K}?

- Sequential datapath to reuse available HW
  - identify repeated kernels
  - instantiate kernels under resource constraints
  - schedule computation to reuse instantiated kernels

  Do this at formula level with math-level knowledge

---

Tensor as Streaming Pipeline

\[
I_m \otimes A_n \quad I_m \otimes^{SR} A_n \quad I_{mn/w} \otimes^{SR} (I_{w/n} \otimes A_n)
\]

- fully parallel
- fully streamed
- partially streamed

Like data-parallel loops we seen in regular HLS
Pease DFT_8

\[ \text{DFT}_8 = R_8 \cdot T_2(I_4 \otimes F_2) L_4^8 \cdot T_1(I_4 \otimes F_2) L_4^8 \cdot T_0(I_4 \otimes F_2) L_4^8 \]

Streaming Pease DFT_8

\[ \text{DFT}_8 = R_8 \cdot T_2(I_4 \otimes F_2) L_4^8 \cdot T_1(I_4 \otimes F_2) L_4^8 \cdot T_0(I_4 \otimes F_2) L_4^8 \]
Iterative Reuse

\[ \prod_{\ell=0}^{m-1} A_n \]

no reuse

\[ \prod_{\ell=0}^{m-1} A_n \]

1 block, reused \( m \) times

\[ \prod_{\ell=0}^{p-1} \prod_{k=0}^{(m/p)-1} A_n \]

\( m/p \) blocks, reused \( p \) times

partially iterative reuse

Like data-dependent loops we seen in regular HLS

Iterative Pease DFT_8

\[ w = 2, 4, \ldots, N \]

\[ w_{\text{max}} = N \]

\[ w_{\text{min}} = 2 \]

Fine-grained control over cost/latency tradeoff

\[ \text{cost} \propto w; \text{ latency} \propto \frac{1}{w} \]
## Rewrite Rules for Streaming and Reuse

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>base-SR</td>
<td>$A_k \xrightarrow{stream} A_k$</td>
<td></td>
</tr>
<tr>
<td>product-SR</td>
<td>$A_1 \cdot A_2 \cdots A_k \xrightarrow{stream} A_1 \cdot A_2 \cdots A_k$</td>
<td>$A_k \neq 0$</td>
</tr>
<tr>
<td>stream-IR</td>
<td>$\prod A_k \xrightarrow{stream} \prod A_k$</td>
<td>$A_k \neq 0$</td>
</tr>
</tbody>
</table>

| Stream1 | $I_{m \times w} \otimes \delta \rightarrow I_{m \times w} \otimes \delta \left( I_{n \times k} \otimes \delta \right)$ | $m > w$ and $k \leq w$ |
| Stream1-dep | $I_{m \times w} \otimes \delta \rightarrow I_{m \times w} \otimes \delta \left( I_{n \times k} \otimes \delta \right)$ | $m > w$ and $k \leq w$ |
| Stream2 | $I_{n \times k} \otimes \delta \rightarrow I_{n \times k} \otimes \delta$ | $k > w$ |
| Stream2-dep | $I_{m \times w} \otimes \delta \rightarrow I_{m \times w} \otimes \delta$ | $k > w$ |
| Stream-diag | $\delta \xrightarrow{stream} \text{StreamDiag}(D_n, w)$ | $w \mid n$ |
| Stream-perm | $\delta \xrightarrow{stream} \text{StreamPerm}(P_n, w)$ | $w \mid n$ |

"pragmas"

## Applicability to other transforms

- **DFT radix 2**
  \[ R_{2^k} \prod_{i=0}^{k-1} \left[ T_i \left( I_{2^{k-i}} \otimes DFT_{2^i} \right) \right] \]

- **DFT radix $2^r$**
  \[ R_{2^k} \prod_{i=0}^{k-1} \left[ T_i \left( I_{2^{k-i}} \otimes DFT_{2^r} \right) \right] \]

- **2-D DFT$_{nxn}$**
  \[ \prod_{i=0}^{1} \left[ L_n^{2^i} \left( I_n \otimes DFT_n \right) \right] \]

- **WHT**
  \[ \prod_{i=0}^{k \cdot r - 1} \left[ I_{2^{k-i}} \otimes WHT_{2^i} \right] \]

- **DCT (type II)**
  \[ DP \prod_{i=0}^{k-1} \left[ A_{k-i} \cdot L_{2^i}^{2^{k-i}} \right] \]
Toward Very-High-Level IPs

Is DFTgen Easy to Use?

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem specification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transform size</td>
<td>4 ( )</td>
<td>4-32/64</td>
<td>Number of samples [1]</td>
</tr>
<tr>
<td>direction</td>
<td>forward</td>
<td></td>
<td>forward or inverse DFT [2]</td>
</tr>
<tr>
<td>data type</td>
<td>fixed point</td>
<td></td>
<td>fixed or floating point [2]</td>
</tr>
<tr>
<td></td>
<td>16 ( )</td>
<td>4-32 bits</td>
<td>fixed point precision [2]</td>
</tr>
<tr>
<td></td>
<td>unscaled</td>
<td></td>
<td>scaling mode [2]</td>
</tr>
<tr>
<td>Parameters controlling implementation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>architecture</td>
<td>full streaming</td>
<td></td>
<td>iterative or fully streaming [1]</td>
</tr>
<tr>
<td>radix</td>
<td>2 ( )</td>
<td>2, 4, 8, 16, 32, 64</td>
<td>size of DFT basic block [3]</td>
</tr>
<tr>
<td>streaming width</td>
<td>2 ( )</td>
<td>2-64</td>
<td>number of complex words per cycle [3]</td>
</tr>
<tr>
<td>data ordering</td>
<td>natural/renved</td>
<td></td>
<td>natural or digit-reversed data order [3]</td>
</tr>
<tr>
<td>BRAM budget</td>
<td>1000</td>
<td></td>
<td>maximum # of BRAMs to utilize (-1 for no limit) [3]</td>
</tr>
<tr>
<td>Permutation method</td>
<td>JACH/M9 (registered)</td>
<td></td>
<td>Please click for more information</td>
</tr>
</tbody>
</table>

http://www.spiral.net/hardware/dftgen.html
Easy to Use for Whom?

- Powerful? Very!
- Easy to use? Not Really . . .
  - low-level cryptic domain-specific parameters
  - complexity of integrating, using, tuning and validating an instantiated IP within an enclosing context
- If you went to DFTgen right now
  - which configuration would you ask for first?
  - if not good enough, how to get a better one . . . .
  - do you know what good enough is . . . .

Different Kinds of Experts

- **Application Developers**
  - Assemble, configure and integrate multiple IPs to build larger chips
- **Domain Experts**
  - Know the underlying algorithms and theory specific to the domain
- **IP Authors**
  - Can build HW based on a set of specs or SW implementation
Make generator the IP

- Why limit to structural view of design
- Why not offer also . . . .
  - pre-knowledge about outcome & tradeoff of parameter combinations
  - IP-specific “meaningful” parameterizations, that is, ask how fast? instead of how many?
  - performance self-monitor, interface protocol checker
  - any X where IP authors can do better than IP users

Shift burdens from IP users to IP authors

⇒ make knowledge and expertise reusable

Parting Thoughts

- Encapsulating domain knowledge in a domain specific tool for truly high-level design automation

- Why is Spiral-DSP so good?
  Ans: it only does linear DSP transforms
  (fortunately FFT is pretty important)

  - very well understood mathematics
  - highly structured, highly regular computation
  - enumerable design space

Underlying approach/framework is generalizable!!