FPGA Spread Spectrum Modulator

EE572 Project, Fall 1999

Steven Harwood

Andrew Iverson

Presentation Outline

- What is it?
- Why do we want to build it, AKA "Motivation!"
- Design Goals
- Implementation
- Forward Plans

What is it: Outline

- Goal of the device
- 30 second tour of Spread Spectrum
- Modulator block diagram
  - Input data handler/memory interface
  - Spreading code generator: PRBS
  - Baseband FIR Filter

Goal of Device

- A spread spectrum modulator for arbitrary spreading sequence
  - Input is Encoded Data Stream; protocol issues have been handled “upstream”
  - Outputs are in-phase (I) and quadrature-phase (Q) words at output sample rate,
  - Outputs go to Digital to Analog conversion, on the way to RF chain
- Design entire modulator into an FPGA

30 second tour of Multiple Access Communications, Spread Spectrum

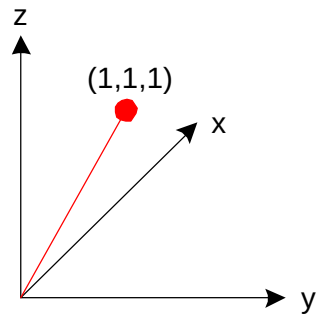
Orthogonality

- Given a funtion, f(t), we often want to specify the function by a set of numbers not dependent on the variable t. Express as

$$f(t) = \Sigma f_n \phi_n(t)$$

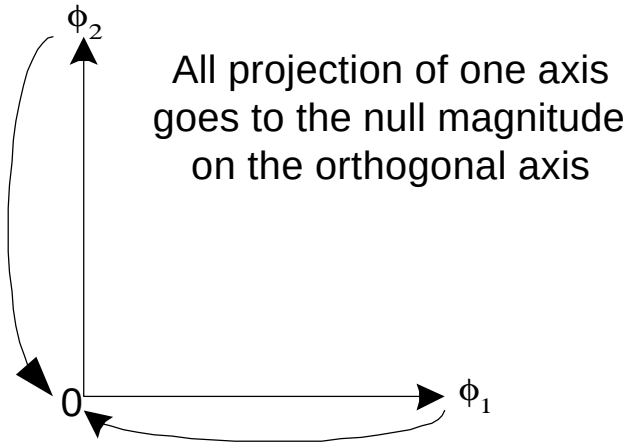
where the chosen $\phi_n(t)$ is known as a basis function for f(t); f_n is a set of numbers independent of time.

- This is analgous to expressing a vector as the sum of a set of numbers in a Cartesian coordinate system: (1,1,1) = 1 • x + 1 • y + 1 • z



Orthogonality

- As with a Cartesian coordinate system , the axes are chosen such that the projection of the vector on each coordinate axis is independent of the projection of the vector on other axes.
- When the vector $\phi_1(t)$ has no component along vector $\phi_2(t)$, the two vectors are perpendicular and are defined to be *orthogonal*.

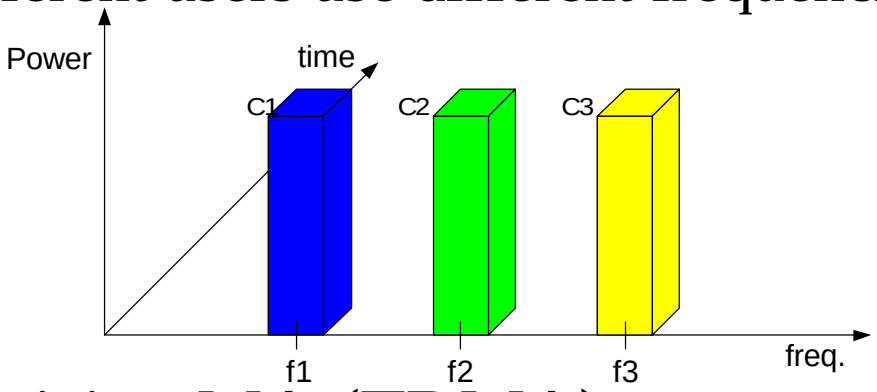


Orthogonality

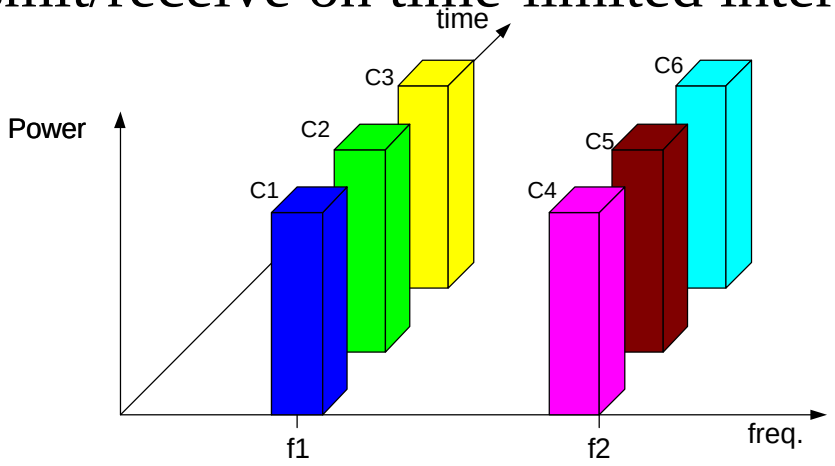
- In general terms, there are many sets of orthogonal functions. Examples of orthogonal vector sets are Cartesian coordinates, polar coordinates, time allocated events, trigonometric functions, exponential functions.
- Often it is advantageous to normalize the axes of a set of basis functions to unity. When an orthogonal set of basis functions are normalized, they are *orthonormal*.
- Orthogonal systems play a vital role in communications. By requiring different communication channels to be orthogonal, multiple users may access a common communications domain.

Multiple Access Communications

- Frequency Division MA (FDMA)
  - Orthogonal in Frequency
    - Different users use different frequencies

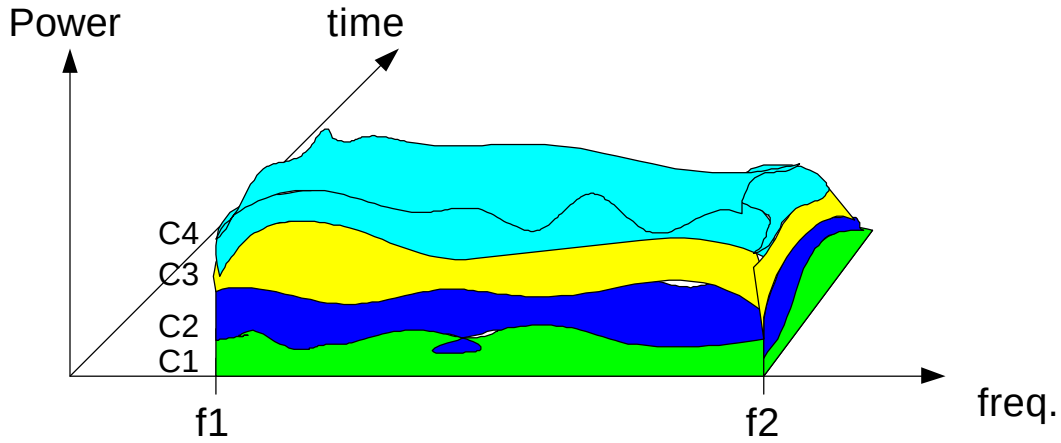


- Time Division MA (TDMA)
  - Orthogonal in Time and Frequency
    - Different users use different frequencies, but only transmit/receive on time-limited intervals.

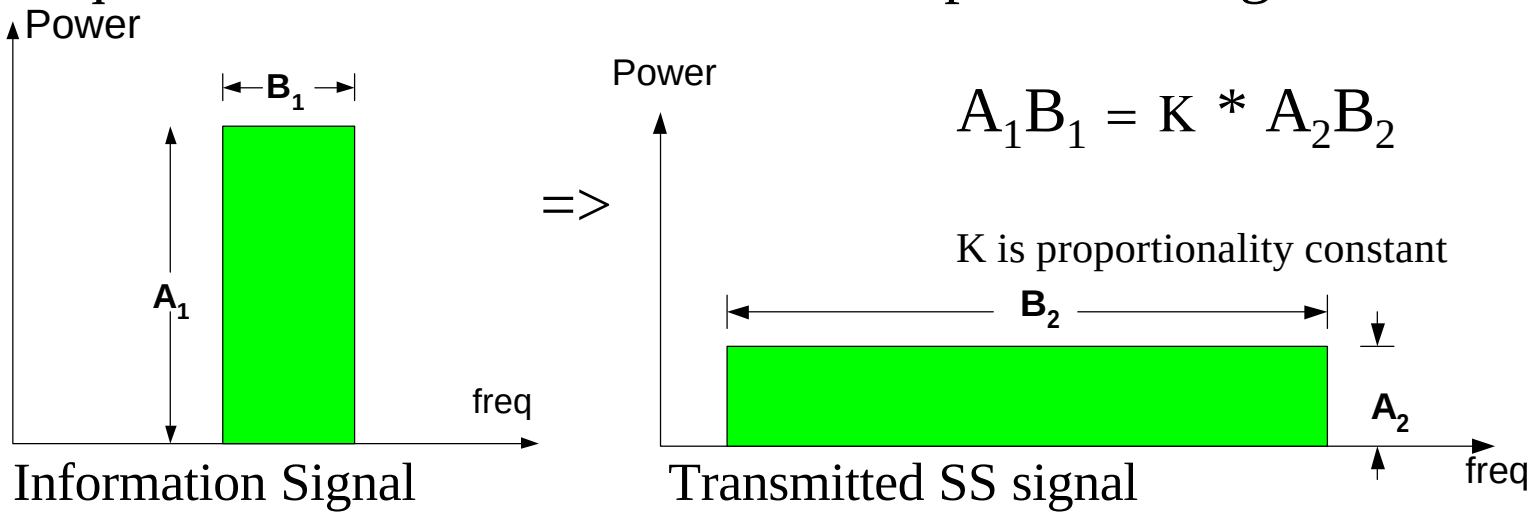


Multiple Access Communications

- Code Division MA (CDMA)
  - Orthogonal in Coding, all user's share a frequency band

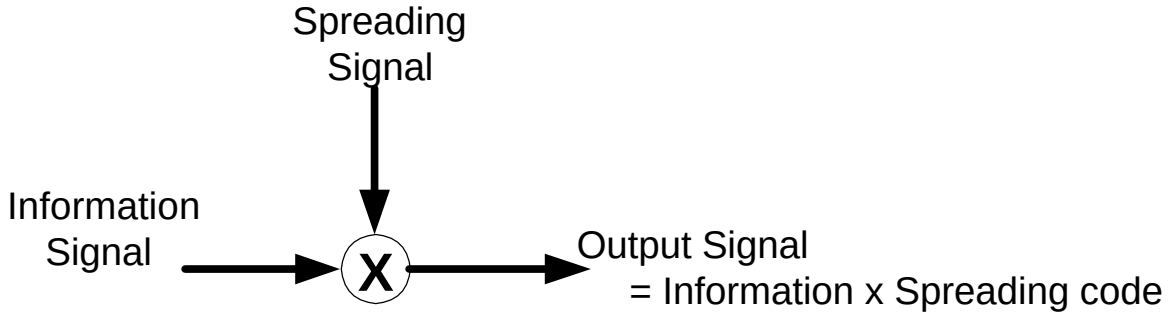


  - Transmit signal in a bandwidth much greater than the required information bandwidth: "spread the signal"



Spread Spectrum

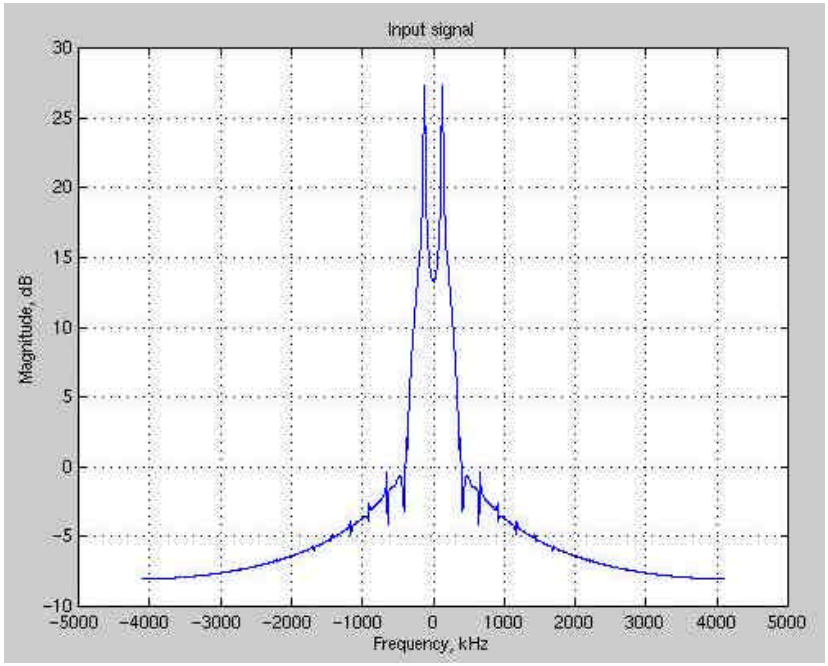
- Incoming data is modulated (multiplied) with a spreading sequence
  - Spreading signal typically much higher rate than information signal



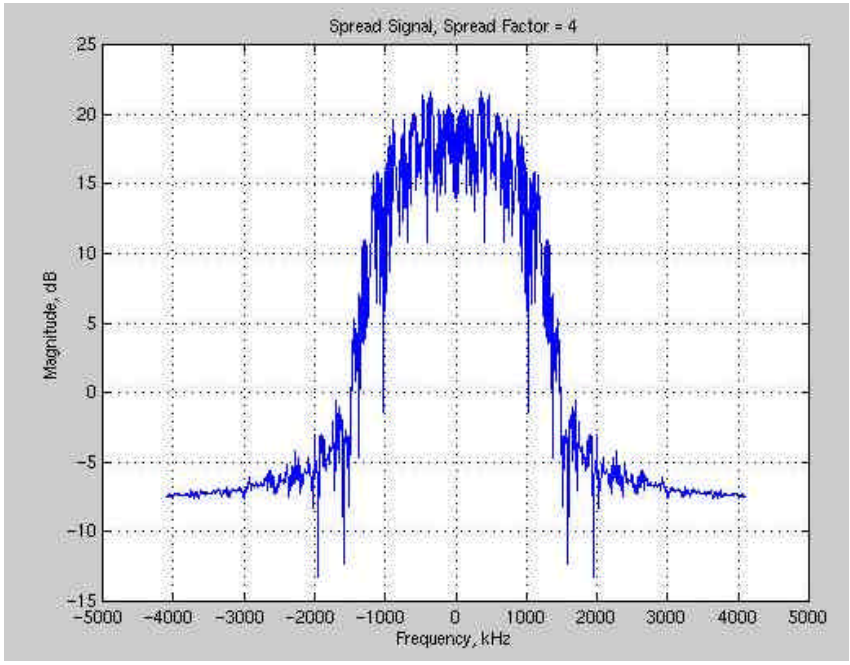
  - The modulating operation is a bitwise Exclusive-OR.
  - The spreading signal is clocked at ***n*** times the incoming information signal data rate, where n is an integer, ***n>1***. Common values for ***n*** = 128, 512

Spread Spectrum

- Spreading a data signal example
  - Spreading Factor of 4 shown here



Spectrum of Original Signal



Spectrum of Spread Signal

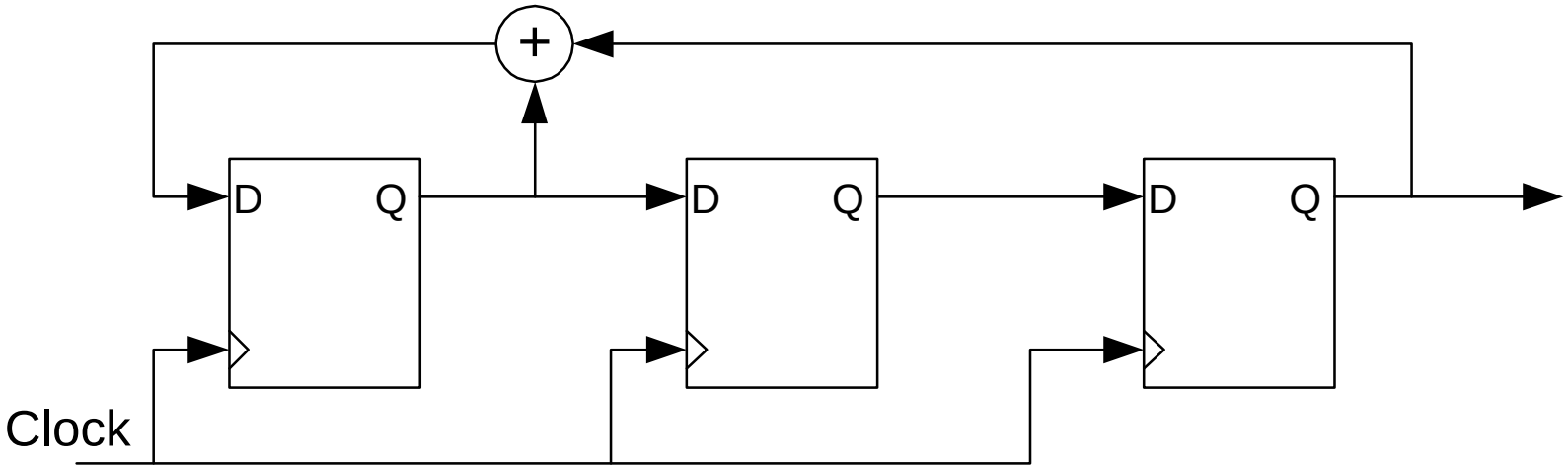
Spread Spectrum

- Spreading Codes generated by pseudo-random binary sequence (PRBS) generator
- PRBS is a stream of “random” 1’s and 0’s that repeats at known intervals
  - Generated in hardware by shift registers with XOR feedback
  - PRBS Length dictated by length of shift register chain and feedback taps
  - A maximal length sequence has length = 2^n -1 states before repeating, where “n” is the length of the shift register

Spread Spectrum

- Example PRBS: f(x) = x3 + x + 1 , length=3,
  number of states= 7

  111 ⇒ 110 ⇒ 101⇒ 010 ⇒ 100 ⇒ 001⇒ 011⇒ 111...



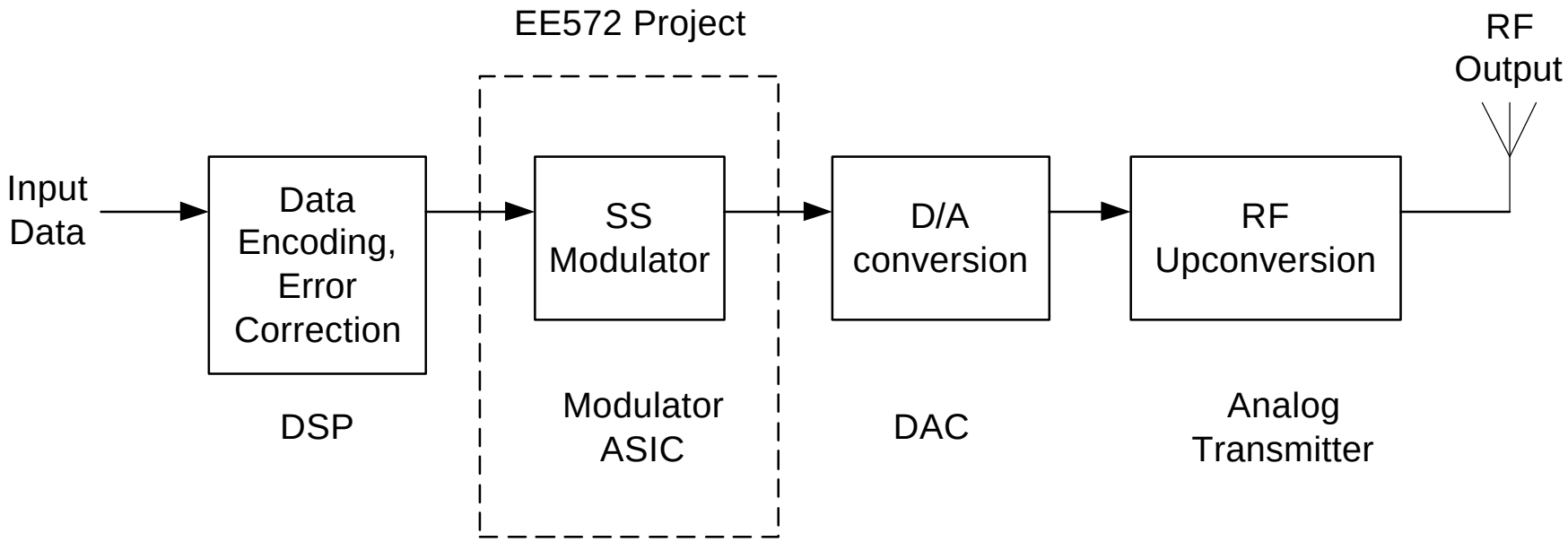
Spread Spectrum

- Commercial SS uses a pre-specified PRBS
- Orthogonality is achieved by having all users codes' appear unique
  - Achieved by using offset versions of the same spreading code
    - user 1 will use seed value "A", user 2 will use seed value "B"
    - user 1's version of PRBS always looks different from user 2's version of PRBS, orthogonality is maintained.
    - In commercial systems, there are a finite number of offset positions per common channel. Ex. IS-95 has 512 different offset positions per cell.
  - COTS hardware available for specific spreading codes

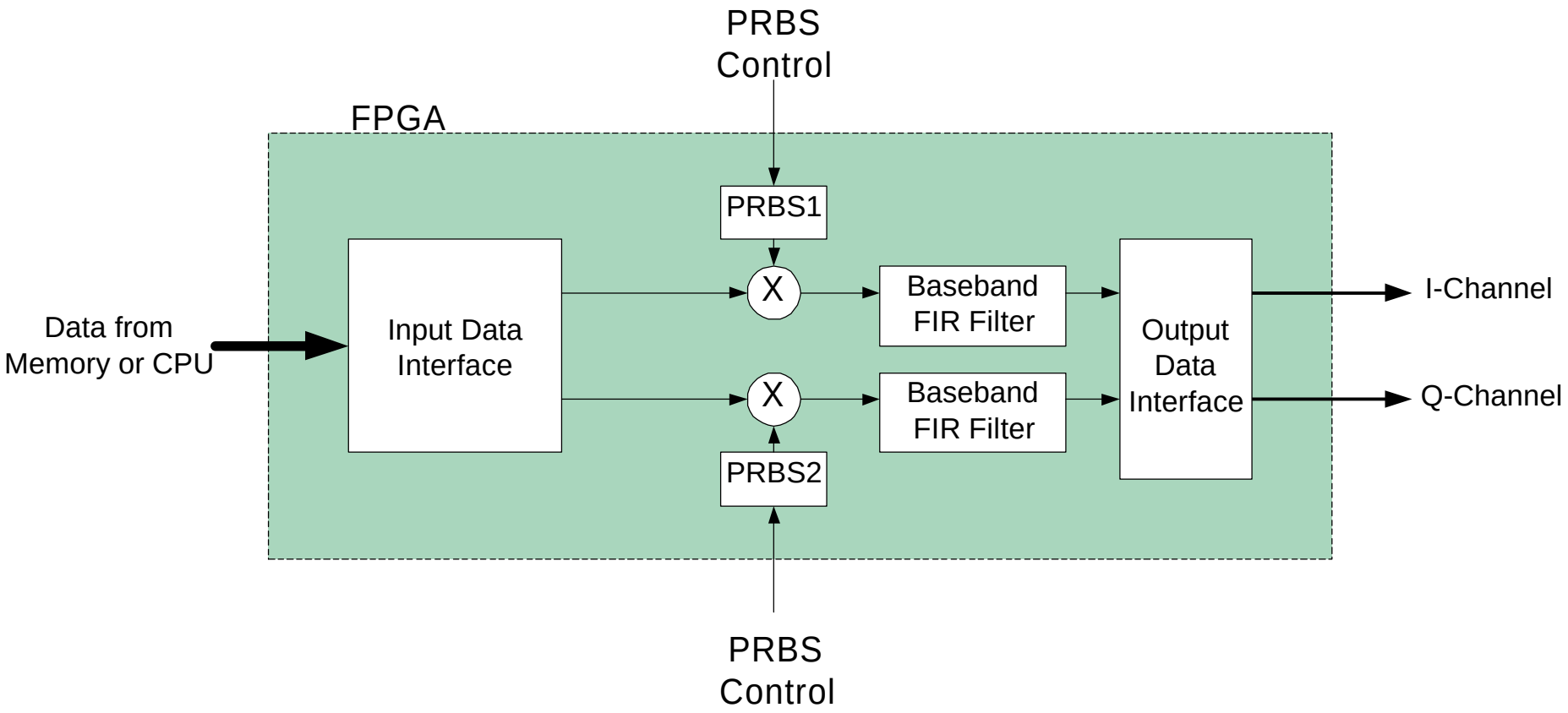
Spread Spectrum

- COTS hardware available for specific spreading codes
- Arbitrary spreading code transmitter not generally available
  - Want to look even more different than the other channel, yields more secure channel.

Spread Spectrum (SS): Transmitter Block Diagram

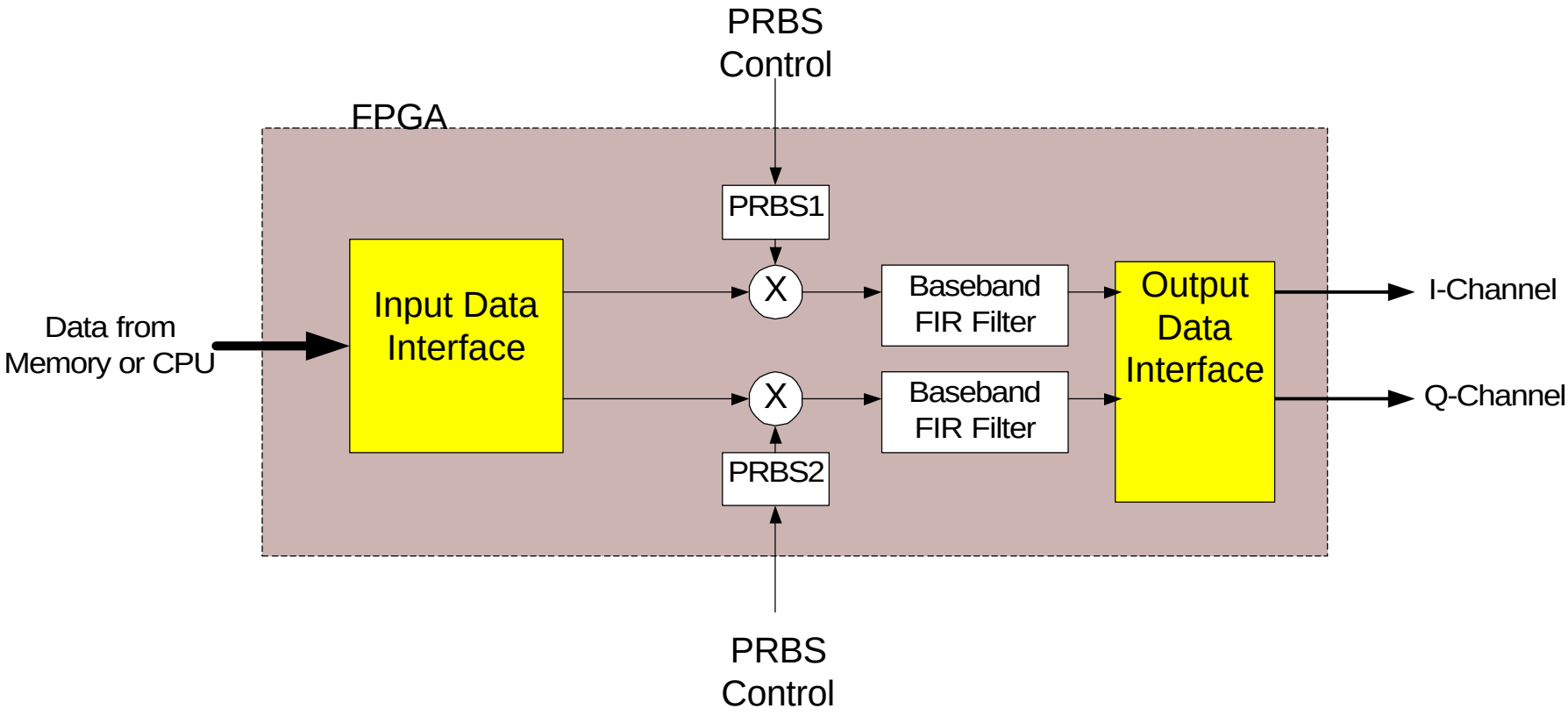


Modulator Block Diagram

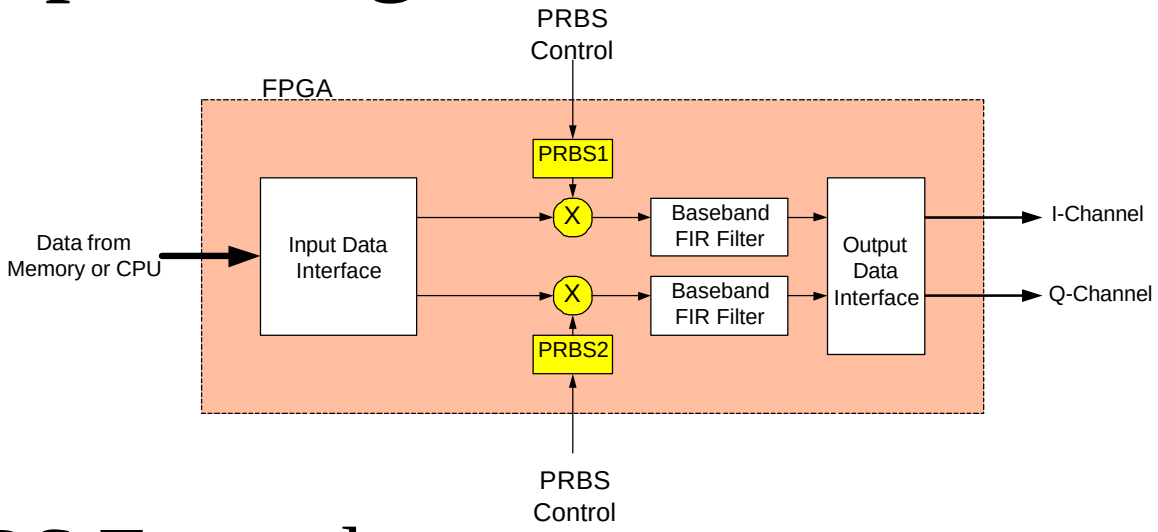


Data Input/Output

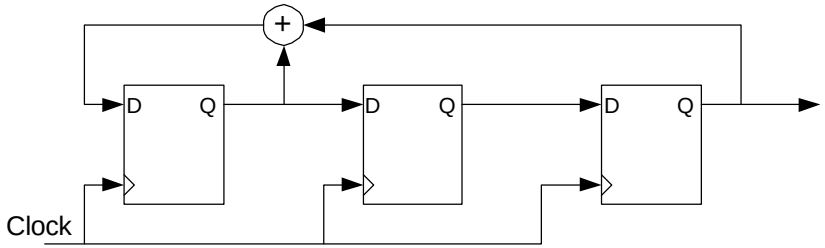
- Accept 8-bit parallel data stream
- Output two modulated data streams
  - 18-bit Parallel Output



Spreading Code Generators



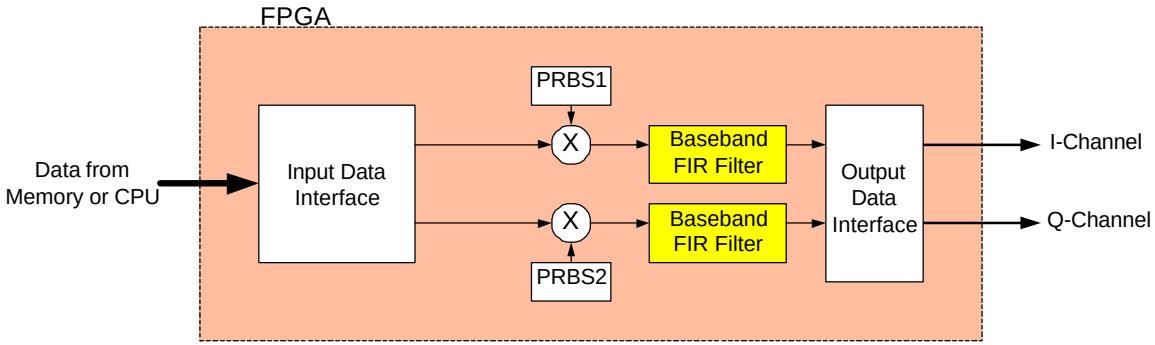
- PRBS Example: f(x) = x3 + x + 1 , length=3



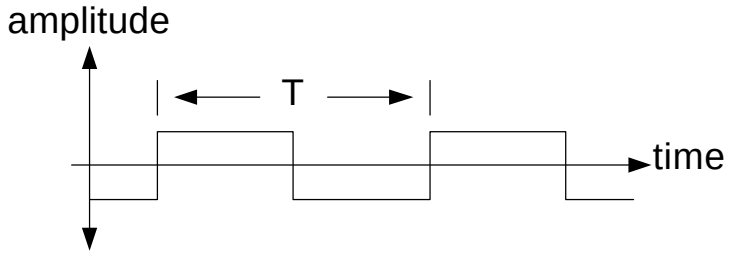
$111 \Rightarrow 110 \Rightarrow 101 \Rightarrow 010 \Rightarrow 100 \Rightarrow 001 \Rightarrow 011 \Rightarrow 111...$

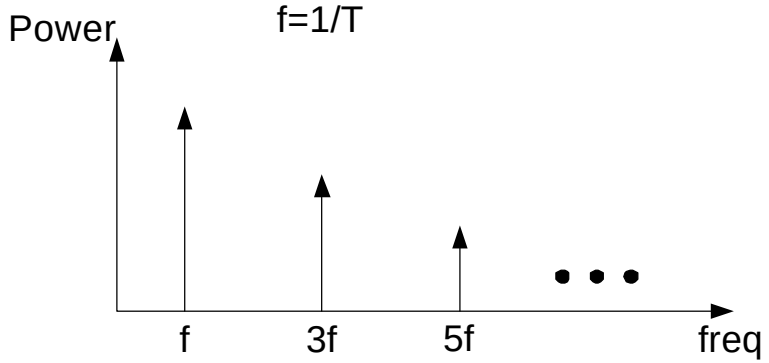
- IS-95 uses PRBS with length 32; $2^{32}-1 \Rightarrow$length of 4,294,967,295
- Different PRBS may be generated in Programmable Logic

Baseband FIR Filter



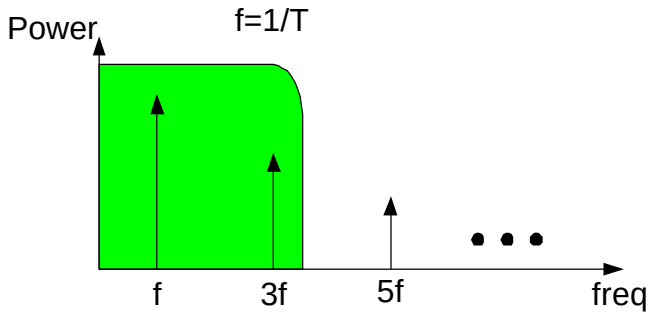
- Must use a digital filter to bandlimit the output signal.
  - Recall the Fourier transform of square pulse train:
    - Spectral content to infinity





Baseband Filter

- After filtering, the signal to be transmitted has finite bandwidth.
  - Filter is represented by green shaded region



  - All spectral content outside the filter bandwidth is highly attenuated, based on the filter characteristics.
  - Prevents our transmitted signal from becoming an unwanted interferor.

Motivation

- Why put into an FPGA?
- Why not a programmable DSP?

Why not a general purpose processor or DSP?

- As data rates increase, DSP's and CPU's can't keep up
- Example: IS-95 baseband filter
  - 1.288MHz input data, 5.152MHz output data
  - High Performance DSP can perform 1Multiply-Accumulate (MAC) per cycle.
  - 200MHz DSP, Filter is 48-taps in length

$$\frac{output}{48taps} \times \frac{tap}{1MAC} \times \frac{1MAC}{cycle} \times \frac{200cycle}{\text{second}} \equiv 4.167 \frac{outputs}{\text{second}}$$

4.167MHz output < 5.152MHz $\Rightarrow$ DSP can't keep up!

Why an FPGA?

- Can offload specialized operations, e.g. filtering, PRBS generation and modulation.
- Can change portions of the design quickly
  - Filter length and tap weights are easily adjusted
  - Alter PRBS on the fly
  - Add or remove functionality in system, and without a PCB change.

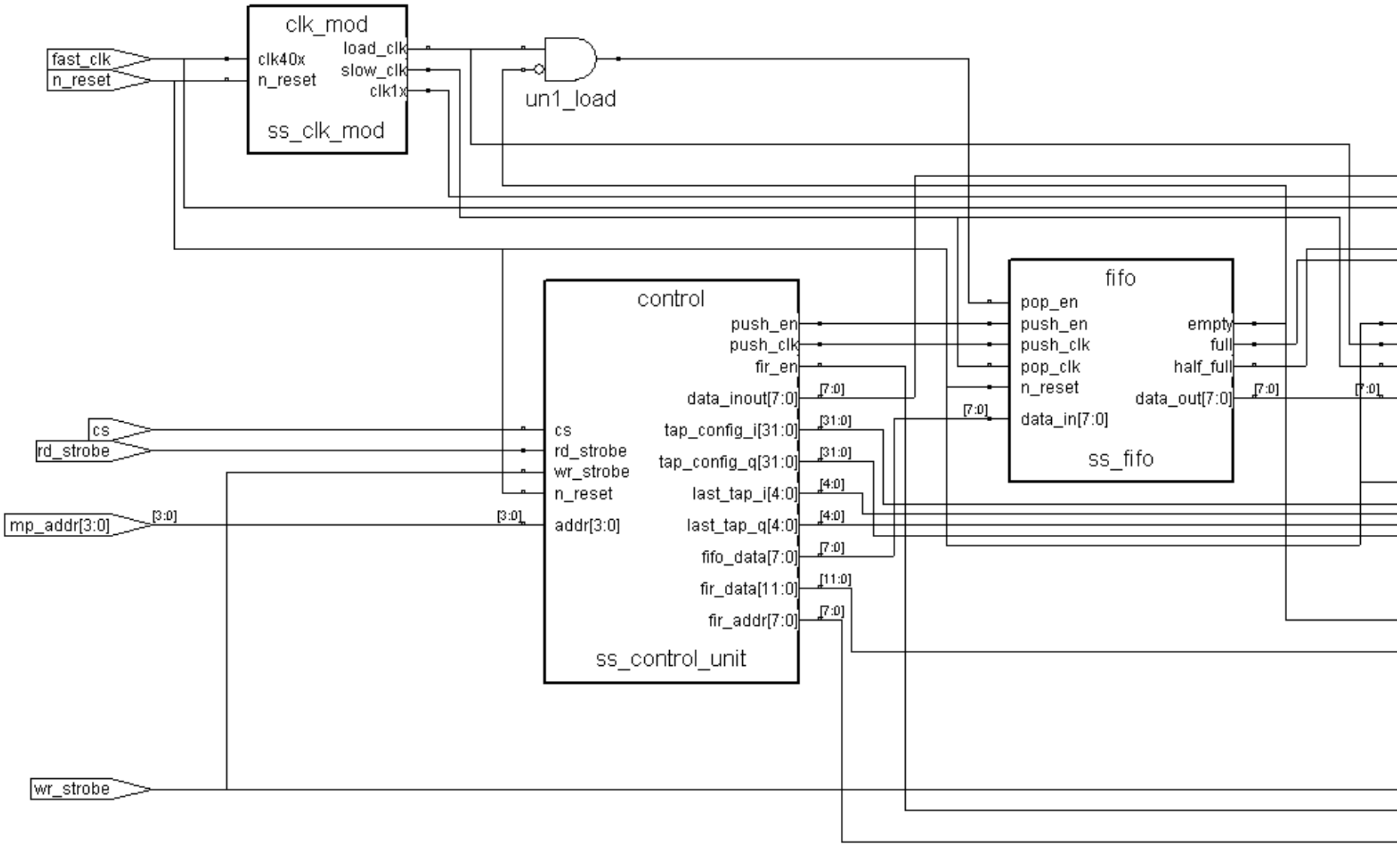
Design Goals

- Arbitrary spreading sequence
- At speed operation (filters must keep up!)
- Fit the entire design into one FPGA

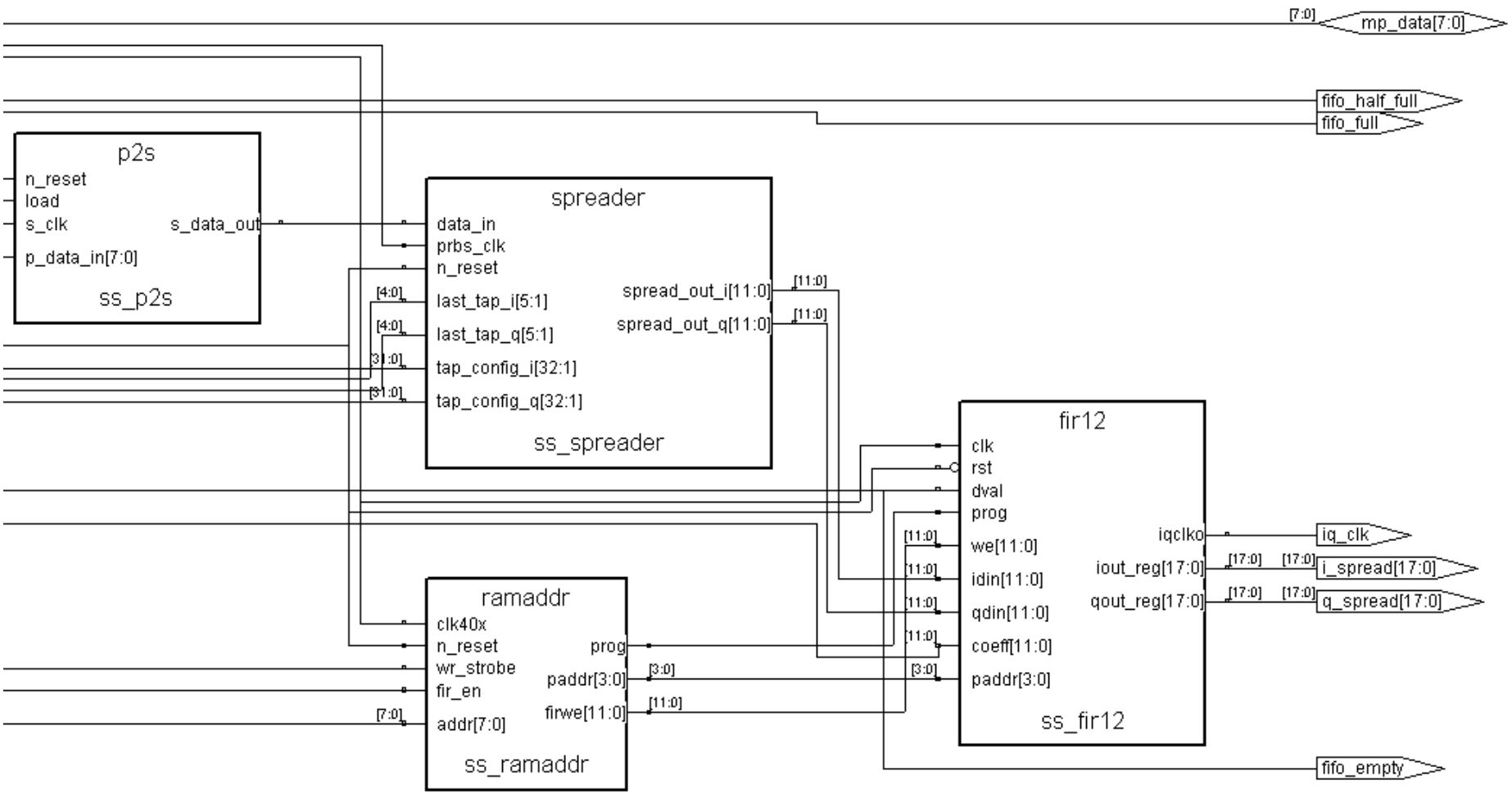
Spread Spectrum Transmitter

- Clock Modification block
- Microprocessor/DSP interface block
- FIFO data input block
- 8-bit parallel to serial conversion block
- Programmable I and Q channel Pseudo Random Binary Sequence Generator.
- Programmable Finite Impulse Response Filter

Spread Spectrum Transmitter Block Diagram



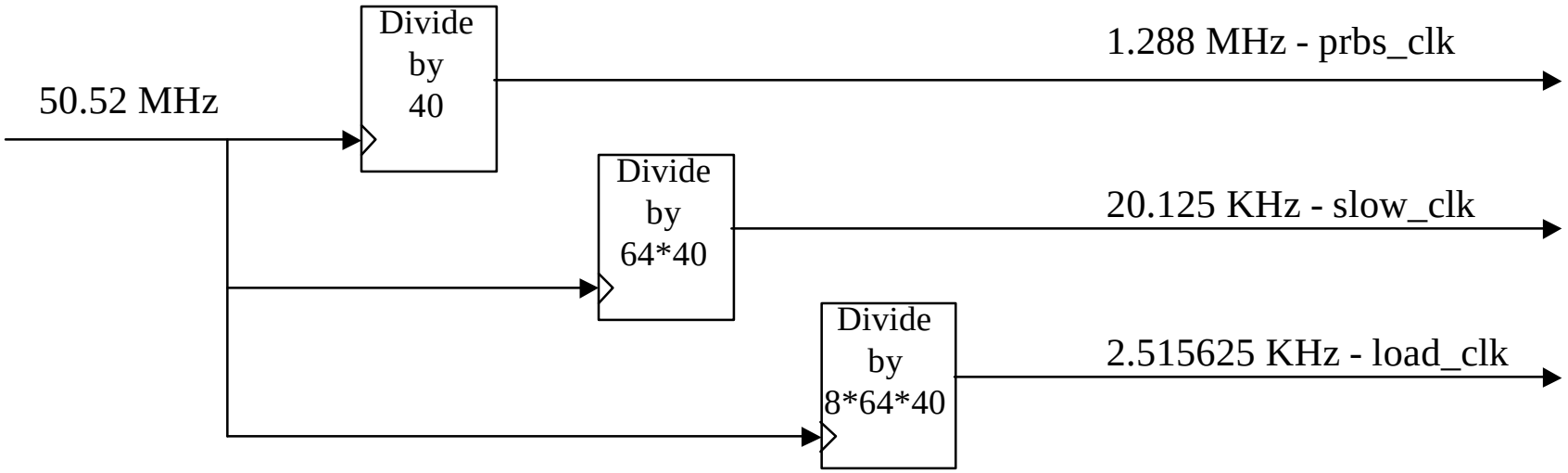
Spread Spectrum Transmitter Block Diagram (cont.)



Subsection Outline

- Clock Modification
- DSP/Microprocessor interface
- Data interface: FIFO & Serial conversion
- PRBS Design Plan
- Filter Design Plan

Clock Modification

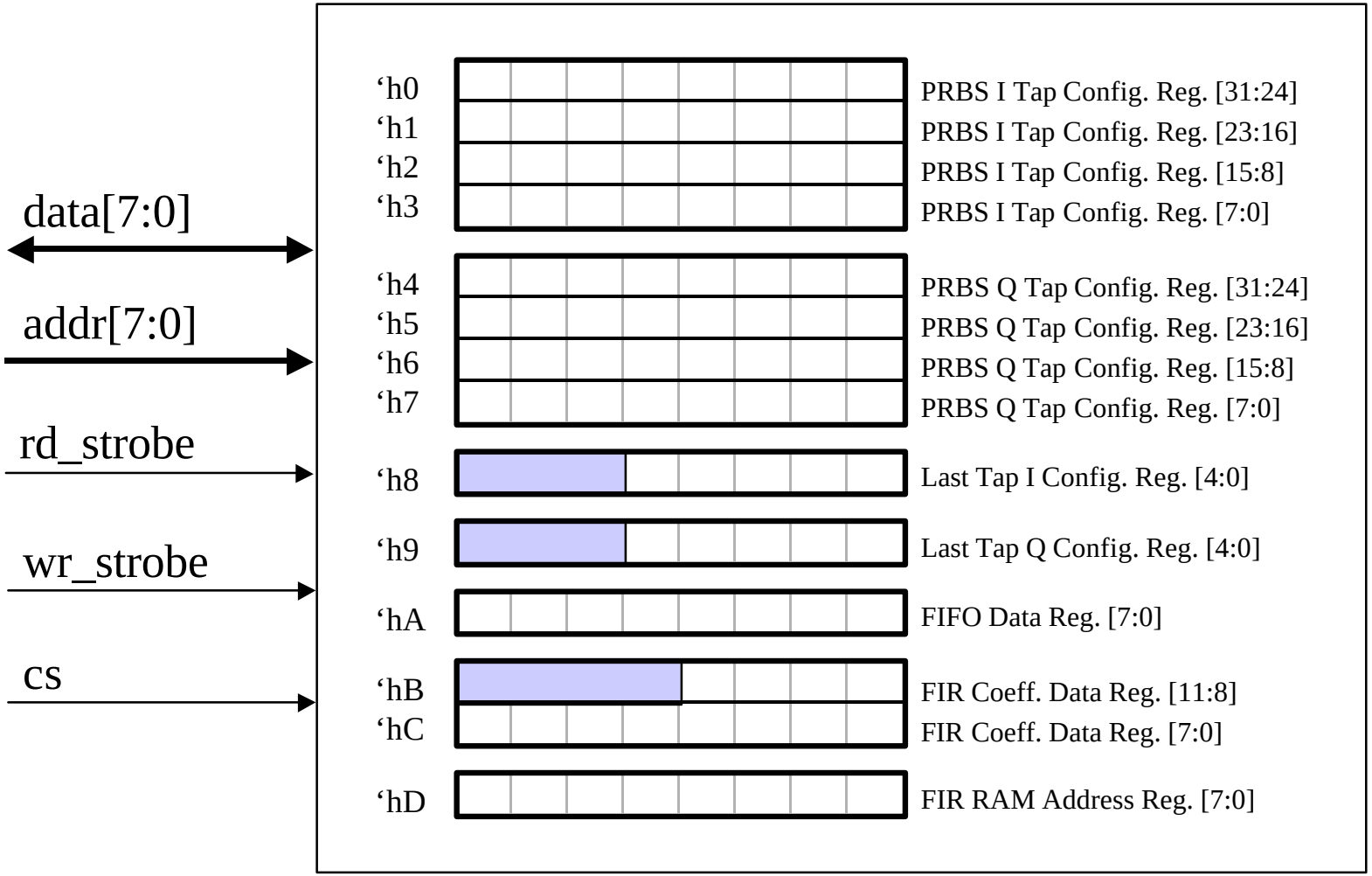


- Utilizes same input clock on all clock dividers to minimize clock skew.
- 51.52 MHz input (FIR Clock)
- 1.288 MHz output (PRBS clock)
- 20.125 KHz output (Serial Clock)
- 2.515625 Khz pulse (Load Strobe & FIFO Pop Clk)
- Associated Verilog Files: clk_mod.v

DSP/Microprocessor Interface

- 8 - bit Bi-Directional Data Bus
- 4 - bit Address Bus
- 14 uniquely addressable Data Registers
- Read/Write capability
- Chip Select
- Associated Verilog Files: control.v

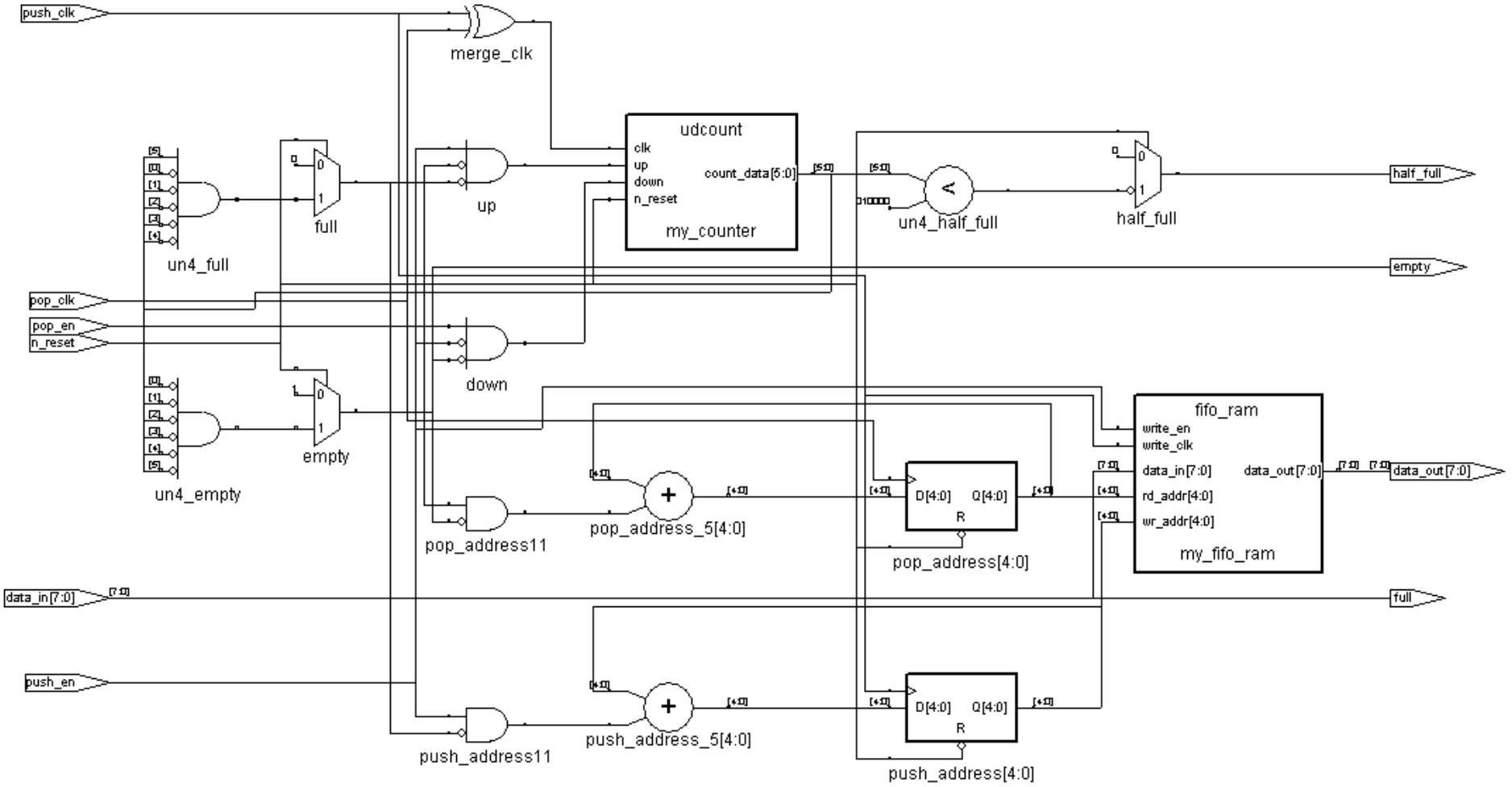
DSP/Microprocessor Interface Register Map



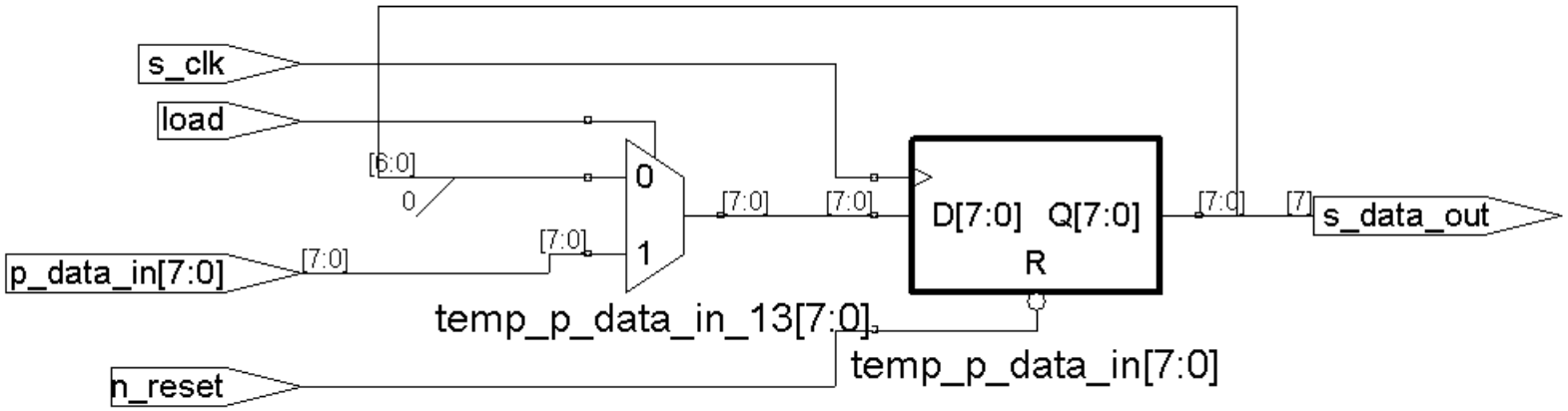
Data Interface : FIFO

- Async. Interface (separate push/pop clocks)
- 8 - bits wide x 32 deep
- Full, half_full, and empty status flags
- Push and Pop binary address counters
- Up/Down counter
  - Keeps track of number of datum in FIFO
- Dual Port RAM
- Associated Verilog files: fifo.v, fifo_ram.v, udcount.v

FIFO Block Diagram

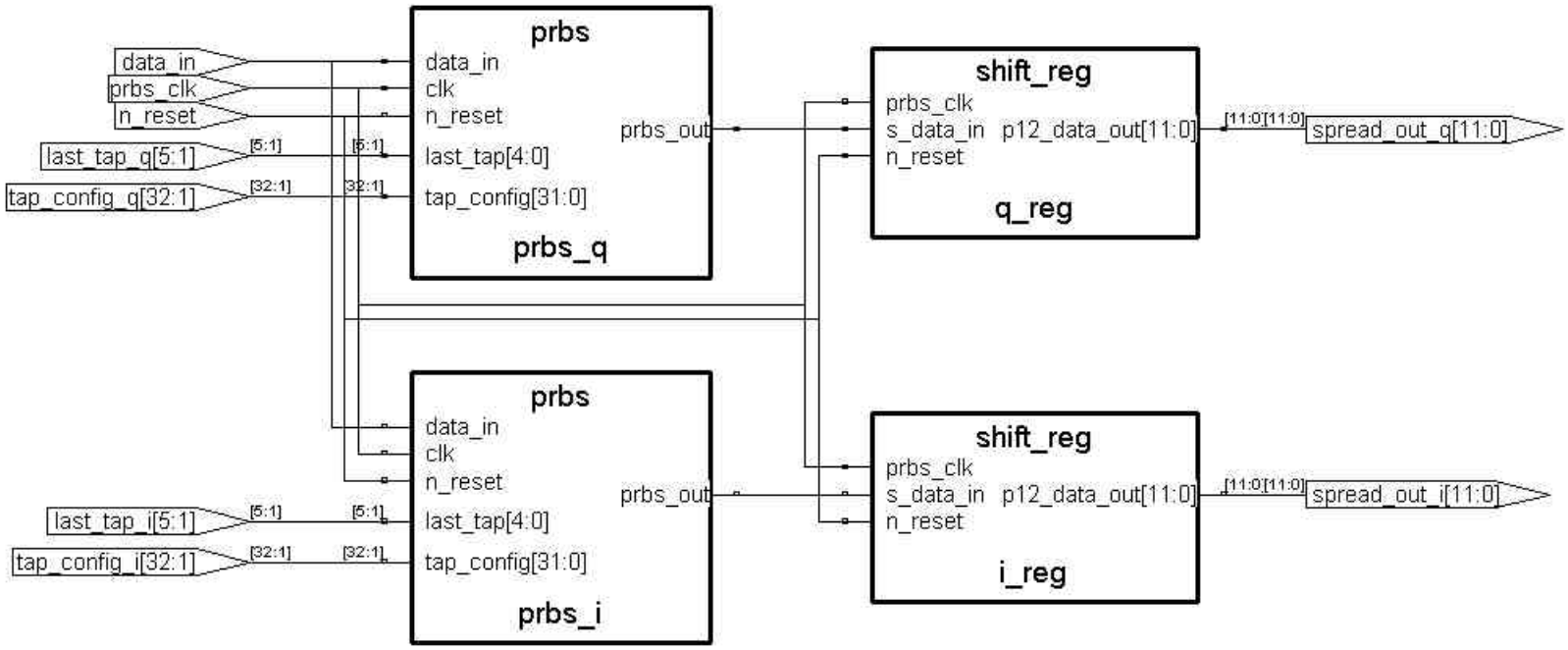


Parallel to Serial Converter



- 8-bit data in.
- Synchronous Load
- Asynchronous Reset
- Serial Clock
- Associated Verilog Files: p2s.v

I & Q channel “Spreader”

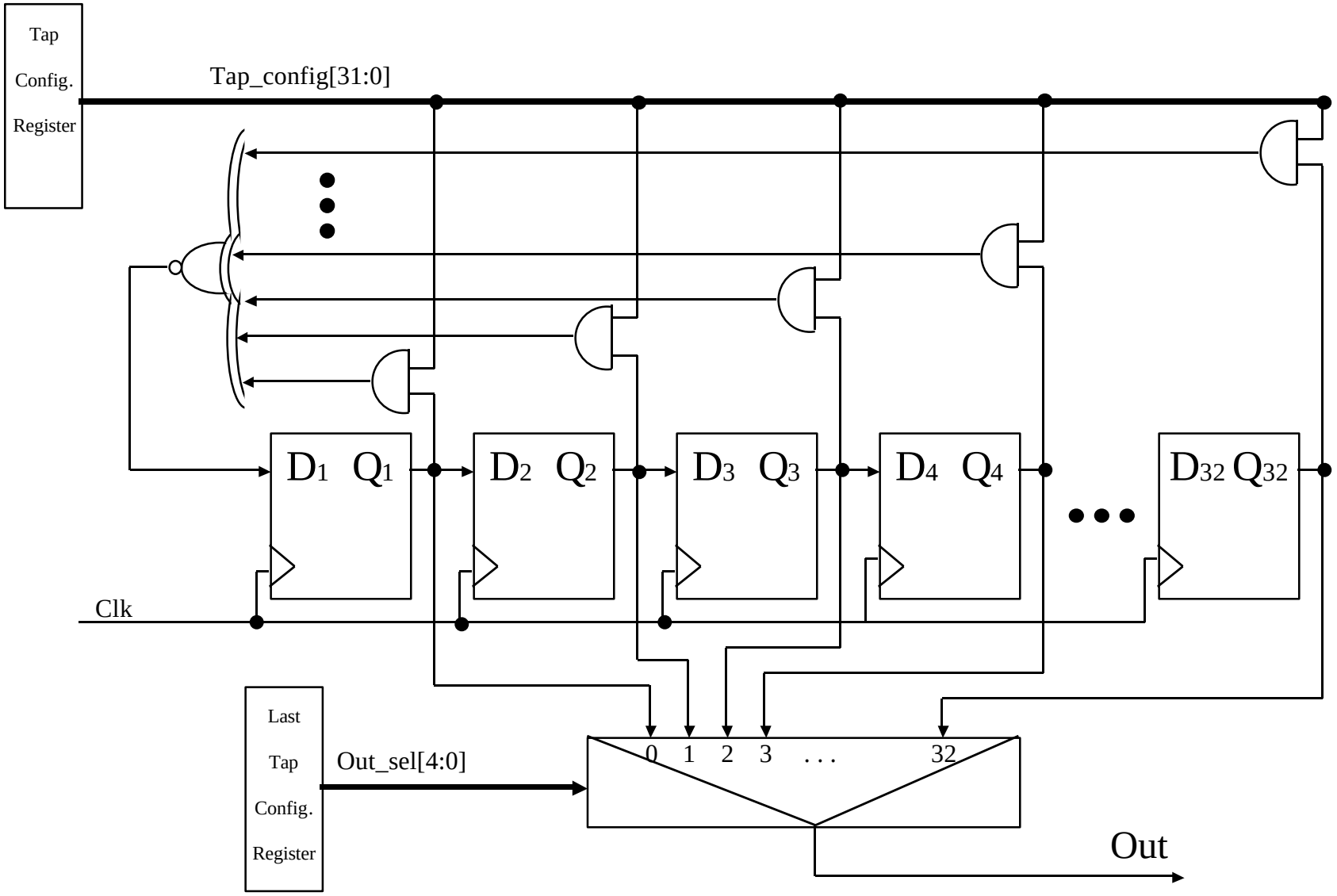


- Serial Data Input
- Independent I & Q channel PRBS.
- 12-bit serial to parallel converters
- Associated Verilog Files: prbs.v, dff.v, shift_reg.v

PRBS Design

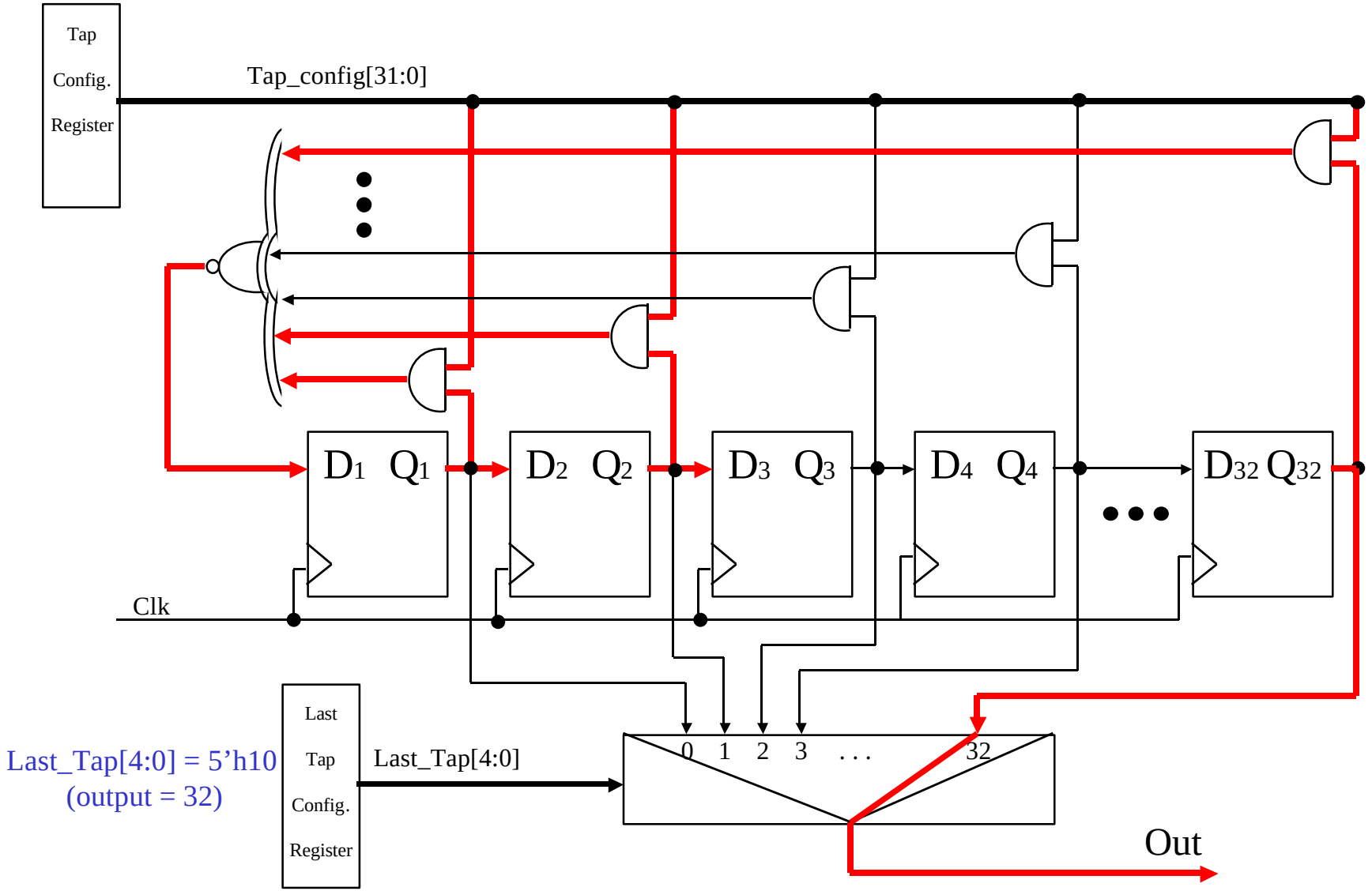
- Design such that the virtual register length may be varied by selecting any one of 32 register outputs as the PRBS data output
  - Max length = 32 registers deep
- Programmable Feedback paths
  - Design such that any register output may be included in feedback path
- Design options
  - PRBS configured in hardware by updating a register
    - Most flexible method; User can change PRBS on-the-fly
    - Inefficient hardware utilization

PRBS Design: Register Configured



PRBS Design: Example

Tap_config[31:0] = 32'h80000003 (only feedbacks 32, 2, and 1 active)



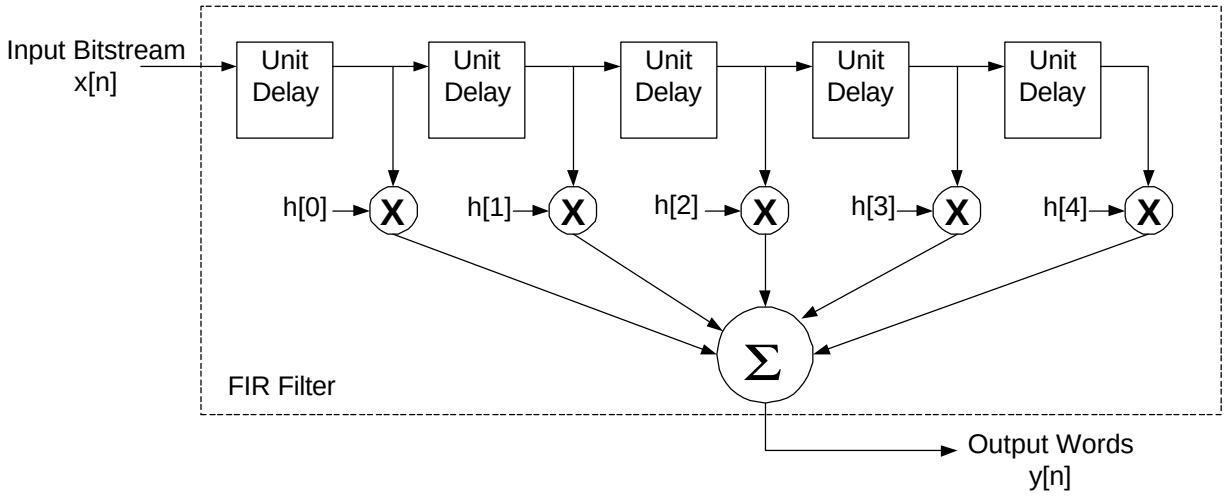
Last_Tap[4:0] = 5'h10
(output = 32)

PRBS (cont.)

- By creating a large X-NOR feedback path, logic delay is minimized. Allows for faster PRBS operation.
- Utilizes X-NOR feedback. On start-up, PRBS is in valid all-zeros state.
- Properly configured for a length of 32 registers, the resulting Pseudo Random Binary Sequence is:
  - (2n - 1) = (232 - 1) = 4,294,967,295 chips long.
- Operating at 1.288MHz, sequence repeats after:
  - 4,294,967,295 chips * 1/1288000 secs/chip * 1 min/60 secs = 55.6 minutes

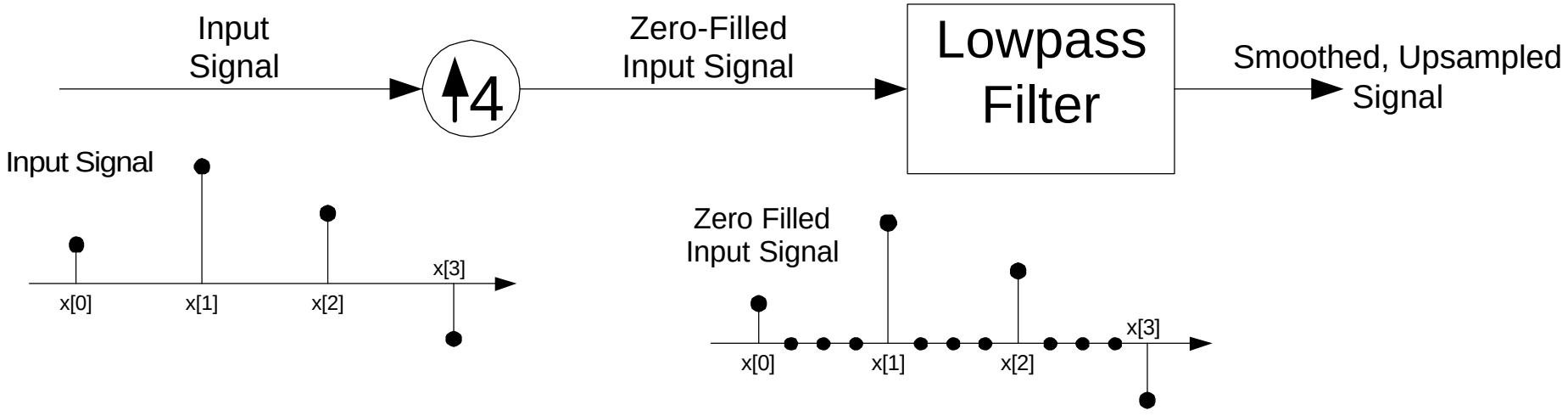
FIR Transverse Filter Structure

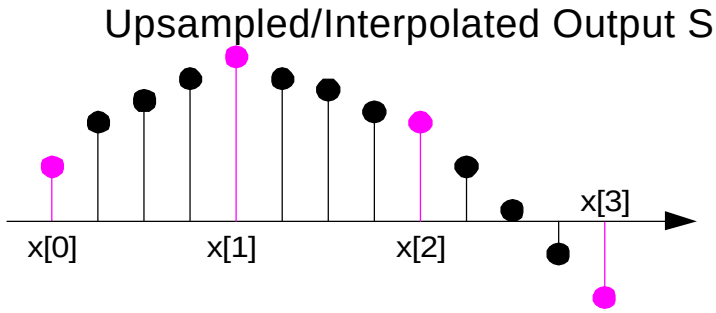
- Incoming data is convolved with desired filter response
- Output is a band-limited version of the original data



Interpolating FIR Filter

- An interpolator increases the sample rate to provide additional time resolution
  - Two operations: zero filling, then filter

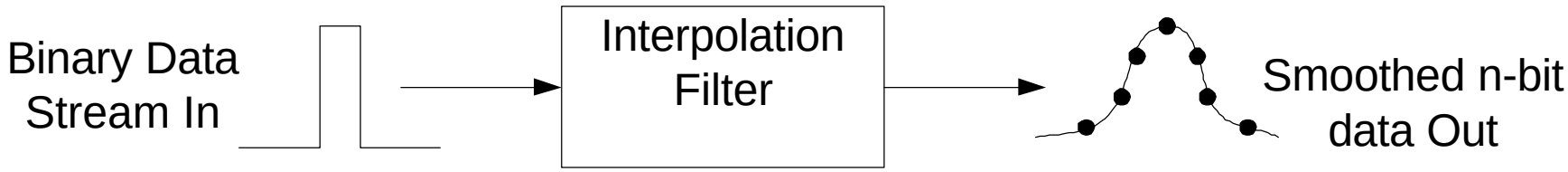




- Magenta ⇒ Original samples
- Black ⇒ Interpolated samples

Interpolation Filter

- Allows the system to “shape” the binary data into bandlimited signals



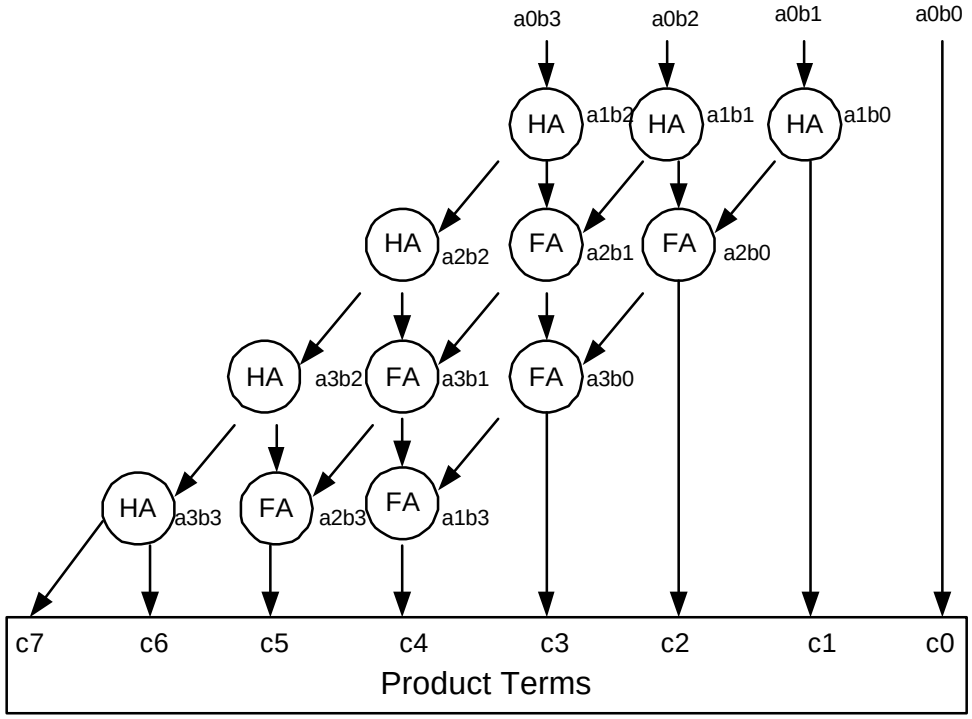
Filter Hardware Design

- Need many multipliers:
  - 1 multiplier for each filter tap
  - Multipliers can require substantial area
  - Example: IS-95 FIR Filter (48 tap), a direct implementation would require 48 multiplies and accumulation of all products, per output.
- Multiplier Implementation
  - Constant Coefficient Multiplier, well suited to FPGA implementation

Binary Multiply

2's Complement

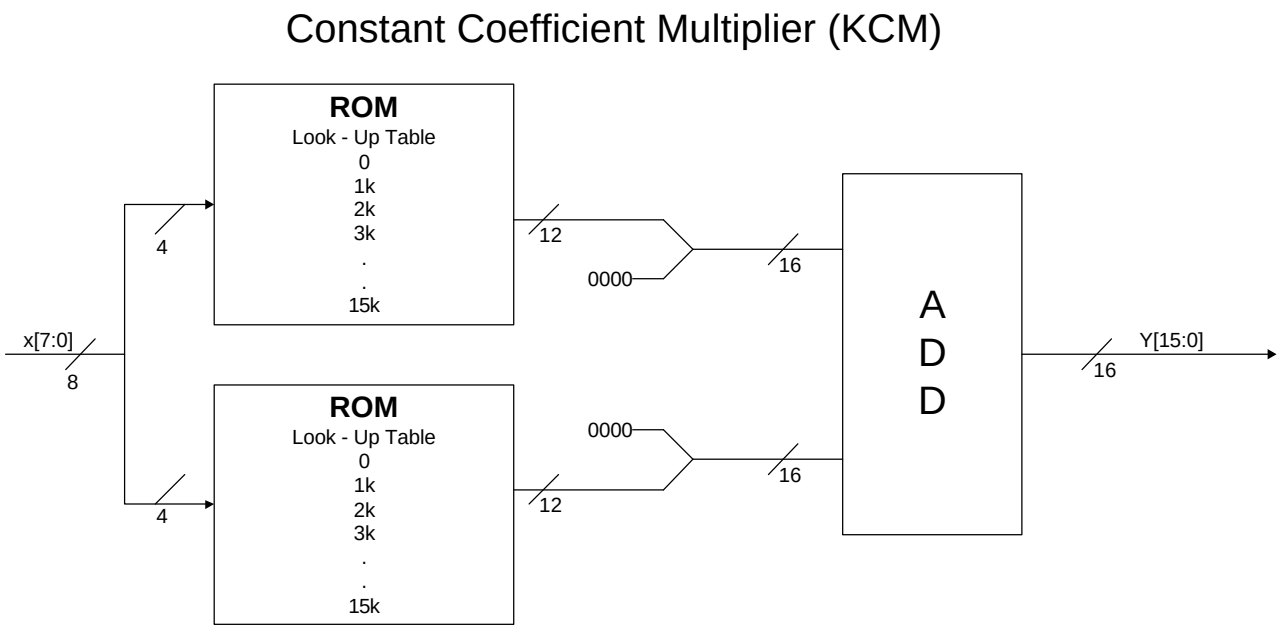
```
     1101
x    0111
     1101
    1101
   1101
  0000
------------
 01011011
```



FA= Full Adder
HA=Half Adder
$a_n b_n = a_n \& b_n$

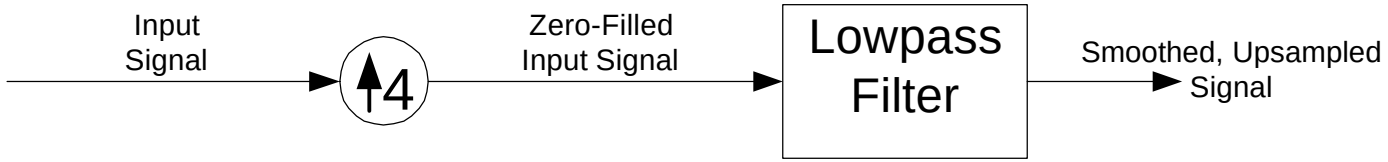
Constant Coefficient Multiplier

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

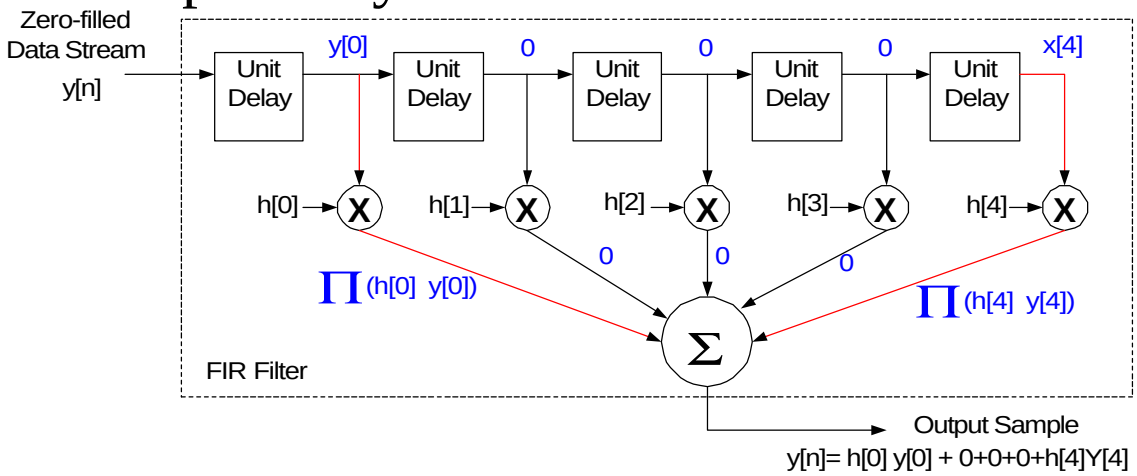


Interpolation Filter Optimization

- Recall from the Interpolation filter block diagram, that for every one input sample, four output samples are generated by zero-filling the input to the lowpass filter



- Because there is only one non-zero sample for every four input samples to the lowpass filter, only 1 multiply actually needs be done per four adjacent taps.
  - Can exploit this property and reduce the number of multipliers by a factor of four!



FIR: Output Sample Generation

- A given output sample is given by one of the following sums, due to the zero-filling property of the interpolating FIR filter

$$y_0[n] = x[n] \times h_0 + x[n-4] \times h_4 + \ldots + x[n-44] \times h_{44}$$

$$y_1[n] = x[n-1] \times h_1 + x[n-5] \times h_5 + \ldots + x[n-45] \times h_{45}$$

$$y_2[n] = x[n-2] \times h_2 + x[n-6] \times h_6 + \ldots + x[n-46] \times h_{46}$$

$$y_3[n] = x[n-3] \times h_3 + x[n-4] \times h_7 + \ldots + x[n-47] \times h_{47}$$

FIR: Output Sample Generation

- The output of the PRBS is a serial binary stream, where logic zero is mapped to -1. Also, when input sample data is not valid, the output of a filter tap (at time of invalid data) must be zero.
- Thus, $x[n] \in \{1,0,-1\}$
- For any filter tap, there are three possible values: $h[n] \bullet x[n] \in \{h_n, 0, -h_n\}$

FIR: Output Sample Generation

- From the columns of:

$$y_0[n] = x[n]\times h_0 + x[n-4]\times h_4 + \ldots + x[n-44]\times h_{44}$$
$$y_1[n] = x[n-1]\times h_1 + x[n-5]\times h_5 + \ldots + x[n-45]\times h_{45}$$
$$y_2[n] = x[n-2]\times h_2 + x[n-6]\times h_6 + \ldots + x[n-46]\times h_{46}$$
$$y_3[n] = x[n-3]\times h_3 + x[n-4]\times h_7 + \ldots + x[n-47]\times h_{47}$$

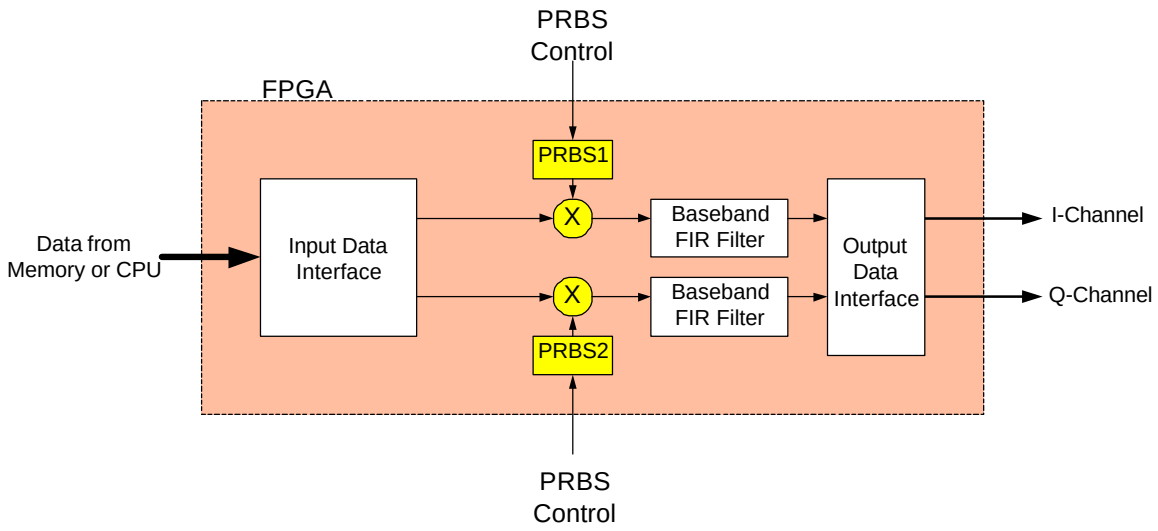
we note that only one of the four rows will have non-zero x[n] data. Thus each column contains: 4×3=12 - 3 = 9 unique values

(only one of the four "0" cases need be represented)

- Example: $y_{col[0]}=\{h_0,-h_0, h_1, -h_1, h_2, -h_2, h_3, -h_3,0\}$
- Each of the 12 columns (or taps) may be represented by a 9-word deep RAM.
  - RAM address[3:0] = {DataValid, Interpolation_Step[1:0],x[i]}

I&Q channel FIR Filters

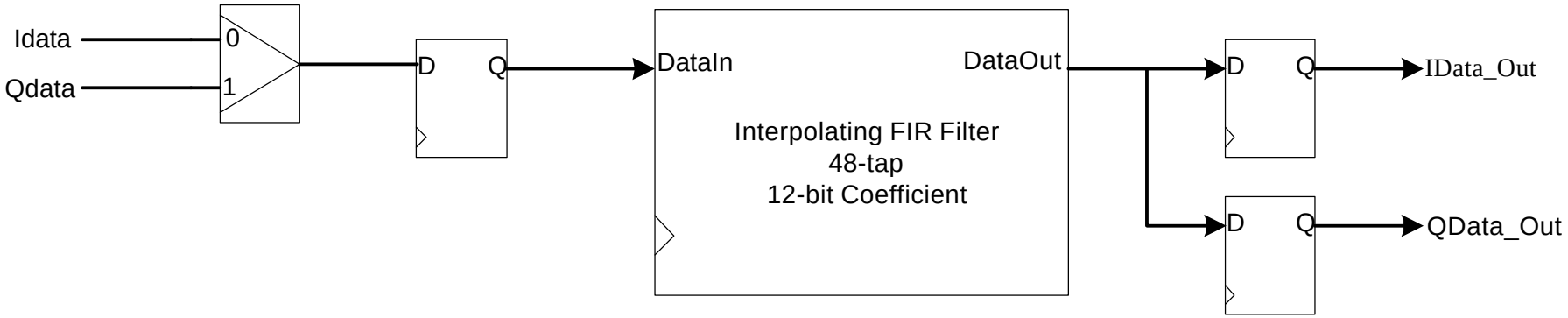
- The design requires the I and Q output channels to provide filtered output, as shown in the block diagram.



- The output data rate is *not* very high in current FPGA terms, only about 5MHz.

I&Q channel FIR Filters

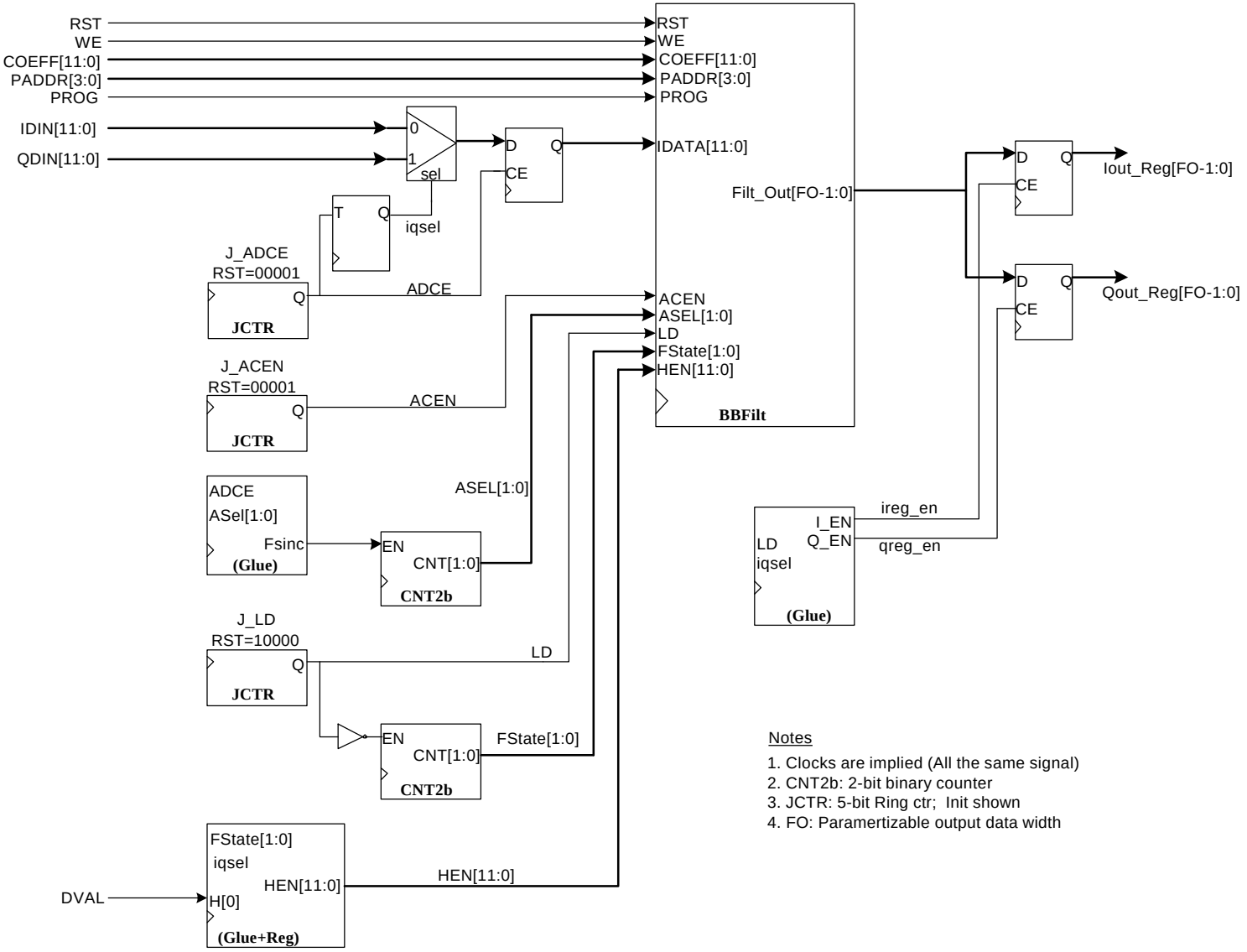
- Multiplex the two channels through a common filter at 2 times the output rate.



- More efficient use of resources
  - Less area at the cost of double the rate
  - Save power: half the routing, half the device utilage, double the frequency (CV^2f)
    - $(0.5 \times 0.5 \times C) \times V^2 \times (2 \times f) = 0.5 \times CV^2f$

50% less area $\Rightarrow$ Smaller, Cheaper FPGA
and $\Rightarrow$ 50% Less Power

FIR: Top Level Block



Notes

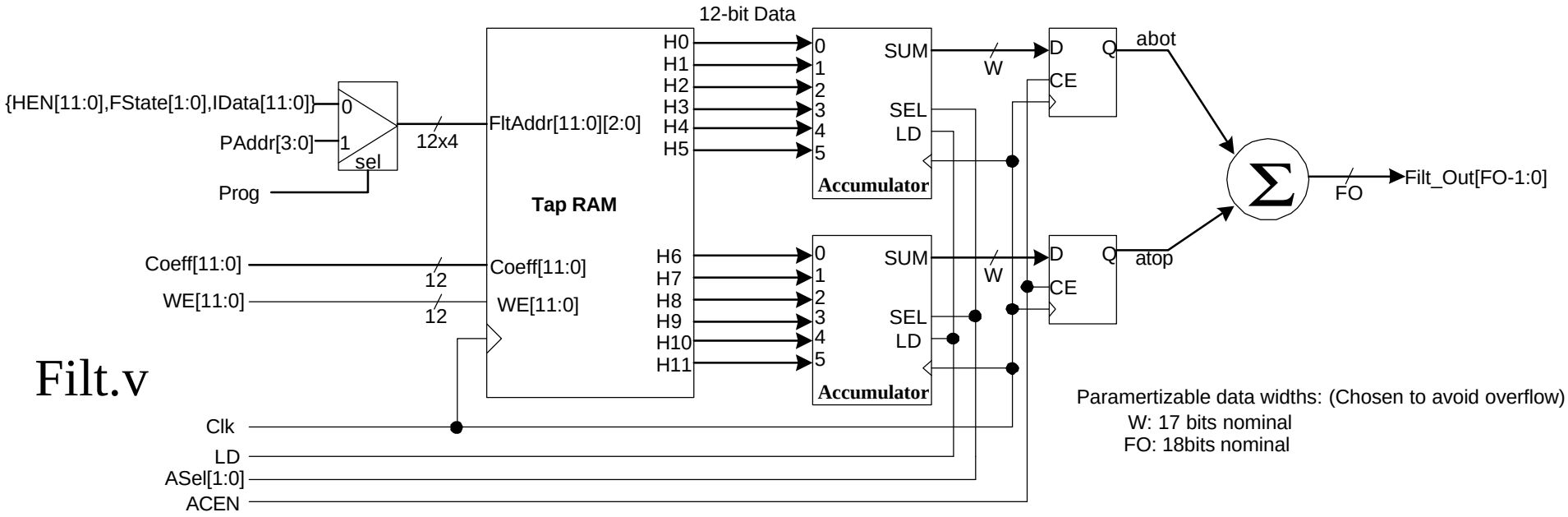
1. Clocks are implied (All the same signal)
2. CNT2b: 2-bit binary counter
3. JCTR: 5-bit Ring ctr; Init shown
4. FO: Paramertizable output data width

FIR: Top Level Block

Processes

- All FIR control functions
- I and Q channel input and output multiplexing
- Output clock generation
- Data output widths parameterizable at compile time
- Verilog filename: fir12.v

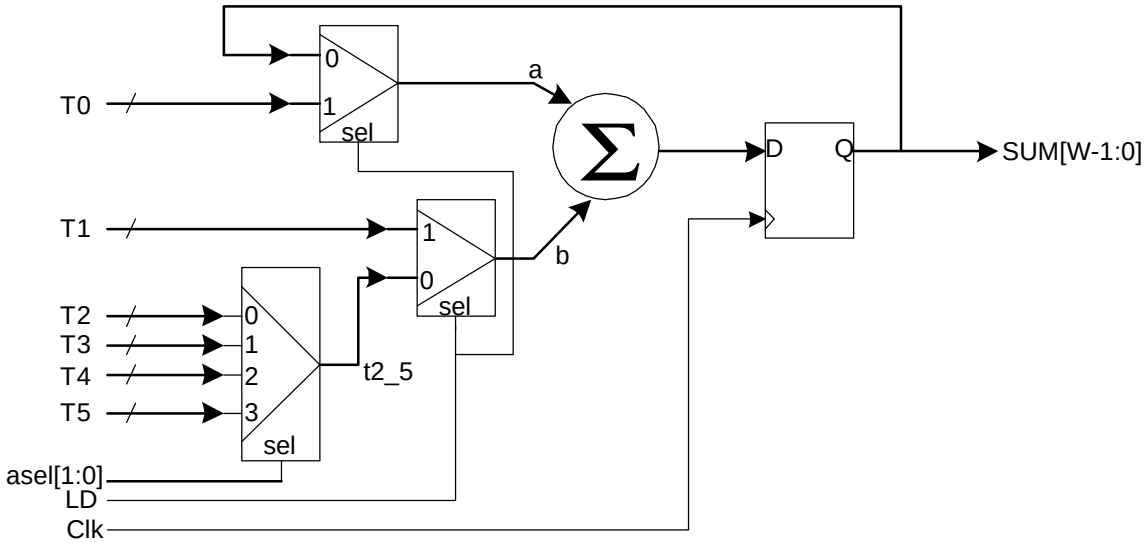
FIR: Filter Block



Filt.v

- Coefficient RAM is implemented as 12-bit wide by 16-deep
- Use RAM instead of ROM to allow in-system filter adjustment
- Top six and Bottom six 12-bit tap outputs are fed into independent accumulators operating at 5X the output rate, producing one new sample at the required data rate.
- One pipeline stage prior to summing the accumulator outputs
- Accumulator width: 18-bits nominal, avoids internal overflow

FIR: Accumulator



- When LD=1, SUM = T1 + T2
- When LD=0, SUM = SUM + T2_5 (4:1 mux output)
- Multiplexer controls handled outside this block for modular instantiation.
- Accumulator width: 17-bits, avoids internal overflow.
- New data is expected every 6th clock to guarantee overflow avoidance.
- Verilog file: accum.v

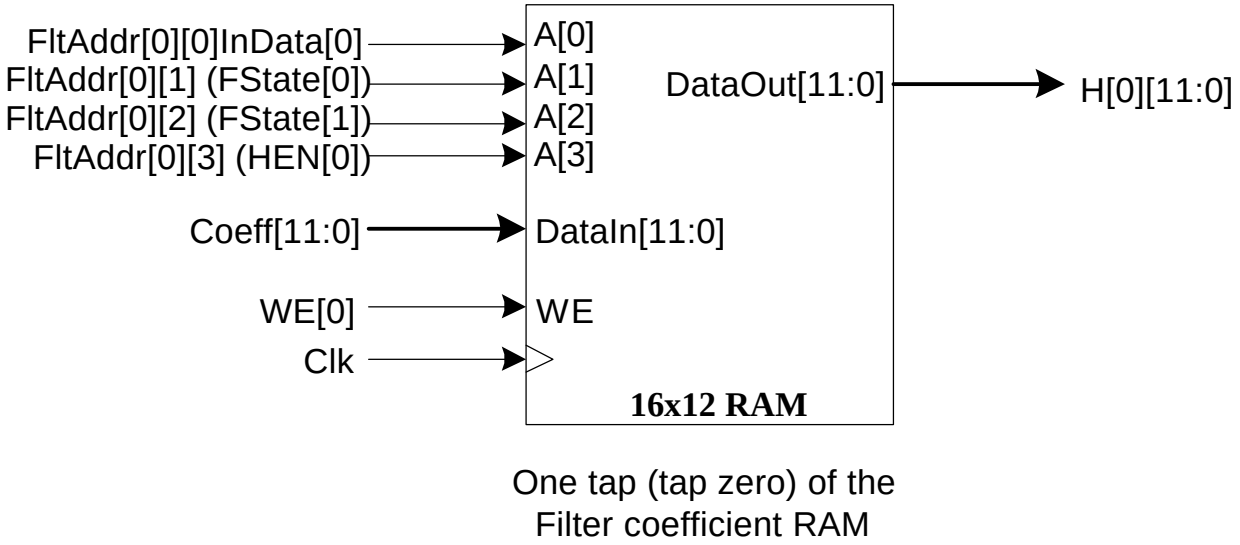
FIR: Synthesizing Adders

- Synplicity did a much better job synthesizing the design to the target architecture with the behavioral code:

```
assign sum = a + b;
```

  - Maps the logic to dedicated carry chains and arithmetic circuitry
- Designing a modular ripple carry adder and synthesizing produced far inferior results!
  - Used nearly twice the area, and did not utilize dedicated arithmetic logic.

FIR: Tap RAM



One tap (tap zero) of the
Filter coefficient RAM

- (One of 12 RAM Taps shown)
- Synchronous 12-bit by 16-deep RAM.
- RAM content is loaded through a dedicated programming mode from the microprocessor interface
- Verilog file: ram_fir.v

What we accomplished:

- Implemented entire design in Verilog HDL
- Block-Level Simulation performed using Cadence Verilog-XL 2.8
- Synthesized Verilog HDL using Synplicity Synplify 5.2.2a
- Targeted Virtex 50 and processed design through Xilinx Alliance 2.1i.

Place and Route Results

Design Information

Target Device : xv50

Target Package : bg256

Target Speed : -4

Device utilization summary:

Number of External GCLKIOBs	3 out of 4	75%
Number of External IOBs	54 out of 180	30%
Number of CLBs	190 out of 384	49%
Number of GCLKs	4 out of 4	100%
Number of TBUFs	16 out of 832	1%

Total equivalent gate count for design: 17557

Design statistics:

Minimum period: 15.469ns (Maximum frequency: 64.645MHz)

Conclusion

- *Designed and Built a Spread Spectrum Modulator in an FPGA*
- Programmable PRBS Structure
- Programmable FIR Interpolation Filter
- Synthesized, Place and Routed for Xilinx FPGA, XV50-4
- Design uses 190 CLB's, or 49% of capacity of Virtex 50.
- Maximum Accumulator Frequency = 64.645MHz.
- Max Data output frequency = 6.4645MHz.
  - (Required only 5.152MHz)

What now?.

- *Re-Design for smaller Xilinx chip. Virtex device is expensive and twice as large as the logic required. Logic will fit inside of cheaper Spartan XCX30XL, however without re-designing, the maximum clock speed achieved was only 40 MHz (11.52 MHz too slow)*
- *Increase PRBS size. N = 32 is too small, and the additional logic required is minimal*
- *Allow different PRBS seeds.*
- *Enhance Microprocessor interface.*
- *Alter FIFO design to better utilize Spartan architecture.*
- *Increase FIR filter resolution.*

References

[1] Viterbi, A.J. *CDMA: Principles of Spread Spectrum Communications*, Addison-Wesley, 1995.

[2] Hwang, K., *Computer Arithmetic*, Wiley, 1979.

[3] Rappaport, T.S., *Wireless Communications: Principles and Practice*, Prentice-Hall, 1996.

[4] Lee, J.L., Miller, L.E., *CDMA Systems Engineering Handbook*, Artech House, 1998.

[5] Xilinx, *1999 Programmable Logic Data Book*

[6] Palnitkar, S., *Verilog HDL, A Guide to Digital Design and Synthesis*, Prentice-Hall, 1996.

[7] Lim, Y.C., Parker, S., "FIR Filter Design Over a Discrete Powers-of-Two Coefficient Space," *IEEE Trans. On ASSP*, June 1983, pp. 583-591.

[8] Goslin, G., Newgard, B., "16-tap, 8-Bit FIR Filter Application Guide," Xilinx application note, November 21,1994.

References (cont.)

[9] Alfke, Peter “Efficient Shift Registers, LFSR counters, and Long Pseudo-Random Sequence Generators,” Xilinx application note, July 7, 1996.

[10] Chapman, Ken, “Constant Coefficient Multipliers for XC4000E,” Xilinx application note, December 11, 1996.

[11] Camilleri, Nick, “170 MHz FIFOs Using the Virtex Block SelectRAM+,” Xilinx application note, December 10, 1998.