Pipelining
and
Parallel Processing
cont.
Repetition

- **DSP algorithms are non-terminating** = repeatedly execute same code
- **Iteration** = all operations are executed once
- **Iteration period** = time to perform one iteration
- **Sampling rate (throughput)** = number of samples per second
- **Latency** = time difference between output sample and corresponding input sample (how long before producing output samples)

**Block Diagram** - close to actual hardware – interconnected functional blocks, potentially with delay elements between blocks

**DFG** - Capture the data-driven nature of a DSP system, intra-iteration and inter-iteration constraints, nodes are computations (functions, subtasks), edges are data paths, very general description. Difference from block diagram: Hardware not allocated, scheduled in DFG.
Repetition

- **Critical path** - the combinational path with maximum total execution time
- **Loop (=cycle)** - a path beginning and ending at same node
- **Loop bound for loop**
  \[
  T_j \cdot \frac{T_j}{W_j}
  \]
  - loop computation time
  - number of delays in loop
- **Iteration Bound** - maximum of all loop bounds
  \[
  T_\infty = \max_{l \in L} \left( \frac{t_l}{w_l} \right)
  \]
  It is the lower bound on execution time for DFG (assuming only pipelining, retiming, unfolding)
Repetition

**Pipelining** - insert delay elements to reduce critical path length. In an $M$-level pipelined system, the number of delay elements in any path from input to output is $(M-1)$ greater than that in the same path in the original sequential circuit.

- Faster (more throughput), lower power
- Added latency, latches/clocking
Repetition: Feedforward Cutsets

Feedforward cutset
Must place delays on all edges in the cutset

$$T_{critical} = 3 \rightarrow 5 \rightarrow 4 \rightarrow 6$$
Repetition: Feedforward Cutsets

\[ T_{\text{critical}} = 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \]

Not a Feedforward cutset

Pipelining not possible
Let’s continue…
We said that the critical path could be cut in half by introducing pipelining. Is that always true??
Example: Pipelining when ripple-carry adders

Assume the delay in calculating the sum, $s_{\text{delay}}$, is equal to calculating the carry, $c_{\text{delay}}$.

What is the critical path?

$$4 \times c_{\text{delay}}$$
Example: Pipelining when ripple-carry adders

What is the critical path now?

\[ 4 \times c_{\text{delay}} + s_{\text{delay}} \approx 5 \times c_{\text{delay}} \]
Example: Pipelining when ripple-carry adders

What is the critical path now?

$4 \times c_{\text{delay}}$
Example: Pipelining when ripple-carry adders

What is the critical path now?

\[ 4 \times c_{\text{delay}} \]

Not much gained!
Example: Pipelining when ripple-carry adders

What is the critical path now?

Possible??
Conclusion

The result depends on the structure of the used blocks, e.g. type of adder (ripple-carry, carry save, carry look-ahead,...).

We have to understand how the blocks work and if the critical paths are independent or if there is a relationship.
Let $T_M=10$ units and $T_A=2$ units. If the multiplier is broken into 2 smaller units with processing times of 6 units and 4 units, respectively (by placing the latches on the horizontal cutset across the multiplier), then the desired clock period can be achieved as $(T_M+T_A)/2$. 

Feedforward cutset?
Fine-Grain pipelining

- Equal paths. We get a more balanced design.
- A fine-grain pipelined version of the 3-tap data-broadcast FIR filter is shown below.
Pipelining to Reduce Switching Activity

In chained operations there will be spurious transitions.

Non Pipelined

Pipelined

Results from PhD student Rakesh Gangarajaiah
Synchronous Pipelining

- **Reg**
- **Combinatorial Logic**
- **Reg**

Clock time

Logic depth

- **In**
- **Out**

$t_{\text{min}}$, $t_{\text{max}}$, $t_{\text{min}}$, $t_{\text{max}}$

$0$, $T_{\text{clk}}$

Time

$\text{Logic depth}$

Fredrik Edman, Dept. of Electrical and Information Technology, Lund University, Sweden-ww.eit.lth.se
Wave Pipelining

New input data is applied before the previous computation is done.

\[ T_{\text{clk}} < T_{\text{critical path}} \]
DSP Design

Wave pipelining: Pros and cons?

+ shorter $T_{\text{clk}}$

- extensive simulation
- tedious design
- hard to verify
- lack of tools
Parallell Processing
Parallel Processing

Two samples are processed in parallel
- double throughput
- lower power consumption due to reduced $V_{DD}$
Parallel Processing

- Parallel processing and pipelining techniques are duals of each other: if a computation can be pipelined, it can also be processed in parallel. Both of them exploit concurrency available in the computation in different ways.

- How to design a Parallel FIR system?
  - Consider a single-input single-output (SISO) FIR filter:
    - \( y(n) = ax(n) + bx(n-1) + cx(n-2) \)
  - Convert the SISO system into an MIMO (multiple-input multiple-output) system in order to obtain a parallel processing structure
    - To get a parallel system with 3 inputs per clock cycle
      - \( y(3k) = ax(3k) + bx(3k-1) + cx(3k-2) \)
      - \( y(3k+1) = ax(3k+1) + bx(3k) + cx(3k-1) \)
      - \( y(3k+2) = ax(3k+2) + bx(3k+1) + cx(3k) \)

Parallel processing system is also called **block processing**, and the number of inputs processed in a clock cycle is referred to as the **block size**.
Parallel Processing (cont’d)

- For example:

When block size is 2, 1 delay element = 2 sampling delays

\[ x(2k) \rightarrow D \rightarrow x(2k-2) \]

When block size is 10, 1 delay element = 10 sampling delays

\[ x(10k) \rightarrow D \rightarrow X(10k-10) \]
Ex. Parallel 3-Tap FIR

\[
\begin{aligned}
\begin{cases}
y(n) = b_0 x(n) + b_1 x(n - 1) + b_2 x(n - 2) \\
y(n + 1) = b_0 x(n + 1) + b_1 x(n) + b_2 x(n - 1)
\end{cases}
\end{aligned}
\]

\[
\begin{aligned}
\begin{cases}
y(2k) = b_0 x(2k) + b_1 x(2k - 1) + b_2 x(2k - 2) \\
y(2k + 1) = b_0 x(2k + 1) + b_1 x(2k) + b_2 x(2k - 1)
\end{cases}
\end{aligned}
\]

\(n\) changed to \(2k\)
Parallel 3-Tap (2)

\[
\begin{align*}
    y(2k) &= b_0 x(2k) + b_1 x(2k - 1) + b_2 x(2k - 2) \\
    y(2k + 1) &= b_0 x(2k + 1) + b_1 x(2k) + b_2 x(2k - 1)
\end{align*}
\]
Parallel 3-Tap (3)

\[
\begin{align*}
y(0) &= b_0 x(0) + b_1 x(-1) + b_2 x(-2) \\
y(1) &= b_0 x(1) + b_1 x(0) + b_2 x(-1)
\end{align*}
\]
Parallel 3-Tap (4)

\[
\begin{align*}
  y(0) &= b_0 x(0) + b_1 x(-1) + b_2 x(-2) \\
  y(1) &= b_0 x(1) + b_1 x(0) + b_2 x(-1)
\end{align*}
\]

2 Inputs

\[k=0\]
Parallel 3-Tap (4)

\[
\begin{align*}
y(0) &= b_0 x(0) + b_1 x(-1) + b_2 x(-2) \\
y(1) &= b_0 x(1) + b_1 x(0) + b_2 x(-1)
\end{align*}
\]
Parallel 3-Tap (4)

\[
\begin{align*}
y(0) &= b_0 x(0) + b_1 x(-1) + b_2 x(-2) \\
y(1) &= b_0 x(1) + b_1 x(0) + b_2 x(-1)
\end{align*}
\]
Parallel 3-Tap (5)

\[
\begin{align*}
  y(0) &= b_0 x(0) + b_1 x(-1) + b_2 x(-2) \\
  y(1) &= b_0 x(1) + b_1 x(0) + b_2 x(-1)
\end{align*}
\]

2 Inputs

k=0
Parallel 3-Tap (5)

\[
\begin{align*}
  y(0) &= b_0 x(0) + b_1 x(-1) + b_2 x(-2) \\
  y(1) &= b_0 x(1) + b_1 x(0) - b_2 x(-1)
\end{align*}
\]
Parallel 3-Tap (5)

\[
\begin{align*}
y(0) &= b_0 x(0) + b_1 x(-1) + b_2 x(-2) \\
y(1) &= b_0 x(1) + b_1 x(0) + b_2 x(-1)
\end{align*}
\]
Parallel Processing (cont’d)

- Note: The critical path of the block (or parallel) processing system remains unchanged. But since \( L \) samples are processed in 1 clock cycle, the iteration (or sample) period is given by the following equations:

\[
T_{\text{clock}} \geq T_M + 2T_A \quad \text{for a 3-tap FIR filter}
\]

\[
T_{\text{iteration}} = T_{\text{sample}} = \frac{T_{\text{clock}}}{L}
\]

- So, it is important to understand that in a parallel system

\[
T_{\text{sample}} \neq T_{\text{clock}}, \text{ whereas in a pipelined system } T_{\text{sample}} = T_{\text{clock}}
\]
Parallel Processing (cont’d)

• Why use parallel processing when pipelining can be used equally well?
  – Consider the following chip set: when the critical path is less than the I/O bound (output-pad delay plus input-pad delay and the wire delay between the two chips), we say this system is *communication bounded*

  – So, we know that pipelining can be used only to the extent such that the critical path computation time is limited by the communication (or I/O) bound. Once this is reached, pipelining can no longer increase the speed
Parallel Processing (cont’d)

- So, in such cases, pipelining can be combined with parallel processing to further increase the speed of the DSP system.

- By combining parallel processing (block size: L) and pipelining (pipelining stage: M), the sample period can be reduced to:

\[ T_{\text{iteration}} = T_{\text{sample}} = \frac{T_{\text{clock}}}{L \cdot M} \]

- Parallel processing can also be used for reduction of power consumption while using slow clocks.
Parallel Processing - Converters

- A serial-to-parallel converter

Sample Period T/4

- A parallel-to-serial converter
Pipelining and Parallel Processing for Low Power
Power Dissipation

Two measures are important

• Peak power (sets dimensions)

\[ P_{\text{peak}} = V_{DD} \times I_{DD\text{max}} \]

• Average power (battery and cooling)

\[ P_{\text{av}} = \frac{V_{DD}}{T} \int_{0}^{T} i_{DD}(t) \, dt \]

or rather ”the energy”.
Power consumption in CMOS

\[ P_{\text{total}} = P_{\text{dynamic}} + P_{\text{short-circuit}} + P_{\text{static}} \]

Gaining more importance with technology scaling

The one we will look at and reduce with pipelining and parallelism
CMOS Power Consumption

\[ P_{\text{tot}} = P_{\text{dynamic}} + P_{\text{short-circuit}} + P_{\text{static}} = \]

\[ = \alpha f C_L V_{\text{DD}}^2 + V_{\text{DD}} I_{\text{sc}} + I_{\text{leakage}} V_{\text{DD}} \]

\[ \alpha = \text{probability for switching} \]

\[ C_L = C_{\text{charge}} \]
Short Circuit - Current Spikes

Current peak when both N- and PMOS are open
Static Power Consumption due to leakage current

\[ P_{\text{stat}} = I_{\text{leakage}} \times V_{DD} \]

- Drain Leakage
- Subthreshold Current

\[ I_{\text{leakage}} \] increases with decreasing \( V_T \)
**VT Scaling: VT and I\textsubscript{OFF} Trade-off**

Performance vs Leakage:

\[ \text{VT} \downarrow \Leftrightarrow \text{I}_{\text{OFF}} \uparrow \]

\[ \text{High VT} \quad \text{Low VT} \]

\[ \text{VG} \]

\[ \ln(\text{I}_{\text{DS}}) \]

\[ \text{I}_{\text{OFFL}} \quad \text{I}_{\text{OFFH}} \]

\[ \text{V}_{\text{TL}} \quad \text{V}_{\text{TH}} \]

\[ \text{V}_{\text{G}} \]

\[ \downarrow \text{As VT decreases, sub-threshold leakage increases} \]
Dynamic Power Consumption

Assumed that $V_{\text{swing}} = V_{DD}$, otherwise $V_{\text{swing}} \neq V_{DD}$

Energy charged in a capacitor

$$E_C = CV^2/2 = C_L V_{DD}^2/2$$

Energy $E_c$ is also discharged, i.e.

$$E_{\text{tot}} = C_L V_{DD}^2$$

Power consumption

$$P = C_L V_{DD}^2 f (=C_{\text{tot}} V_0^2 f)$$

Since squared most efficient to reduce $P$
Reduce...

Capacitances
- Transistor/Gate C
- Load C
- Interconnects, more and more important
- External

Activity

Frequency

Power supply – squared so most efficient

...without reducing performance?.
Propagation Delay in CMOS

\[
T_{pd} = \frac{C_L \cdot V_{DD}}{k(V_{DD} - V_T)^2}, \quad k \propto \mu, \frac{W}{L}, C_{ox}
\]

if \( V_{DD} \gg V_T \)

\[
T_{pd} = \frac{C_L}{kV_{DD}} = \frac{1}{f} \quad \text{proportional to} \quad \frac{f}{V_{DD}}
\]
Pipelining, power consumption

\[ x(n) \rightarrow ax(n) \rightarrow abx(n) \]

\[ x(n) \rightarrow ax(n) \rightarrow abx(n-1) \]

**Critical path cut in half**
- double \( f \) \( \Rightarrow \) double throughput
- same \( f \) \( \Rightarrow \) double time for mult \( \Rightarrow \) reduced \( V_{DD} \)

\[ P = f \ C \ V_{DD}^2 \]
Reduction of Critical Path

Propagation delay of the original filter and the pipelined filter

Sequential (critical path):

\[ T_{seq} \]

\[ (V_0) \]

Pipelined: (critical path when \( M=3 \))

\[ T_{pipe} \quad T_{pipe} \quad T_{pipe} \]

\[ (\beta V_0) \]
Pipelining, *power consumption*

- The power consumption in original architecture

\[ P_{\text{seq}} = f \ C_L \ V_{DD}^2 \]

- The supply voltage can be reduced to \( \beta V_{DD} \), \((0 < \beta < 1)\).

Hence, the power consumption of the pipelined filter is:

\[ P_{\text{pipe}} = f \ C_L \ (\beta V_{DD})^2 = \beta^2 P_{\text{seq}} \]
Pipelining

Propagation delays of the sequential and the pipelined architecture:

\[ T_{seq} = \frac{C_L \cdot V_{DD}}{k(V_{DD} - V_T)^2}, \quad T_{pip} = \frac{(C_L/M) \cdot \beta V_{DD}}{k(\beta V_{DD} - V_T)^2} \]

The capacitance in each stage has been reduced.

Since the same \( f \) is maintained \( \Rightarrow T_{seq} = T_{pipe} \)

\[ M(\beta V_{DD} - V_T)^2 = \beta(V_{DD} - V_T)^2 \]
Pipelining

\[ M(\beta V_{DD} - V_T)^2 = \beta(V_{DD} - V_T)^2 \]

\[ T_{add} = 1 \]
\[ T_{mult} = 3 \]

\[ M = ? \]
Pipelining

\[ M(\beta V_{DD} - V_T)^2 = \beta(V_{DD} - V_T)^2 \]

If \( M \) doesn’t divide the critical path evenly you have to use the ”real \( M \)”, i.e. how big is the actual reduction of the critical path.
$P = f \ C \ V^2$

$C = \text{the total switched capacitance}$
Pipelining

Increased C due to register

\[ P_{pipe} = f \times 1.1C \times (0.58V)^2 = 0.37P \]
Power in sub-$V_T$ with Pipelining

AMA = Addition-Multiplication-Addition

MT = Multiplication Tree
Power decreases up to a point and then increase due to increased overhead.

Parallel Processing

Two samples are processed in parallel

- same $f$ but two samples $\Rightarrow$ double throughput

or

- reduce $f$ and two samples $\Rightarrow$ same throughput and reduced $V_{DD}$
Parallel Processing for Low Power

– Total capacitance, $C$, is increased by $L$

– To maintain the same sample rate $f$ is reduced by $1/L$

$-f$ reduced $\Rightarrow V_{DD}$ can be reduced
Parallel Processing

\[ k=0,1,2,3,... \]

\[ x(2k) \quad ax(2k) \quad abx(2k) \quad x(0), x(2), x(4)\ldots \]

\[ x(2k+1) \quad ax(2k+1) \quad abx(2k+1) \quad x(1), x(3), x(5)\ldots \]

\[ P_{\text{para}} = LC_L \frac{f}{L} (\beta V_{DD})^2 = \beta^2 \cdot P_{\text{seq}} \]
Parallel Processing for Low Power

Sequential (critical path):

\[ T_{seq} \]

\[ (V_0) \]

Parallel: (critical path when L=3)

\[ 3T_{seq} \]

\[ (\beta V_0) \]

Propagation delay of the L-parallel system is given by

\[ T_{par} = L \cdot T_{seq} \Rightarrow \frac{C_L \cdot \beta V_{DD}}{k(\beta V_{DD} - V_T)^2} = L \cdot \frac{C_L \cdot V_{DD}}{k(V_{DD} - V_T)^2} \]

\[ L(\beta V_{DD} - V_T)^2 = \beta(V_{DD} - V_T)^2 \]
Parallel Processing

2.15 due to mux

\[ P_{par} = 0.5f \times 2.15C \times (0.58V)^2 = 0.36P \]
Parallel Processing and Pipelining

\[ P_{par,pipe} = 0.5f \times 2.35C \times (0.4V)^2 = 0.19P \]
Summary - The Effect of Pipelining and Parallelization

We have seen that the power can be significantly reduced in a system using pipelining and parallelization.

Pipelining only
\[ P_{\text{pipe}} = f \times 1.1C \times (0.58V)^2 = 0.37P \]

Parallelization only
\[ P_{\text{par}} = 0.5f \times 2.15C \times (0.58V)^2 = 0.36P \]

Pipe- and Parallelization
\[ P_{\text{par,pipe}} = 0.5f \times 2.35C \times (0.4V)^2 = 0.19P \]
End of Lecture 4