Z Fitter Algorithm and Implementation

Masahiro Morii, Harvard U.

- Requirements, I/O
- Algorithm
- Implementation
- Resources & Latency
Executive Summary

- Z Fitter measures track’s $z_0$, $p_T$, $\tan \lambda$
- Algorithm demonstrated in C++ emulation
  - LUTs reduce real-time computation to minimum
- Resource and timing evaluated
  - Use 4% of CLBs in XC2V4000
  - Can process 3 seeds/CLK4 in pipeline
  - Latency < 1 CLK4 for each seed
- FPGA implementation ready to start
**Functionality**

- Fit seed tracks from the Finder to a helix
  - A seed track = a set of 10 TSF segments
  - Measure $z_0$, $p_T$, $\tan \lambda \rightarrow$ Decision Module
I/O

- Inputs from Seed Finder
  - TSF segments, hit map
  - Curvature $\rho$ (or $1/p_T$), $\tan \lambda$
  - FPGA internal bus

- Outputs to Decision Module
  - Fit results: $z_0$, $z_0$ error, $\rho$, $\tan \lambda$
  - Hit map
  - Sent over 10 traces at 45 MHz
Inputs

- TSF segments
  - 10 bit $\phi$ and 4 bit error
    - $\phi$ is relative to the seed segment
      - Only 9 segments/seed are needed
    - Error is not used by the Fitter

- Hit map
  - Which layer had a segment $\rightarrow$ 10 bits
Inputs

- Curvature $\rho$
  - Fitter needs a 1st guess of $\rho$ from the Finder
  - 6-bit resolution

- $\tan\lambda$
  - Not used by the Fitter
  - Finder provides 6-bit resolution
Outputs

- **Fit results**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Unit</th>
<th>Resolution</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_0$</td>
<td>cm</td>
<td>8 bits</td>
<td>±127 cm</td>
</tr>
<tr>
<td>$z_0$ error</td>
<td>cm</td>
<td>4 bits</td>
<td>0 to 15 cm</td>
</tr>
<tr>
<td>$\rho$</td>
<td>$2^{-12} cm^{-1}$</td>
<td>8 bits</td>
<td>$</td>
</tr>
<tr>
<td>$\tan \lambda$</td>
<td>$2^{-5}$</td>
<td>8 bits</td>
<td>$</td>
</tr>
</tbody>
</table>

- **Hit map**
  - Passed through from input
Algorithm

- **Step 1: r-\(\phi\) fit**
  - Ignore stereo information
    - 6 measurements of \(\phi\) at different \(r\)
  - Find seed \(\phi\) and \(\rho\) that minimize \(\chi^2\) in r-\(\phi\)

- **Step 2: \(z_0\) fit**
  - Use stereo information
    - 6 measurements of \(z\) at different \(r\)
  - Find \(z_0\) and \(\tan\lambda\) that minimize \(\chi^2\) in r-\(z\)
**r-ϕ Fit**

- Merge stereo layers $\rightarrow$ virtual axial layers
  - 3 U+V pairs plus 3 axial $\rightarrow$ 6 r-ϕ measurements

Subtract shift due to curvature using input $\rho$

Residuals are due to errors in $\rho$ and $\phi$(seed)
**r-φ Fit**

- **φ residual**
  \[ \phi_i^{\text{resid}} = \frac{\partial \phi_i}{\partial \rho} \Delta \rho - \Delta \phi_{\text{seed}} \]

- **Calculate and minimize**
  \[ \sum (r_i \phi_i^{\text{resid}})^2 \]

\[
\Delta \phi = \frac{\sum r_i^2 \phi_i \sum r_i^2 \left( \frac{\partial \phi_i}{\partial \rho_{in}} \right)^2 - \sum r_i^2 \phi_i \frac{\partial \phi_i}{\partial \rho_{in}} \sum r_i^2 \frac{\partial \phi_i}{\partial \rho_{in}}}{\sum r_i^2 \sum r_i^2 \left( \frac{\partial \phi_i}{\partial \rho_{in}} \right)^2 - \left( \sum r_i^2 \frac{\partial \phi_i}{\partial \rho_{in}} \right)^2}
\]

\[
\Delta \rho = \frac{\sum r_i^2 \phi_i \sum r_i^2 \frac{\partial \phi_i}{\partial \rho_{in}} - \sum r_i^2 \phi_i \frac{\partial \phi_i}{\partial \rho_{in}} \sum r_i^2}{\sum r_i^2 \sum r_i^2 \left( \frac{\partial \phi_i}{\partial \rho_{in}} \right)^2 - \left( \sum r_i^2 \frac{\partial \phi_i}{\partial \rho_{in}} \right)^2}
\]

**Error of ρ**

**Error of seed φ**

\[ \rho = \rho + \Delta \rho \]
**z_0** Fit

- Go back to 6 stereo layers
  - Apply corrections for $\Delta \rho$ and $\Delta \phi$

\[ \phi = \Delta \phi \]

Subtract shift due to curvature using fitted $\rho$

Residuals are due to stereo angles
$z_0$ Fit

- $\phi$ residual

\[ \phi_{\text{resid}}^i = z_i \frac{\tan \beta_i}{r_i} \]

- 6 $z_i$ make a straight line in d-z plane
**$z_0$ Fit**

- **Minimize**
  \[ \chi^2 = \sum \frac{(z - z_0 - d \tan \lambda)^2}{\sigma_i^2} \]

- **Assume error**
  \[ \sigma_i = \frac{r_i \times (1 \text{mm})}{\tan \beta_i} \]

\[
\begin{align*}
  z_0 &= -\frac{\sum \frac{z_i}{\sigma_i^2} \sum \frac{d_i^2}{\sigma_i^2} - \sum \frac{z_i d_i}{\sigma_i^2} \sum \frac{d_i}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{d_i^2}{\sigma_i^2} - \left( \sum \frac{d_i}{\sigma_i^2} \right)^2} \\
  \tan \lambda &= -\frac{\sum \frac{z_i d_i}{\sigma_i^2} \sum \frac{1}{\sigma_i^2} - \sum \frac{z_i}{\sigma_i^2} \sum \frac{d_i}{\sigma_i^2}}{\sum \frac{1}{\sigma_i^2} \sum \frac{d_i^2}{\sigma_i^2} - \left( \sum \frac{d_i}{\sigma_i^2} \right)^2}
\end{align*}
\]
Implementation

- Biggest concern: Speed
- Pre-compute as much as possible
  - Most computation packed in LUTs
  - Only additions and multiplications at run time
- First step: Software emulation
  - Bit-wise emulation of what hardware will do
  - Validate the algorithm
  - Study and optimize the performance
Software Emulation

The whole code (58 lines C++) is made of LUTs, additions, multiplications and bit shifts

```c++
bool LIDczNIFitter::zFitter(const LIDczNIFTable* table, int hitmap,
    const int* segphi, int rhoin,
    int &z0, int &z0err, int &rhoout, int &dipout) const
{
    if (!table->fitok(hitmap)) {
        z0 = -128;
        z0err = 15;
        rhoout = -128;
        dipout = -128;
        return false;
    }
    int phi[9];
    int rh = table->rh5(rhoin);
    int hitax = table->hitax(hitmap);
    int sumz2phi = 0;
    int sumz2phidpdr = 0;
    int i;
    for (i = 0; i < 9; i++) {
        phi[i] = (segphi[i]*table->phiconv(i))>>13;
        if (rhoin > 0) phi[i] += table->curvcorr(i,rh);
        else phi[i] -= table->curvcorr(i,rh);
        if (table->useax(hitax,i)) {
            sumz2phi += table->wr2(i)*phi[i];
            sumz2phidpdr += table->wr2dpdr(i,rh)*phi[i];
        }
    }
    int dPhi1 = (sumz2phi*table->sumz2dpdr2(hitax,rh))>>13;
    int dPhi2 = (sumz2phidpdr*table->sumz2dpdr(hitax,rh))>>13;
    int dPhi3 = (dPhi1-dPhi2)*table->denomzp(hitax,rh);
    int dPhi = dPhi3>>16;
    int dRh01 = (sumz2phi*table->sumz2dpdr(hitax,rh))>>14;
    int dRh02 = (sumz2phidpdr*table->sumz2(hitax))>>14;
    int dRh03 = (dRh01-dRh02)*table->denomrp(hitax,rh);
    int dRho = dRh03>>16;
    rhoout = (rhoin<<2) + dRho;
    hitmap &= 63;
    rh = table->rh3(rhoout);
    int sumzs2 = 0;
    int sumzs22 = 0;
    for (i = 0; i < 6; i++) {
        phi[i] += -dPhi + ((table->dphiadrho(i,rh)*dRho)>>6);
        int z = (table->stereom(i)*phi[i])>>8;
        if (hitmap & (1<<i)) {
            sumzs2 += z*table->sigma2z(i);
            sumzs22 += z*table->dsigma2z(i,rh);
        }
    }
    int z01 = (sumzs2*table->sumd2s2(hitmap,rh))>>6;
    int z02 = (sumzs22*table->sumd2s2(hitmap,rh))>>6;
    int z0 = ((z01-z02)*table->denomz(hitmap,rh))>>16;
    int td1 = (sumzs2*table->sums2(hitmap))>>1;
    int td2 = (sumzs22*table->sums2(hitmap,rh))>>1;
    dipout = (td1-td2)*table->denomzt(hitmap,rh))>>16;
    z0err = table->z0err(hitmap,rh);
    return true;
}
```
Engineering Constraints

- FPGA: Xilinx Virtex-II
  - XC2V4000 chosen for the Seed Track Finder
  - Allow much smaller resources than Finder
    - As few as possible CLBs
- Latency: as short as possible
  - Ideally < 1 CLK4
  - FPGA runs at 180 MHz \( \rightarrow \) 48 ticks/CLK4
    - Most logic operations take 1 tick
    - 18-bit multiplication takes 2 ticks
Seed Counting

- 12 seeds/module/CLK4 → 4 Engines
  - Each Finder/Fitter pair processes 3 seeds
  - Fitter receives a new seed every 1/3 CLK4
  - Fitting takes ~3/4 CLK4 for each seed
  → Pipeline processing

![Diagram of seed counting process]

A seed arriving every 1/3 CLK4
~3/4 CLK4
Data Flow

- 9 TSF segments x 10 bits/segment
- Segment serializer
- r-φ pipeline
- r-φ pipeline
- r-φ pipeline
- Accumulator
- Dual-port memory
- z0 pipeline
- z0 pipeline
- z0 pipeline
- Accumulator
- Accumulator
- z0 error
- curvature
- z0 error
- tanλ
- hit map
- 9/11/01 CDR
- Z Fitter
r-φ Fit Block

3 segs/pipeline

Input 9 segments

Segment serializer

r-φ pipeline

r-φ pipeline

r-φ pipeline

Accumulator

Dual-port memory

Unit conversion
Stereo cancellation
Curvature subtraction

Accumulate

\[ \sum r_i^2 \phi_i \]

\[ \sum r_i^2 \phi_i \frac{\partial \phi_i}{\partial \rho_{\text{in}}} \]

Carry stereo segments to \( z_0 \) Fit
Accumulator

Accumulate 2 quantities from 3 sources arriving every 2 ticks

Pipeline 1

Pipeline 2

Pipeline 3

LUT

Switch

adder

MUX

Σ

\[ \sum r_i^2 \phi_i \]

\[ \sum r_i^2 \phi_i \frac{\partial \phi_i}{\partial \rho_{in}} \]
Δφ and Δρ Calculation

\[ \sum r_i^2 \phi_i \]

\[ \sum r_i^2 \phi_i \frac{\partial \phi_i}{\partial \rho_{in}} \]

hit map

curvature

LUT

LUT

6 multiplications and 2 additions in 5 ticks
$z_0$ Fit Block

6 Stereo segments

Dual-port memory

3 segs/pipeline

$\Delta \phi$ and $\Delta \rho$ from r-$\phi$ Fit

$z_0$ pipeline

$z_0$ pipeline

Accumulator

Accumulate

$\sum \frac{z_i}{\sigma_i^2}$ $\sum \frac{z_i d_i}{\sigma_i^2}$

$\Delta \phi$, $\Delta \rho$ correction

$\phi \rightarrow z$ conversion

$z_0$ tan$\lambda$

Done!
## Resources

### Dominated by the LUTs

<table>
<thead>
<tr>
<th>Function</th>
<th>CLBs</th>
<th>RAM blocks</th>
<th>Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look-up tables</td>
<td>144</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Computation</td>
<td>42</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>Serialization</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dual-port memory</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Z Fitter</strong></td>
<td><strong>217</strong></td>
<td><strong>8</strong></td>
<td><strong>29</strong></td>
</tr>
<tr>
<td>Available in XC2V4000</td>
<td>5760</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td><strong>Usage (%)</strong></td>
<td><strong>4%</strong></td>
<td><strong>7%</strong></td>
<td><strong>24%</strong></td>
</tr>
</tbody>
</table>
Timing and Latency

- Separation between two input seeds ≥ 15
  - OK to process 3 seeds/CLK4 @ 180 MHz
  - 2 seeds/CLK4 if 120 MHz (same as Finder)
- Latency for $z_0$ and $\tan\lambda = 37$ ticks
  - ~3/4 CLK4 @ 180 MHz
  - Output will add ~5 ticks → Still < 1 CLK4
  - If 120 MHz, ~1.3 CLK4
Executive Summary

- Z Fitter measures track’s $z_0$, $p_T$, $\tan \lambda$
- Algorithm demonstrated in C++ emulation
  - LUTs reduce real-time computation to minimum
- Resource and timing evaluated
  - Use 4% of CLBs in XC2V4000
  - Can process 3 seeds/CLK4 in pipeline
  - Latency < 1 CLK4 for each seed
- FPGA implementation ready to start