

**Design and
Implementation of
Signal Processing
Systems:
An Introduction**

Outline

- Course Objectives and Outline, Conduct
- What is signal processing?
- Implementation Options and Design issues:
 - General purpose (micro) processor (GPP)
 - Multimedia enhanced extension (Native signal processing)
 - Programmable digital signal processors (PDSP)
 - Multimedia signal processors (MSP)
 - Application specific integrated circuit (ASIC)
 - Re-configurable signal processors

Issues in DSP

Architectures and Projects

- Provide students with a global view of embedded micro-architecture implementation options and design methodologies for multimedia signal processing.
- The interaction between the algorithm formulation and the underlying architecture that implements the algorithm will be focused:
 - Formulate algorithm for match architecture.
 - Design novel architecture to match algorithm.

Issues to be treated in projects

- Signal processing computing algorithms
- Algorithm representations
- Algorithm transformations:
 - Retiming, unfolding
 - Folding
- Systolic array and design methodologies
- Mapping algorithms to array structures
- Low power design
- Native signal processing and multimedia extension
- Programmable DSPs
- Very Long Instruction Word (VLIW) Architecture
- Re-configurable computing & FPGA
- Signal Processing arithmetics: CORDIC, and distributed arithmetic.
- Applications: Video, audio, communication

What is Signal?

- A SIGNAL is a measurement of a physical quantity of certain medium.
- Examples of signals:
 - Visual patterns (written documents, picture, video, gesture, facial expression)
 - Audio patterns (voice, speech, music)
 - Change patterns of other physical quantities: temperature, EM wave, etc.
- Signal contains **INFORMATION!**



Medium and Modality

- Medium:
 - Physical materials that carry the signal.
 - Examples: paper (visual patterns, handwriting, etc.), Air (sound pressure, music, voice), various video displays (CRT, LCD)
- Modality:
 - Different modes of signals over the same or different media.
 - Examples: voice, facial expression and gesture.

What is Signal Processing?

- Ways to manipulate signal in its original medium or an abstract representation.
 - Signal can be abstracted as functions of time or spatial coordinates.
- Types of processing:
 - Transformation
 - Filtering
 - Detection
 - Estimation
 - Recognition and classification
 - Coding (compression)
 - Synthesis and reproduction
 - Recording, archiving
 - Analyzing, modeling

Signal Processing Applications

- **Communications:**

- Modulation/Demodulation (modem)
- Channel estimation, equalization
- Channel coding
- Source coding: compression

- **Imaging:**

- Digital camera,
- scanner
- HDTV, DVD

- **Audio**

- 3D sound,
- surround sound

- **Speech**

- Coding
- Recognition
- Synthesis
- Translation

- Virtual reality, animation,

- Control

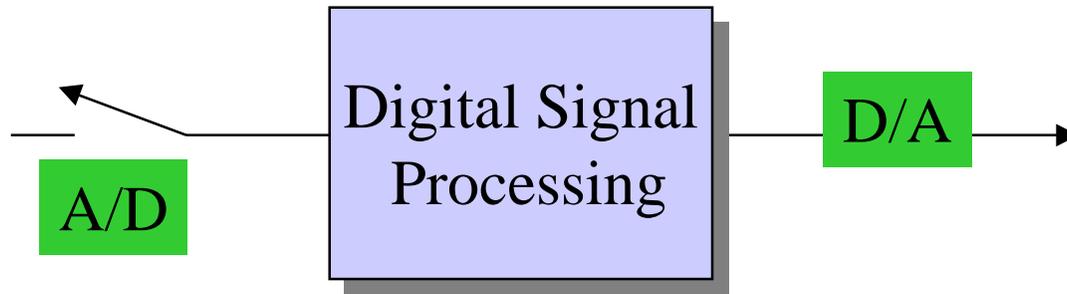
- Hard drive,
- Motor

- Robotics and Intelligent Systems

Digital Signal Processing

- Signals generated via physical phenomenon are *analog* in that
 - Their amplitudes are defined over the range of real/complex numbers
 - Their domains are continuous in time or space.
 - Processing analog signal requires dedicated, special hardware.
- Digital signal processing concerns processing signals using digital computers.
 - A continuous time/space signal must be *sampled* to yield countable signal samples.
 - The real-(complex) valued samples must be quantized to fit into internal word length.

Signal Processing Systems



The task of digital signal processing (DSP) is to process sampled signals (from A/D analog to digital converter), and provide its output to the D/A (digital to analog converter) to be transformed back to physical signals.

Implementation of DSP Systems

• Platforms:

- Native signal processing (NSP) with general purpose processors (GPP)
 - Multimedia extension (MMX) instructions
- Programmable digital signal processors (PDSP)
 - Media processors
- Application-Specific Integrated Circuits (ASIC)
- Re-configurable computing with field-programmable gate array (FPGA)

• Requirements:

- Real time
 - Processing must be done before a pre-specified deadline.
- Streamed numerical data
 - Sequential processing
 - Fast arithmetic processing
- High throughput
 - Fast data input/output
 - Fast manipulation of data

How Fast is Enough for DSP?

- It depends!
- Real time requirements:
 - Example: data capture speed must match sampling rate. Otherwise, data will be lost.
 - Example: in verbal conversation, delay of response can not exceed 50ms end-to-end.
 - Processing must be done by a specific *deadline*.
 - A constraint on *throughput*.
- Different throughput rates for processing different signals
 - Throughput \propto sampling rate.
 - CD music: 44.1 kHz
 - Speech: 8-22 kHz
 - Video (depends on frame rate, frame size, etc.) range from 100s kHz to MHz.

Early Signal Processing Systems

- Implemented with either **main frame computer** or special purpose computers.
- **Batch processing** rather than real time, streamed data processing.
- Accelerate **processing speed** is of main concern.
- **Key approach:**
 - Faster hardware
 - Faster algorithms
- **Faster algorithms**
 - Reduce the number of arithmetic operations
 - Reduce the number of bits to represent each data
 - Most important example: Fast Fourier Transform

Computing Fourier Transform

Discrete Fourier Transform

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp\left[-\frac{2\pi nk}{N}\right]$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \exp\left[\frac{2\pi nk}{N}\right]$$

- To compute the N frequencies $\{X(k); 0 \leq k \leq N-1\}$ requires N^2 complex multiplications

- **Fast Fourier Transform**

- Reduce the computation to $O(N \log_2 N)$ complex multiplications
- Makes it practical to process large amount of digital data.
- Many computations can be “Speed-up” using FFT
- Dawn of modern digital signal processing

Evolution of Micro-Processor

- Micro-processors implemented a central processing unit on a single chip.
- **Performance** improved from 1MFLOP (1983) to 1GFLOP or above
- **Word length** (# bits for register, data bus, addr. Space, etc) increases from 4 bits to 64 bits today.
- **Clock frequency** increases from 100KHz to 1GHz
- **Number of transistors** increases from 1K to 50M
- **Power consumption** increases much slower with the use of lower supply voltage: 5 V drops to 1.5V

Native Signal Processing

- Use GPP to perform signal processing task with no additional hardware.
 - Example: soft-modem, soft DVD player, soft MPEG player.
- Reduce hardware cost!
- May not be feasible for extremely high throughput tasks.
- Interfering with other tasks as GPP is tied up with NSP tasks.
- **MMX (multimedia extension instructions):** special instructions for accelerating multimedia tasks.
- May share same data-path with other instructions, or work on special hardware modules.
- Make use sub-word parallelism to improve numerical calculation speed.
- Implement DSP-specific arithmetic operations, eg. *Saturation arithmetic ops.*

ASIC: Application Specific ICs

- Custom or semi-custom IC chip or chip sets developed for specific functions.
- Suitable for high volume, low cost productions.
- **Examples:** MPEG codec, 3D graphic chip, etc.
- ASIC becomes popular due to availability of IC foundry services.
- *Fab-less* design houses turn innovative design into profitable chip sets using **CAD tools**.
- Design automation is a key enabling technology to facilitate fast design cycle and shorter time to market delay.

Programmable Digital Signal Processors (PDSPs)

- Micro-processors designed for signal processing applications.
- Special hardware support for:
 - **Multiply-and-Accumulate (MAC)** ops
 - Saturation arithmetic ops
 - Zero-overhead loop ops
 - Dedicated data I/O ports
 - Complex address calculation and memory access
 - Real time clock and other embedded processing supports.
- PDSPs were developed to fill a market segment between GPP and ASIC:
 - GPP flexible, but slow
 - ASIC fast, but inflexible
- As VLSI technology improves, role of PDSP changed over time.
 - Cost: design, sales, maintenance/upgrade
 - Performance

Multimedia Signal Processors

- Specialized PDSPs designed for multimedia applications
- **Features:**
 - Multi-processing system with a GPP core plus multiple function modules
 - VLIW-like instructions to promote instruction level parallelism (ILP)
 - Dedicated I/O and memory management units.
- **Main applications:**
 - Video signal processing, MPEG, H.324, H.263, etc.
 - 3D surround sound
 - Graphic engine for 3D rendering

Re-configurable Computing using FPGA

- FPGA (Field programmable gate array) is a derivative of PLD (programmable logic devices).
- They are *hardware configurable* to behave differently for different configurations.
- Slower than ASIC, but faster than PDSP.
- Once configured, it behaves like an ASIC module.
- Use of FPGA
 - Rapid prototyping: run fractional ASIC speed without fab delay.
 - Hardware accelerator: using the same hardware to realize different function modules to save hardware
 - Low quantity system deployment

Characteristics and Impact of VLSI

- The term VLSI (Very Large Scale Integration) is coined in late 1970s.
- **Usage of VLSI:**
 - Micro-processor
 - General purpose
 - Programmable DSP
 - Embedded μ -controller
 - Application-specific ICs
 - Field-Programmable Gate Array (FPGA)
- **Impacts:**
 - Design methodology
 - Performance
 - Power

- **Characteristics**
 - High density:
 - Reduced feature size: 0.25 μ m -> 0.16 μ m
 - % of wire/routing area increases
 - Low power/high speed:
 - Decreased operating voltage: 1.8V -> 1V
 - Increased clock frequency: 500 MHz-> 1GH.
 - High complexity:
 - Increased transistor count: 10M transistors and higher
 - Shortened time-to-market delay: 6-12 months

Design Issues

- Given a DSP application, which implementation option should be chosen?
- For a particular implementation option, how to achieve optimal design? Optimal in terms of what criteria?
- **Software design:**
 - NSP/MMX, PDSP/MSP
 - Algorithms are implemented as programs.
 - Often still require programming in assembly level manually
- **Hardware design:**
 - ASIC, FPGA
 - Algorithms are directly implemented in hardware modules.
- **S/H Co-design:** System level design methodology.

Design Process Model

- Design is the process that links *algorithm* to *implementation*
- Algorithm
 - Operations
 - Dependency between operations determines a partial ordering of execution
 - Can be specified as a *dependence graph*
- Implementation
 - *Assignment*: Each operation can be realized with
 - One or more instructions (software)
 - One or more function modules (hardware)
 - *Scheduling*: Dependence relations and resource constraints leads to a *schedule*.

A Design Example ...

Consider the algorithm:

$$y = \sum_{k=1}^n a(k)x(k)$$

Program:

$y(0) = 0$

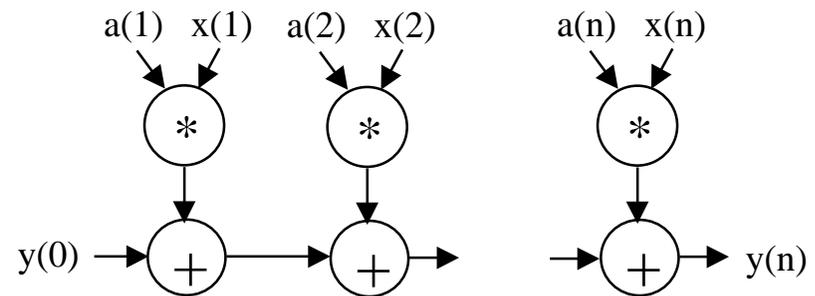
For $k = 1$ to n Do

$y(k) = y(k-1) + a(k)*x(k)$

End

$y = y(n)$

- Operations:
 - Multiplication
 - Addition
- Dependency
 - $y(k)$ depends on $y(k-1)$
 - Dependence Graph:



Design Example cont'd ...

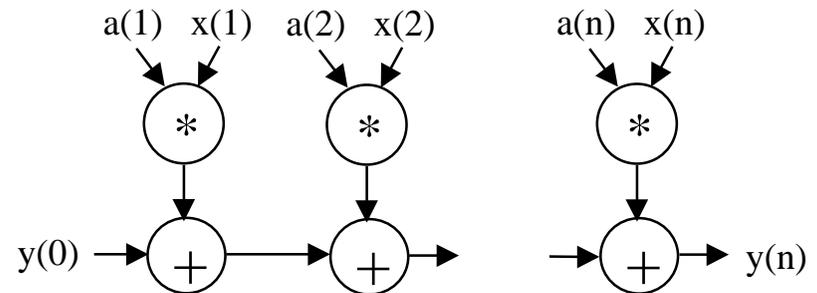
- **Software**

- **Implementation:**

- Map each * op. to a MUL instruction, and each + op. to a ADD instruction.
- Allocate memory space for $\{a(k)\}$, $\{x(k)\}$, and $\{y(k)\}$
- Schedule the operation by sequentially execute $y(1)=a(1)*x(1)$, $y(2)=y(1) + a(2)*x(2)$, etc.
- Note that each instruction is still to be implemented in hardware.

- **Hardware Implementation:**

- Map each * op. to a multiplier, and each + op. to an adder.
- Interconnect them according to the dependence graph:



Observations

- Eventually, an implementation is realized with hardware.
- However, by using the same hardware to realize different operations at different time (scheduling), we have a software program!

- **Bottom line** – Hardware/ software co-design. There is a continuation between hardware and software implementation.
- A **design must explore** both simultaneously to achieve best performance/cost trade-off.

A Theme

- **Matching hardware to algorithm**

- Hardware architecture must match the characteristics of the algorithm.
- Example: ASIC architecture is designed to implement a specific algorithm, and hence can achieve superior performance.

- **Formulate algorithm to match hardware**

- Algorithm must be formulated so that they can best exploit the potential of architecture.
- Example: GPP, PDSP architectures are fixed. One must formulate the algorithm properly to achieve best performance. Eg. To minimize number of operations.

Algorithm Reformulation

- **Matching algorithm** to architectural features
 - Similar to optimizing assembly code
 - Exploiting equivalence between different operations
- **Reformulation methods**
 - Equivalent ordering of execution:
 - $(a+b)+c = a+(b+c)$
 - Equivalent operation with a particular representation:
 - $a*2$ is the same as left-shift a by 1 bit in binary representation
 - Algorithmic level equivalence
 - Different filter structures implementing the same specification!

Algorithm Reformulation (2)

- Exploiting parallelism
 - Regular iterative algorithms and loop reformulation
 - Well studied in parallel compiler technology
 - Signal flow/Data flow representation
 - Suitable for specification of pipelined parallelism

Mapping Algorithm to Architecture

- **Scheduling and Assignment Problem**
 - Resources: hardware modules, and time slots
 - Demands: operations (algorithm), and throughput
- **Constrained optimization problem**
 - Minimize resources (objective function) to meet demands (constraints)
- For regular iterative algorithms and regular processor arrays --> algebraic mapping.

Mapping Algorithms to Architectures

- Irregular multi-processor architecture:
 - linear programming
 - Heuristic methods
 - Algorithm reformulation for recursions.
- Instruction level parallelism
 - MMX instruction programming
 - Related to optimizing compilation.

Arithmetic

- CORDIC
 - Compute elementary functions
- Distributed arithmetic
 - ROM based implementation
- Redundant representation
 - eliminate carry propagation
- Residue number system

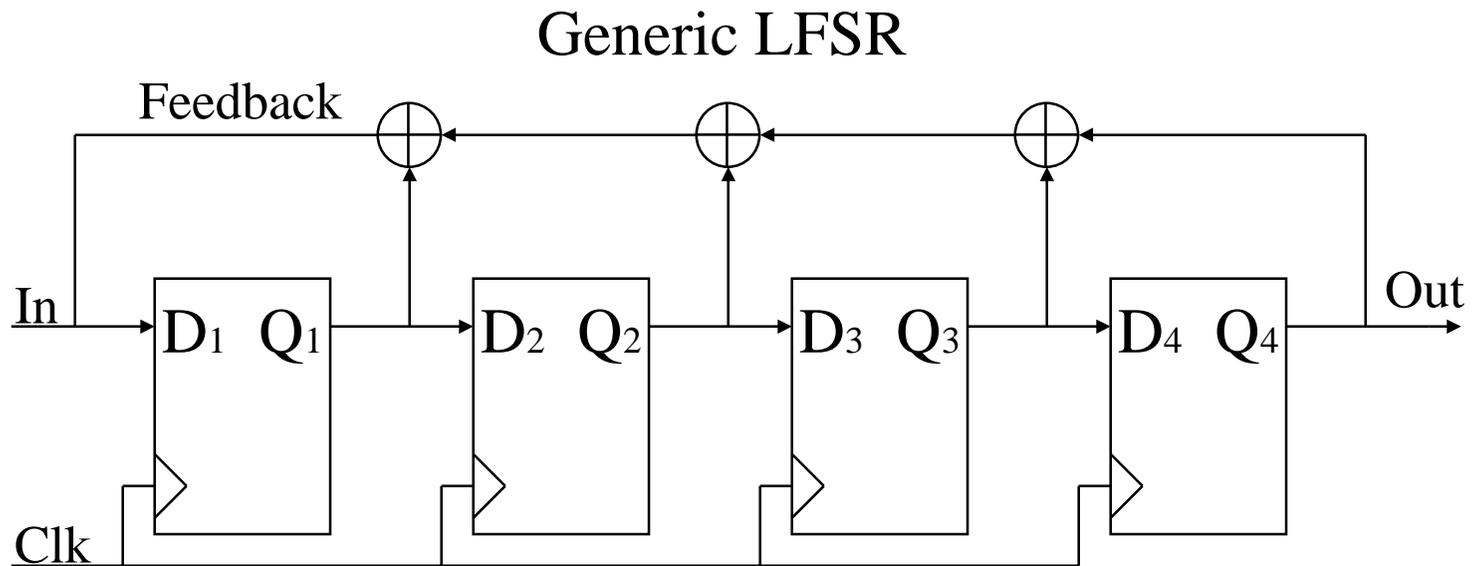
Low Power Design

- Device level low power design
- Logic level low power design
- Architectural level low power design
- Algorithmic level low power design

What is an LFSR & MISR circuit?

- LFSR & MISR (Linear Feedback Shift Register & Multiple Input Signature Register) circuits are two types of a specially connected series of flip flops with some form of XOR/XNOR feedback.
- They are used in many applications for the generation or detection of Pseudo Random Sequences.

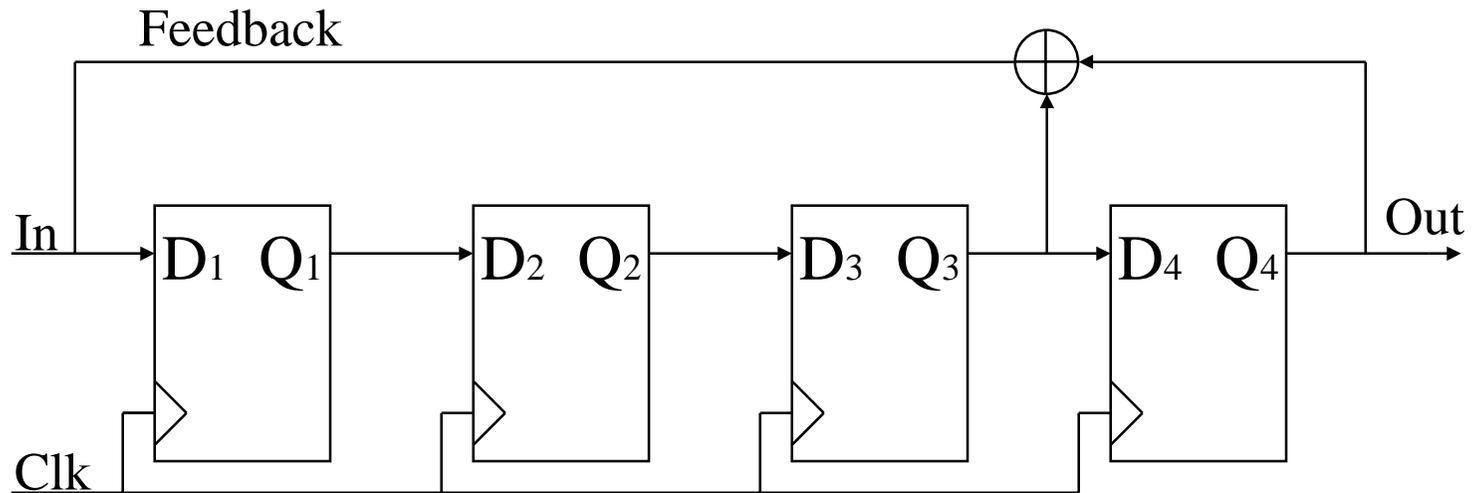
LFSR Block Diagram



LFSR Block Diagram (cont.)

By Changing the Feedback path to “tap” only certain FF’s, a Maximal Length Sequence can be produced.

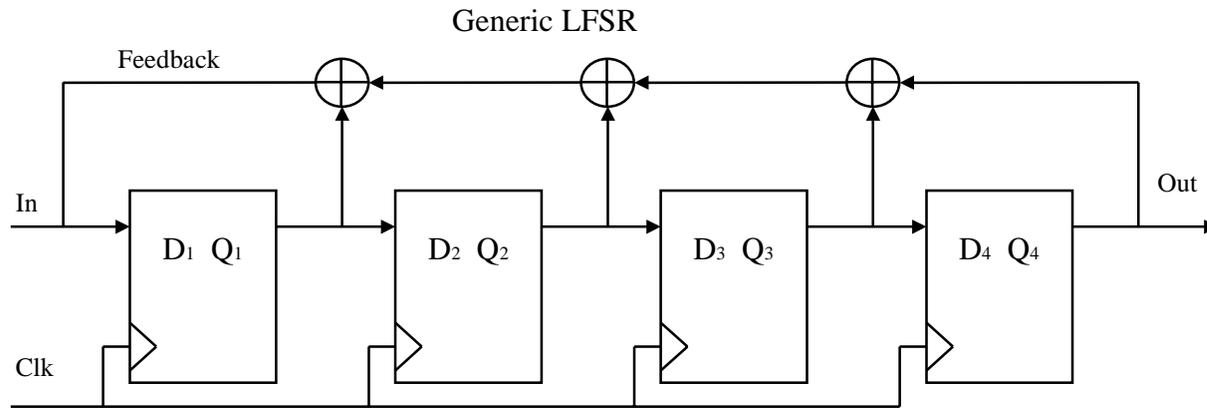
Maximal Length LFSR (n = 4)



Polynomial: $1 + x^3 + x^4$

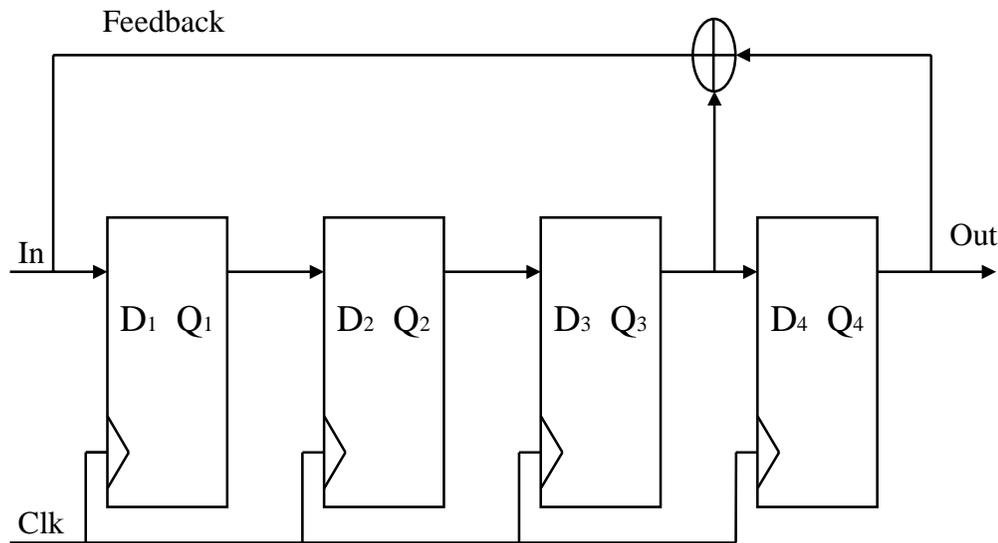
Maximal Length: $(2^n - 1) = (2^4 - 1) = (16 - 1) = 15$

Problems with this type of LFSR



- Setup Time - Feedback for D₁ has to go through N XORs before arriving. N Logic delays slows down circuit performance (may need to run “at speed”). Solution is to have many-input XOR feeding D₁ input (1 logic level).
- State 000 is illegal. When FFs power up, they must be initialized with valid data. Solution is to use XNORs instead. Still produces a PRBS but all zeros is a valid state.

Maximal Length Sequence

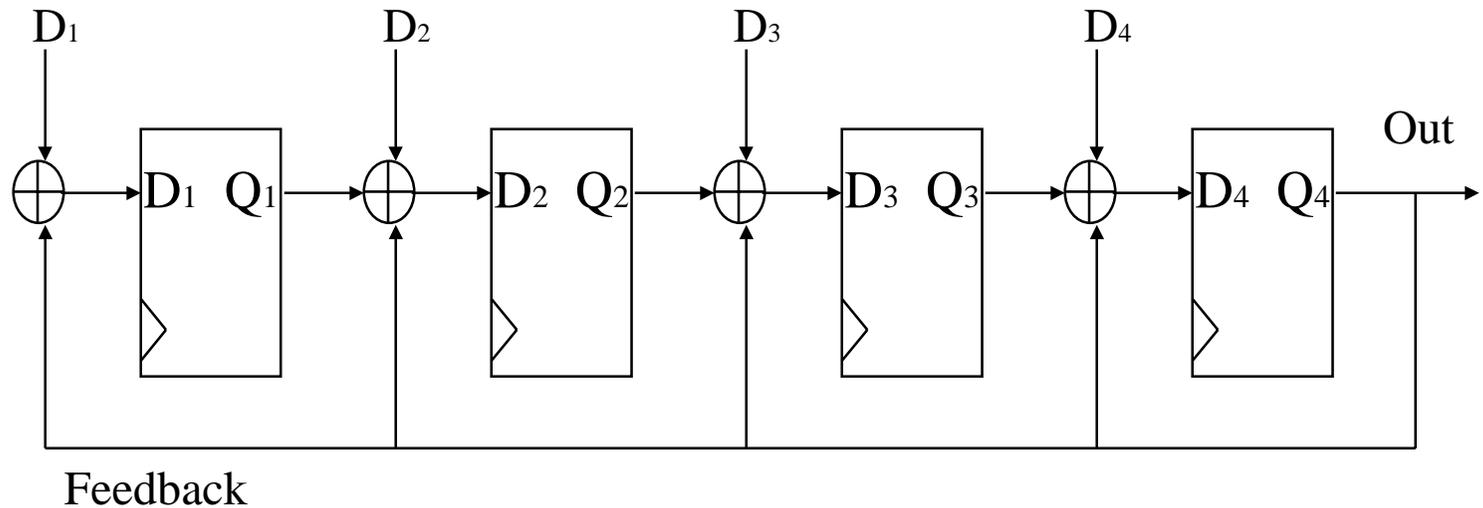


Output Sequence:
100010011010111,10001...

State	FF 1	FF 2	FF 3	FF 4
S0	0	0	0	1
S1	1	0	0	0
S2	0	1	0	0
S3	0	0	1	0
S4	1	0	0	1
S5	1	1	0	0
S6	0	1	1	0
S7	1	0	1	1
S8	0	1	0	1
S9	1	0	1	0
S10	1	1	0	1
S11	1	1	1	0
S12	1	1	1	1
S13	0	1	1	1
S14	0	0	1	1
S15=S0	0	0	0	1
S16=S1	1	0	0	0

MISR Block Diagram

Generic MISR



Multiple Inputs (4-bit wide): $\{D_1, D_2, D_3, D_4\}$

LFSR & MISR Applications:

- BIST (Built-in Self Test) of logic devices.
- Cyclic Encoding/Decoding (Cyclic Redundancy Check)
- Pseudo Noise Generator
- Pseudo Random Binary Sequence Generator
- Spread Spectrum (CDMA) applications

Built-In Self Test (BIST)

- Devices can be self-tested (at speed) by incorporating LFSR and MISR circuits into the design. **Testing** can occur while the device is operating or while in an idle mode.
- An LFSR generates a Pseudo-Random Test Pattern. A small LFSR with the appropriate feedback can generate **very long sequences** of apparently random data.

Built-In Self Test (BIST) (cont.)

- The Pseudo-Random pattern that is generated by the LFSR is feed through the logic under test then into the MISR.
 - The MISR will essentially compare the result with a known “good” signature.
 - If the result is the same, then there were no errors in the logic.
- Refer to Dr. Perkowski’s Built-In Self Test Presentation in Test Class for more information.

Spread Spectrum PRBS

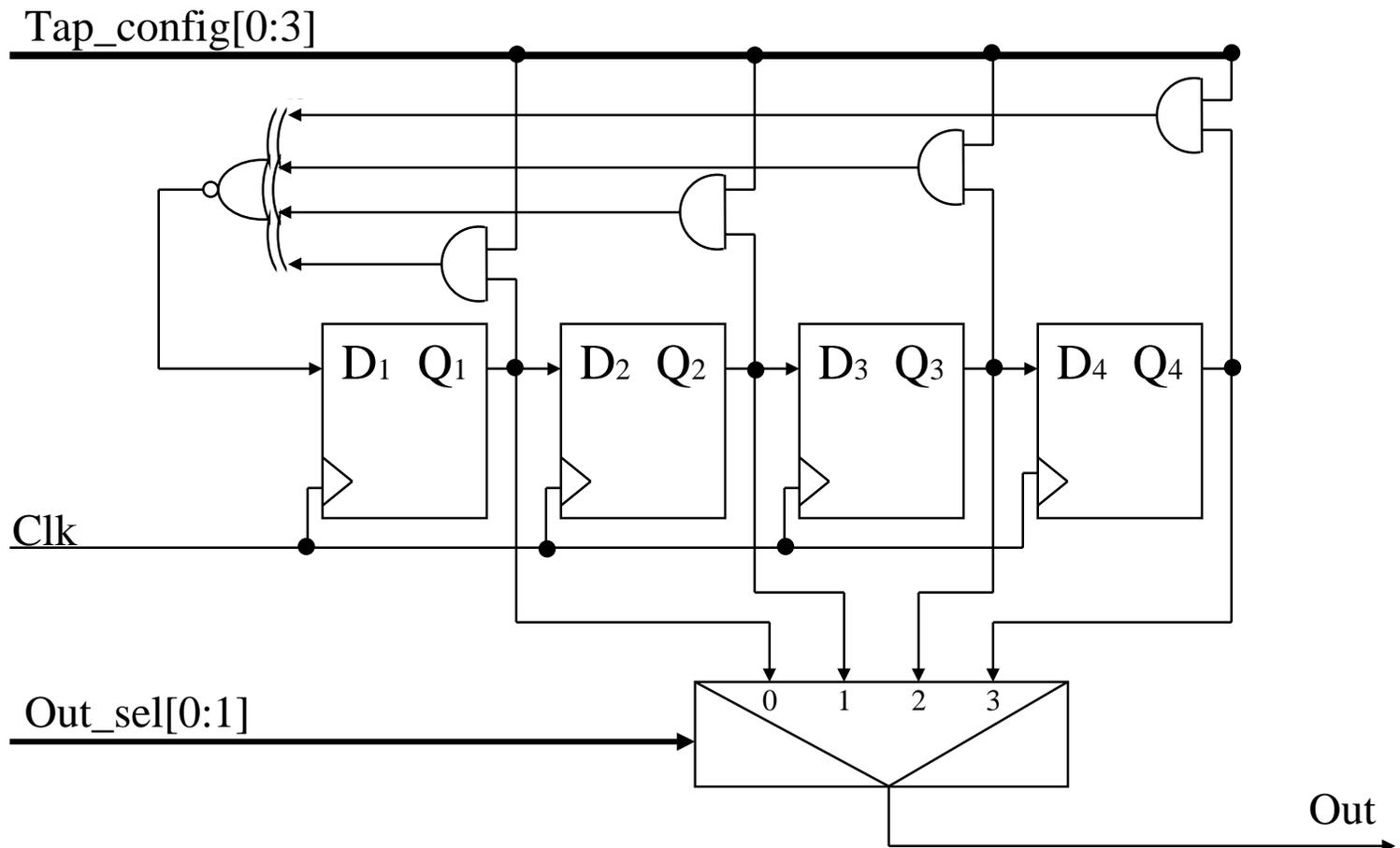
- Because PN signals have good auto-correlation, they are used in Code Division Multiple Access Spread Spectrum Communication Systems.
- Pseudo Random Noise Sequences are used to effectively “spread” the overall bandwidth of a CDMA signal.
- For every data bit that is to be transmitted, a PRNS is substituted. The Information rate remains the same, but the new bit rate is dramatically increased.

1 -> 100010011010111...

0 -> 011101100101000...

Spread Spectrum PRBS (cont.)

- Below is a diagram showing an efficient arbitrary PRBS generator.
- By modifying Tap_config[0:3] and selecting the proper output, this circuit can generate many different Pseudo Random Binary Sequences.



Practical LFSR and MISR circuits

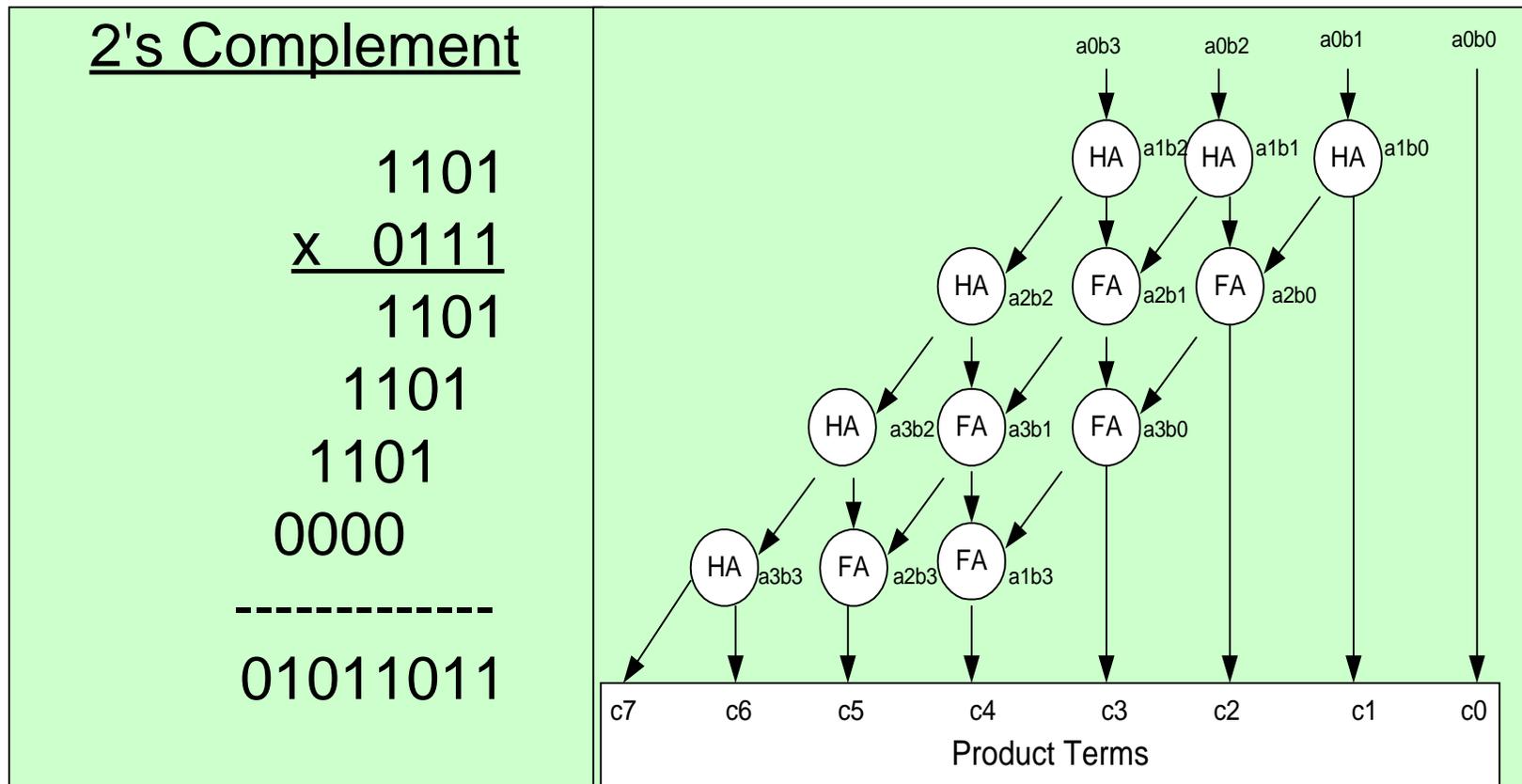
- LFSR and MISR circuits are used in many applications.
- As technology continues to advance, more and more devices will be developed that will utilize the unique properties of these powerful circuits.
- Built-In Self Test and Spread Spectrum (CDMA) applications are but a few of the many places where LFSR and MISR circuits are used.

Practical Combinational Multipliers

What is a combinational multiplier?

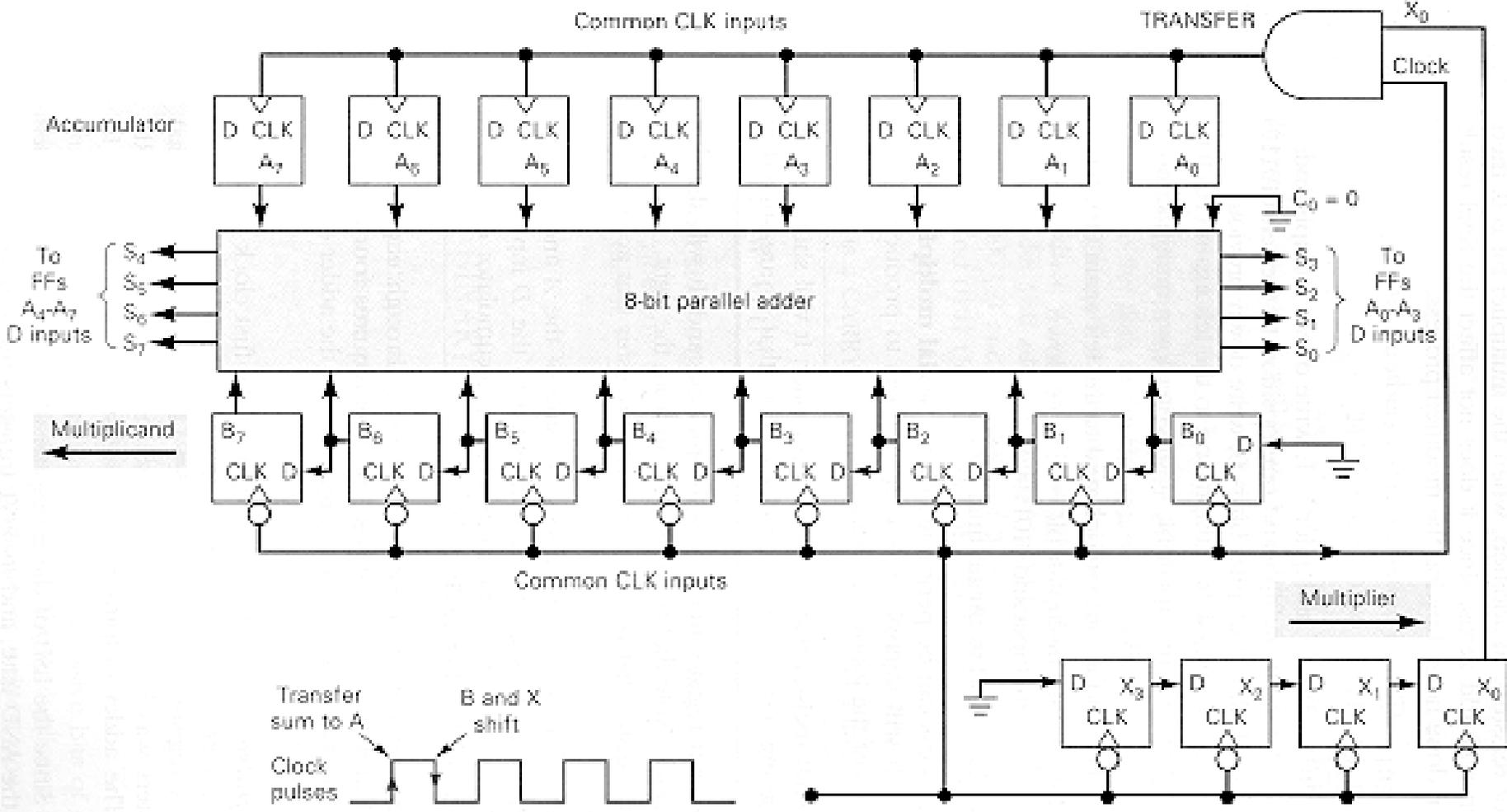
- A combinational multiplier circuit is comprised of multiple shift registers, an adder, and some control logic.
- A multiply is performed by addition and shifting.
- Typical generic multipliers are slow, often taking multiple clock cycles to compute a product.
- Computers without dedicated multipliers must perform a multiply using this method.

Example: 4-bit Multiply



FA= Full Add
HA=Half Add

Generic Serial Multiplier Block Diagram



A register triggers on *positive* clock transition.
 B and X registers trigger on *negative* edge.

Digital Systems Principals and Applications, Ronald J. Tocci, Prentice Hall 1995, pg 280

Figure 6-16 Circuit for a 4-bit serial multiplier that uses shift registers for the multiplier and multiplicand.

So what's wrong with this type of multiplier?

- For an $N \times N$ generic Multiplier, it takes N clock cycles to get a product. **That's too slow!**
- Inefficient use of hardware.

Types of Multipliers

- **Standard Binary Multiplier** (ones complement, twos complement, universal, etc...)
- **Re-coded Multipliers** (Canonical Signed Digit, Booth, etc...)
- **Serial / Parallel Multipliers**
- **Iterative Cellular Array Multipliers** (Wallace, Pezaris, Baugh-Wooley, etc...)
- **ROM based Multiplication Networks** (Constant Coefficient Multipliers, Logarithmic, etc...)

Multiplier Applications

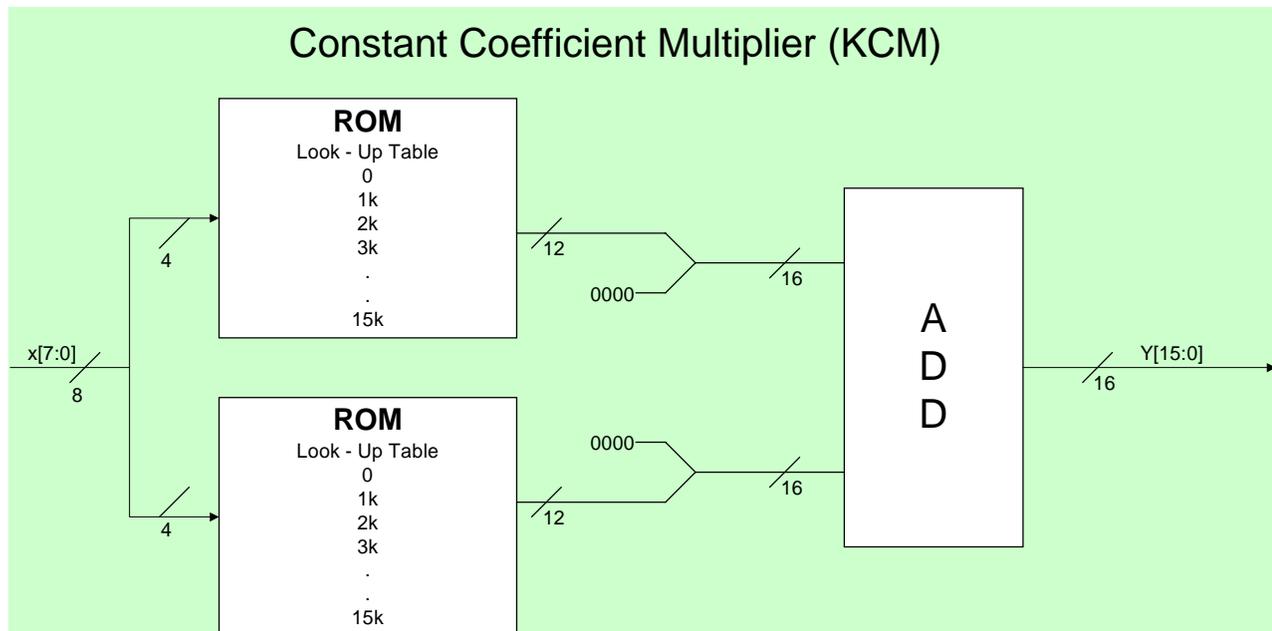
- General Purpose Computing
- Digital Signal Processing
 - Finite Impulse Response Filters
 - Convolution

ROM Based Constant Coefficient Multiplier

- With some DSP applications, such as **FIR filter** generation and **convolution**, where the coefficients remain unchanged and high speed is a requirement, using a look-up table approach to multiplication is quite common.
- Using the known coefficients, every possible product is calculated and programmed into a look-up table. (**ROM or RAM**)
- The unknown multiplicand (input data) is used as an address to “look up” the product.
- This method results in **very high speed multiplies**, however it requires large amounts of storage space.

ROM Based Constant Coefficient Multiplier (cont.)

- Uses ROM to generate partial product
- Sum all partial product ROM outputs



Practical Combinatorial Multipliers

- Generic Shift/Add type multipliers are SLOW!
- People will always be searching for methods of performing **faster multiplies**.
- Multipliers are used in many areas.
- General purpose math for PCs and DSP (FIR filters, Convolution, etc...) applications are just a few of the places where multipliers are utilized.

References

- Digital Systems Principals and Applications, Ronald J. Tocci, Prentice Hall 1995, pg 278-282
- Xilinx Application Note (XAPP 054). Constant Coefficient Multipliers for XC4000E. <http://www.xilinx.com/xapp/xapp054.pdf>
- Altera Application Note (AN 82). Highly Optimized 2-D convolvers in FLEX Devices. http://www.altera.com/document/an/an082_01.pdf
- Computer Arithmetic Principles, Architecture, and Design, Kai Hwang, John Wiley & Sons, Inc. 1979, pg129-212

References

- Dr. Perkowski. Design for Testability Techniques (Built-In Self-Test) presentation.
http://www.ee.pdx.edu/~mperkows/CLASS_TEST_99/BIST.PDF
- Digital Communications Fundamentals and Applications, Bernard Sklar, Prentice Hall 1988, Pg 290-296, Pg 546-555
- Xilinx Application Note (XAPP 052). Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators.
<http://www.xilinx.com/xapp/xapp052.pdf>
- Sun Microsystems' sponsored EDACafe.com website. Chapter 14 - Test. <http://www.dacafe.com/Book/CH14/CH14.htm>

Sources

- Yu Hen Hu
- Andrew Iverson, ECE 572