Signal Flow Graph Manipulation

Lec 7

Reading - Parhi

Let's start with our FIR filter

\[ X(n) \rightarrow z^{-1} \rightarrow h_0 \rightarrow \times \rightarrow h_1 \rightarrow \times \rightarrow h_2 \rightarrow \times \rightarrow X_{out}(n) \]

\[ z^{-1} \quad \equiv \quad \text{DECEL} \quad X_{in}(n-1) \]

A delay by one sample period

A more abstract and efficient notation

\[ X_{in}(n) \rightarrow z^{-1} \rightarrow z^{-1} \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow X_{out}(n) \]

Assume an add when nodes come together
2) Sample interleaving (cont.)

Multiple data streams can be multiplexed into one SFG

\[ X_n(0), X_n(1), X_n(2) \]
\[ Y_n(0), Y_n(1), Y_n(2) \]
\[ Z_n(0), Z_n(1), Z_n(2) \]

\[ \text{At clock rate } f_c \]

\[ \xrightarrow{\text{At clock rate } 3f_c} X_n(0), Y_n(0), Z_n(0), X_n(1), Y_n(1) \]

This goes into SFG on previous page.

3) Changing the latency (delay from input to output) usually is not considered important, i.e., \[ z^{-3} z^{-1} z^{-1} \]

\[ X_n(n) \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow z^{-4} \rightarrow X_{out}(n) \]

is "equivalent" to our original filter.

The question is "How to optimally use all these extra delays to reduce the critical path?"
2) Sample Interleaving (cont.)

Multiple data streams can be multiplexed into one SFG

\[ X_{in}(0), X_{in}(4), X_{in}(2) \]
\[ Y_{in}(0), Y_{in}(1), Y_{in}(2) \]
\[ Z_{in}(0), Z_{in}(1), Z_{in}(2) \]

At clock rate \( f_c \)

\[ \xrightarrow{\text{At clock rate } 3f_c} X_{in}(0), Y_{in}(0), Z_{in}(0), X_{in}(1), Y_{in}(1) \]

This goes into SFG on previous page

3) Changing the latency "usually" (delay from input to output) is not considered important

i.e. \( 2^{-3} \) \( 2^{-1} \) \( 2^{-1} \)

\[ X_{in}(n) \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow 2^{-4} \]
\[ X_{out}(n) \]

Is "equivalent" to our original filter.

The question is "How to optimally use all these extra delays, to reduce the critical path?"
4) Nodal delay transfer

Since $Z^{+1}$ are non-causal they must be removed from the SFG by combining with $Z^{-1}$'s to be implementable
**Definition of a cutset:**

Separates the SFG into two disjoint graphs

\[ X_{in}(n) \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow X_{out}(n) \]

**5) Cutset Retiming**

Uses a generalization of 4 above

Delay can be added to all incoming edges to a cutset if advances are added to all outgoing edges and vice-versa.
5) CUTSET RETIMING

Optimizes the placement of delays to minimize the critical path.
(Certain forms of retiming to decrease the number of registers, reduce power, reduce the clock rate, etc.)

6) LEISERSON-SAXE RETIMING

Developed an algorithm to optimally place the delays to minimize the critical path.

We need to add the time it takes to go through an operator. The critical path is "4" clock rate is then \( \frac{1}{4} \) (in some units).
Since ADDS can be performed in any order we can redraw our filter

\[
\begin{align*}
X_{in}(n) & \xrightarrow{z^{-1}} h_0 \xrightarrow{z^{-1}} h_1 \xrightarrow{z^{-1}} h_2 \xrightarrow{z^{-1}} X_{out}(n) \\
X_{out}(n) & \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} 
\end{align*}
\]

Using these two cutsets add an advance to the top line

\[
\begin{align*}
X_{in}(n) & \xrightarrow{z^{-1}} z^{+1} \xrightarrow{z^{-1}} z^{+1} \xrightarrow{z^{-1}} z^{+1} \\
X_{out}(n) & \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} 
\end{align*}
\]

\[
\begin{align*}
X_{in}(n) & \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} \\
X_{out}(n) & \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} \xrightarrow{z^{-1}} 
\end{align*}
\]

Critical path is now 3

much more improvement for a larger filter.
There are a number of bounds in addition to critical path

Minimum clock period - longest zero delay path

\[ T_{\text{min}} = \sum_i (\text{Top}_i) \]

\( \text{TIB:} \)

1) Iteration bound on clock period

\[ D_L = \text{Total operator delay in each simple loop (no repeating edges)} \]

\[ n_L = \# \text{ of delay's} \]

\[ T_{IB} = \max \left\{ \frac{D_L}{n_L} \right\} \text{loops} \]

(Note: Retiming does not improve iteration bound)

2) Processor (operator) bound

Assume all delays are optimally placed

\[ P_{\text{bound}} = \frac{\sum_i (\text{Top}_i)}{\max \{T_{IB}, T_{min}\}} \]

\( \Rightarrow \) No. of processors needed
\[ M + 1 = \text{total number of digits in code} \]
\[ L = \text{number of nonzero (ternary) digits in code} \]
\[ \{ 0 \} \in \{ 0', 1, 1' \} \text{ and } p_k \in [0, 1', 1'] \]
\[ s_k \in \{ 0', 1, 1' \} \]

where

\[ \sum_{k=1}^{K} \frac{s_k}{2^k} = x \]

Radix-2 canonical signed-digit code representation

substantial hardware reductions

signed-digit (CSD) coefficients result in

For dedicated filtering applications, canonical

Implementation

Multiplierless FIR Filter
(6 addresses vs. 1 address hardware complexity)

127 = 01111111 = 10000001

nonzero digits
numbers to be represented with much fewer

Added flexibility of negative digits allows most

where \( \_ \) denotes -1

111 = 0101 = 0111 = 1111

Example of 4-digit radix-2 SD representation:

3 = 0011

Canonical representation has minimum number
of nonzero digits (also not necessarily unique)

A given number has several signed-digit (SD)

Properties of CSD Codes

UCS/1
Filter response with little performance degradation from the ideal. Result is a low-complexity multiplicityless FIR filter.

Implementation

Example of CSD multiplier coefficient. By simply shifting data busses, power-of-two multiplications are obtained for free.

Multiplicityless FIR Filter
Transpose Direct Form has pipelining "built-in"
FIR Filter Implementations