802.11a PHY
Digital Back-end Design

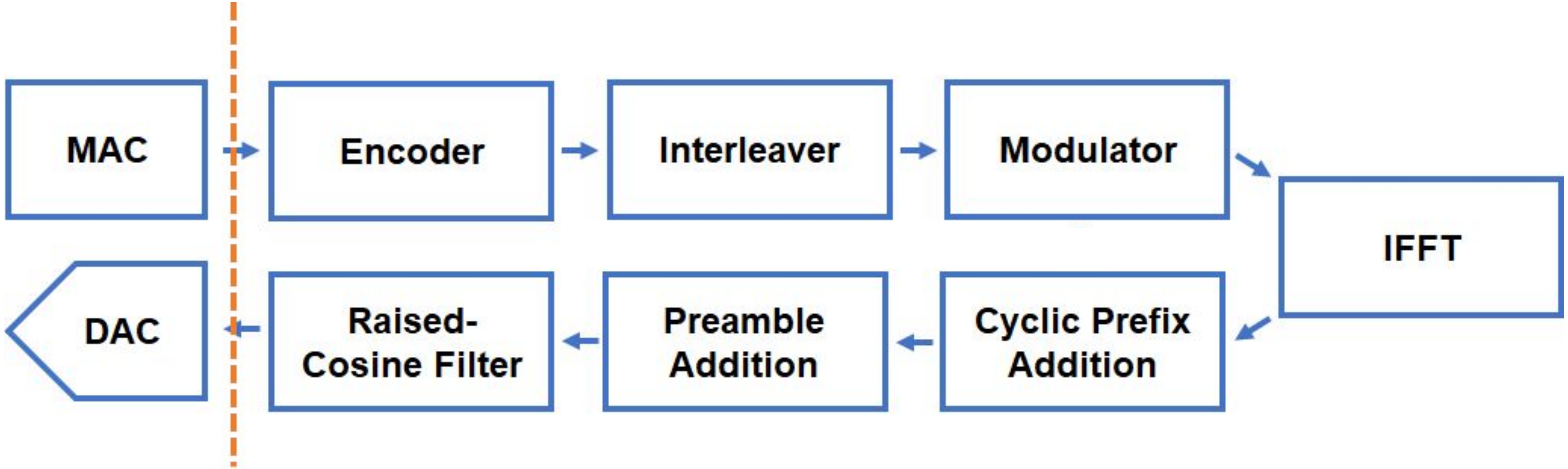
Sean Huang, Kunmo Kim, Paul Kwon, Josh Sanz, Meng Wei

Why do OFDM in Chisel?

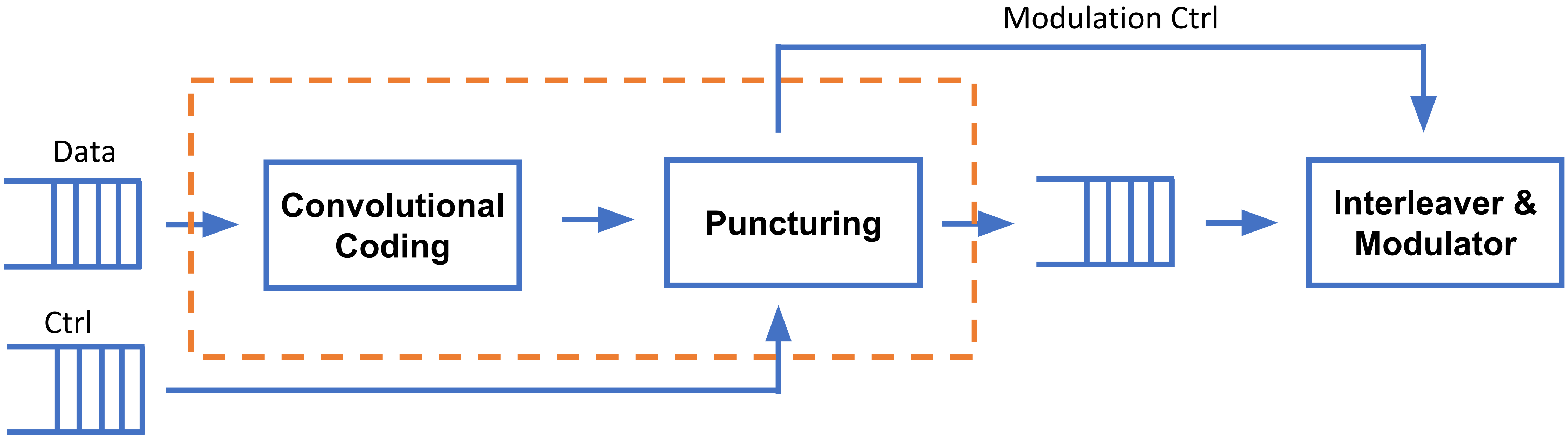
- Several Verilog/VHDL implementations of WiFi exist
- But they’re not flexible
  - Written to support one standard in particular [1]
  - Or wrap several standard specific basebands[2]
- And they’re big
  - 880,000 lines of code[1]
  - 172,000 lines of code[2]

[1] J. Shi, “OpenOFDM.” https://github.com/jhshi/openofdm, 2017.
[2] phthinh, “IEEE 802.11 OFDM-based transceiver system.” https://github.com/phthinh/OFDM_802_11

802.11a Transmit Chain

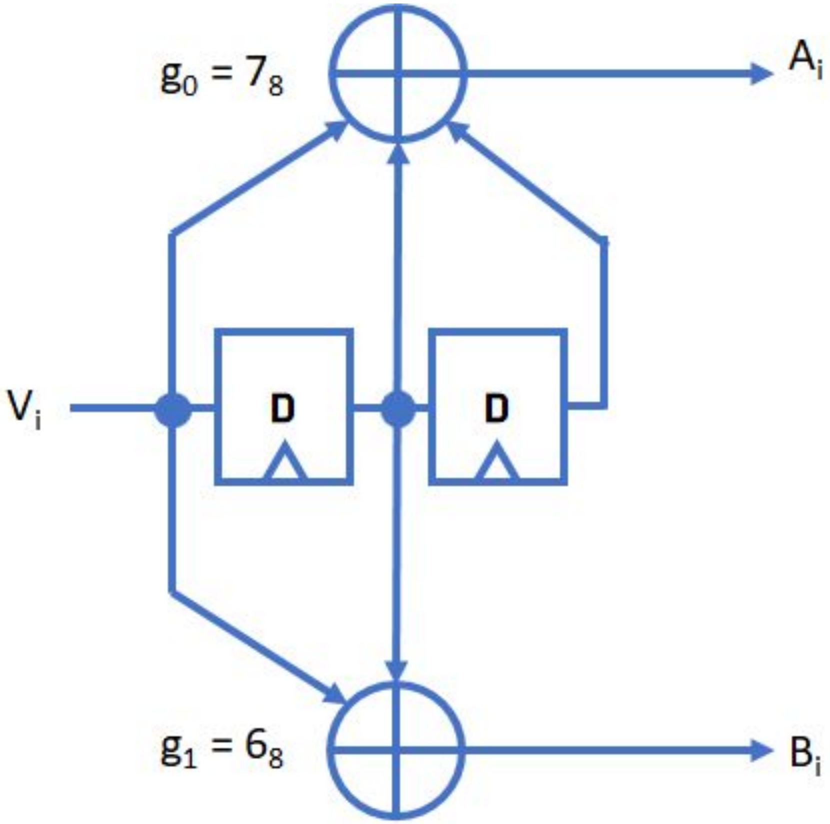


802.11a Tx Encoder



- Fully compatible with 802.11a Specification
    - Puncturing matrix can be modified on the fly
    - Support programmable convolutional coding matrix

802.11a Tx Encoder - Convolutional Code (7, 6)

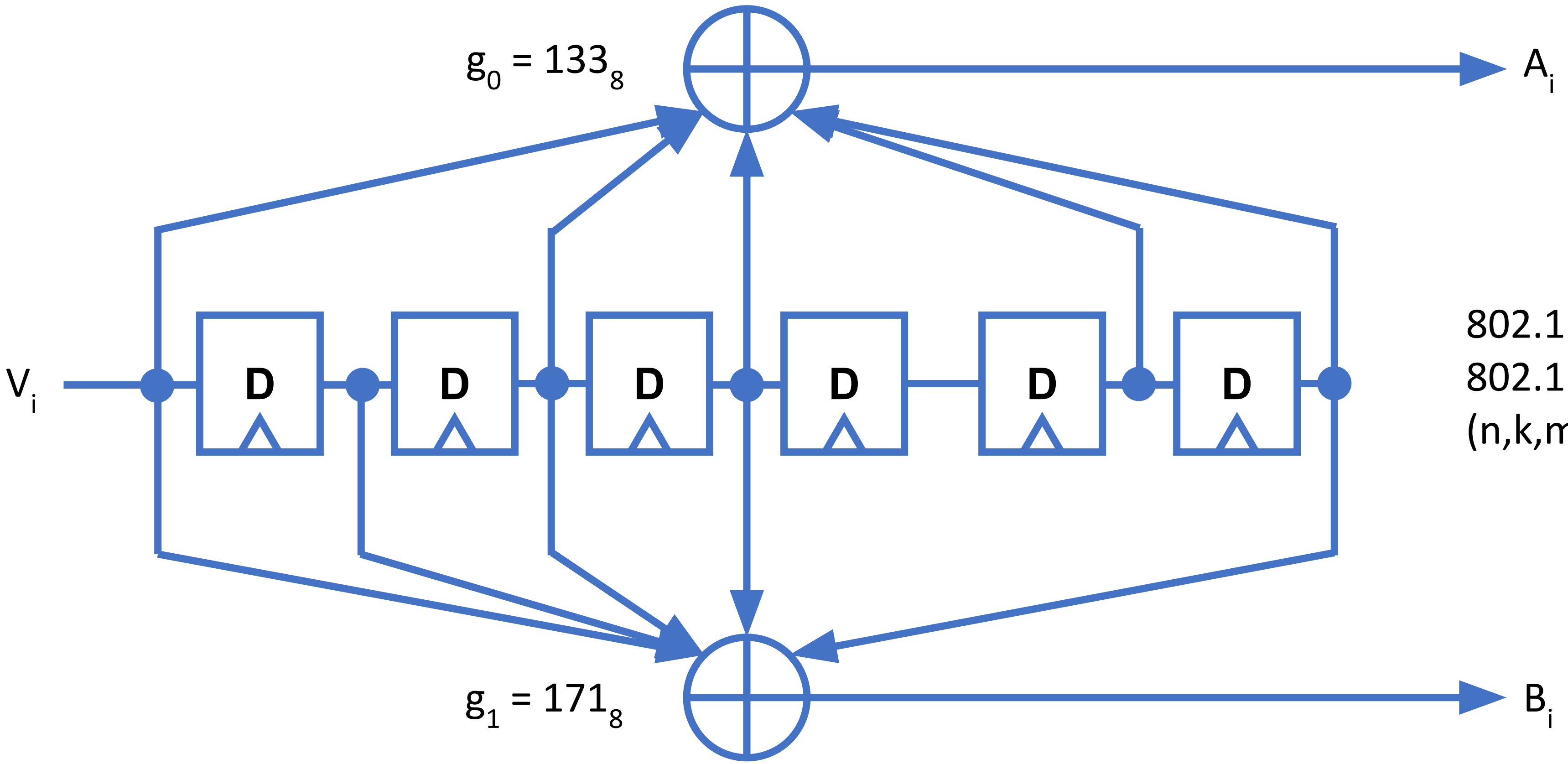


Current State	Input	Output (g_0g_1)	Next State
00	0	00	00
	1	11	10
01	0	10	00
	1	01	10
10	0	11	01
	1	00	11
11	0	01	01
	1	10	11

Some useful equation:

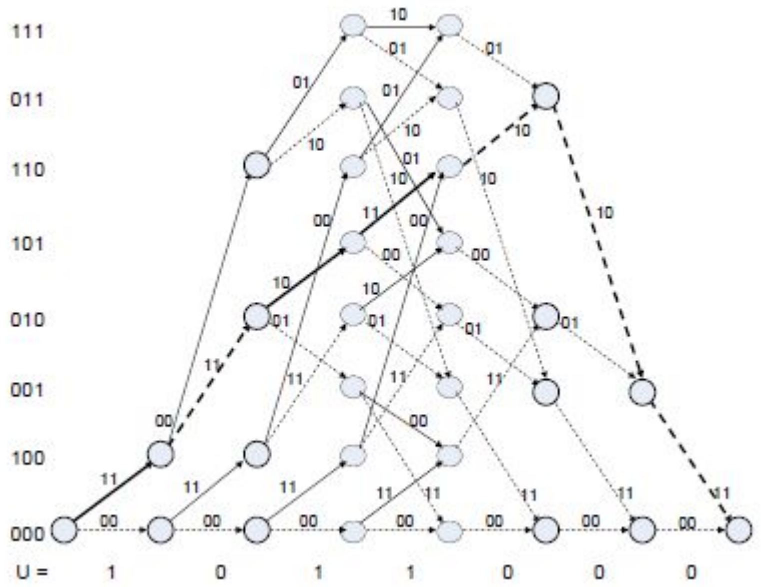
- # of memory element = m
- Number of states = 2^m
- Constraint length $K = m + 1$
- Coding rate $R = \frac{k}{n}$

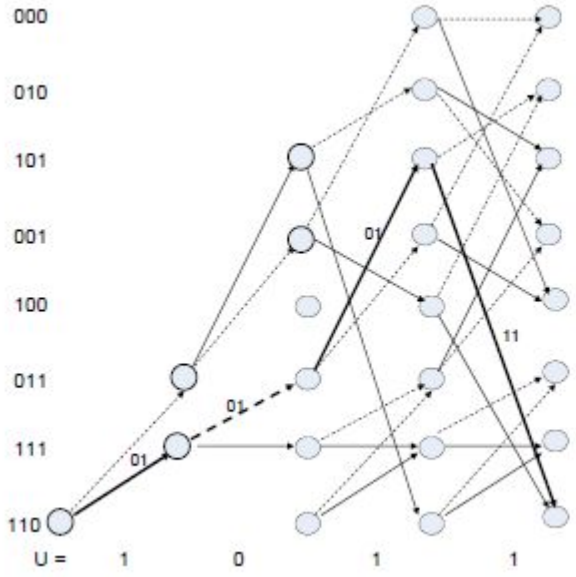
802.11a Tx – Convolutional Encoder (802.11a Standard)



802.11a default coding rate = ½
802.11a default constraint length (K) = 7
(n,k,m) = (2,1,6)

Tail-Biting





- Zero-flushing causes **rate loss** for a short input sequence
  - m-zeros will be padded after ‘L’ input sequence
  - $R_{eff} = R_{ideal} \cdot \frac{L}{L+m}$
- Tail-biting: simply initialize ‘m’ flops with the last m bits of input sequence
- Tail-biting allows code-rate to remain the same

802.11a Tx – Puncturing

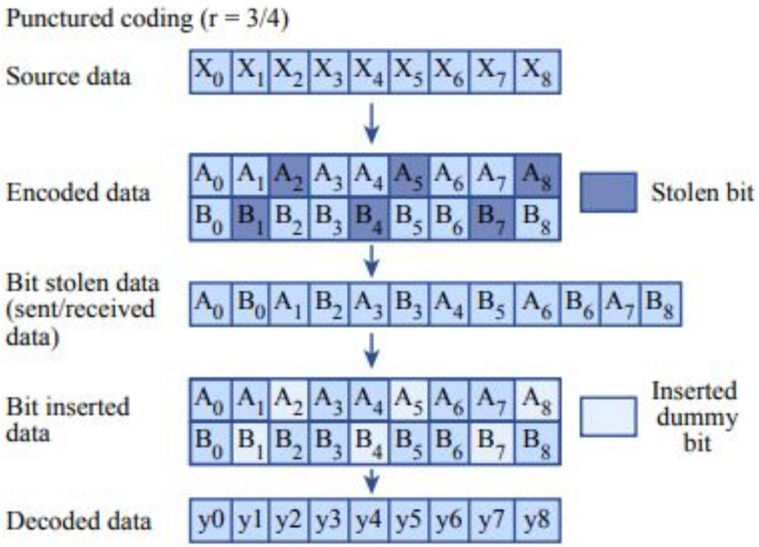
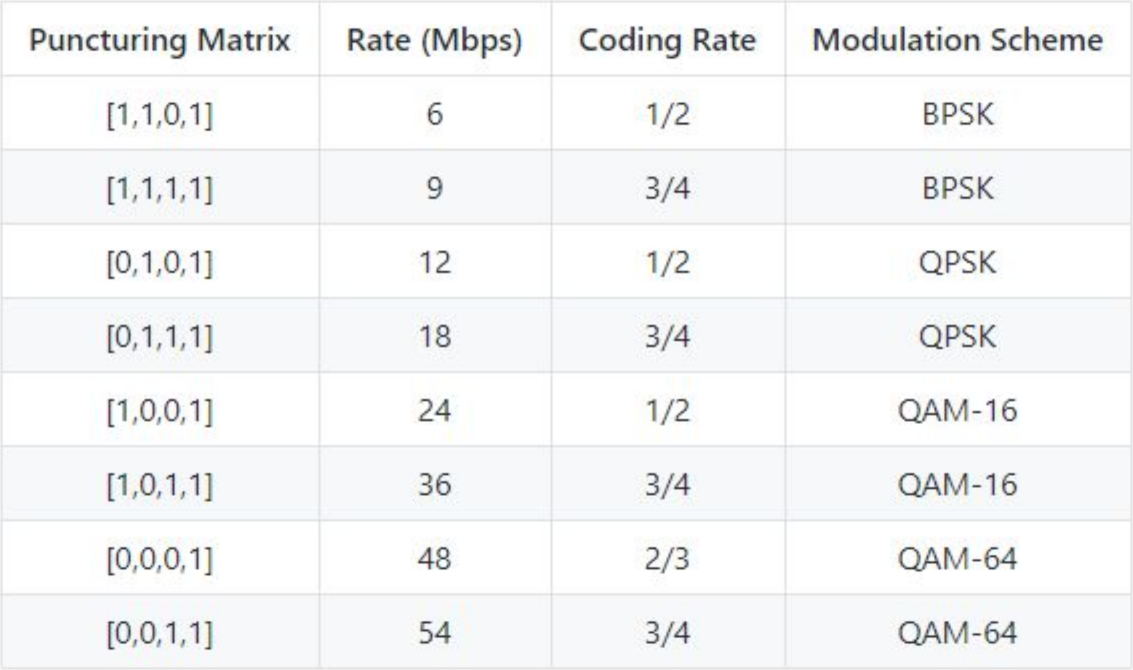
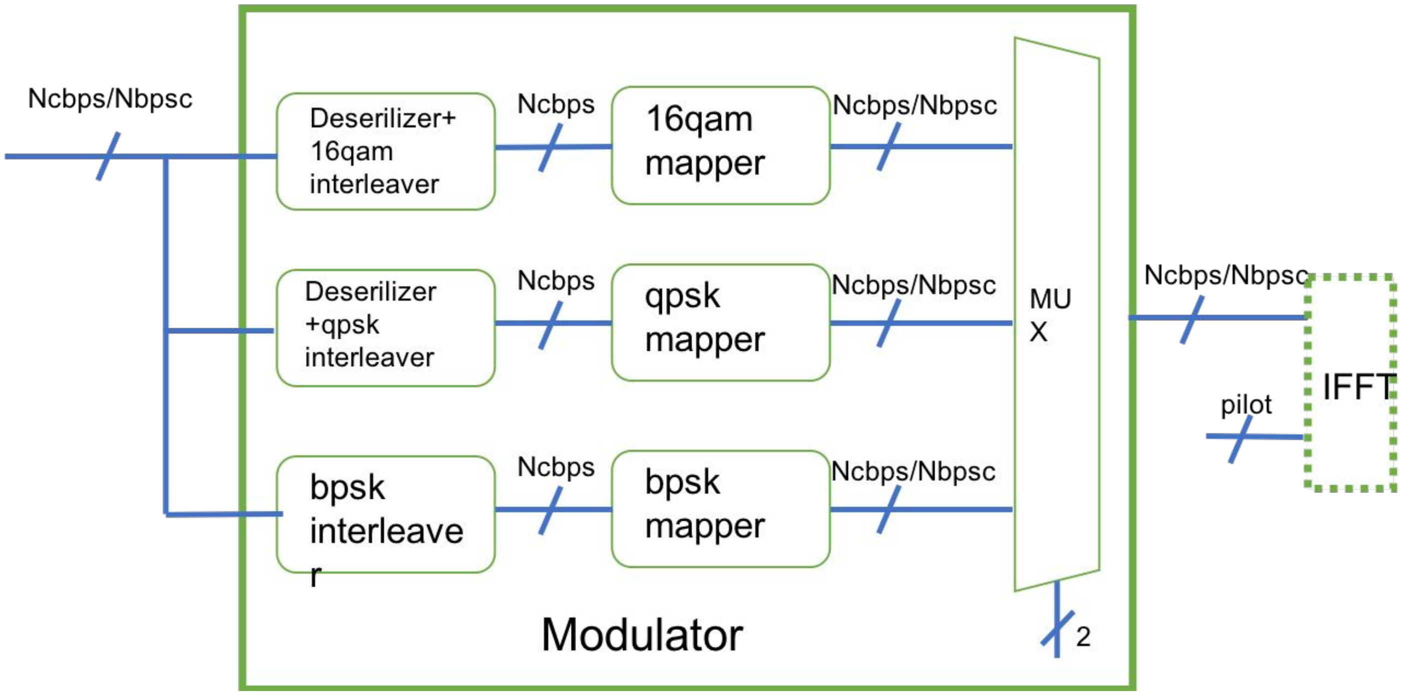

Punctured coding (r = 3/4)
Source data
X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8
Encoded data
A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8
Stolen bit
Bit stolen data (sent/received data)
A_0 B_0 A_1 B_2 A_3 B_3 A_4 B_5 A_6 B_6 A_7 B_8
Bit inserted data
A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8
Inserted dummy bit
Decoded data
y0 y1 y2 y3 y4 y5 y6 y7 y8

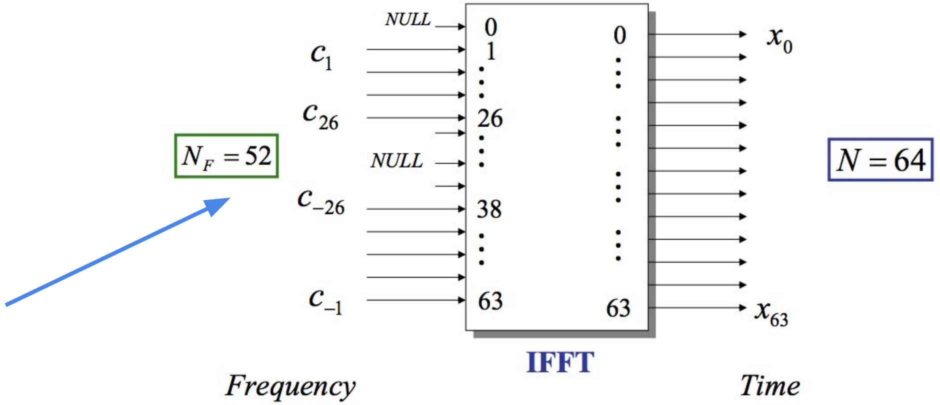
Image from [1]



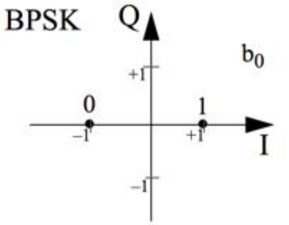
Puncturing Matrix	Rate (Mbps)	Coding Rate	Modulation Scheme
[1,1,0,1]	6	1/2	BPSK
[1,1,1,1]	9	3/4	BPSK
[0,1,0,1]	12	1/2	QPSK
[0,1,1,1]	18	3/4	QPSK
[1,0,0,1]	24	1/2	QAM-16
[1,0,1,1]	36	3/4	QAM-16
[0,0,0,1]	48	2/3	QAM-64
[0,0,1,1]	54	3/4	QAM-64

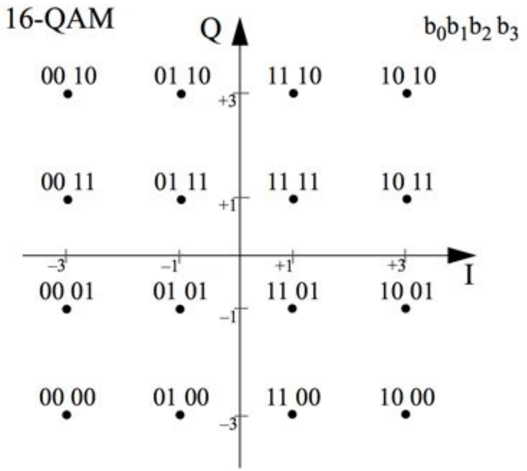
Tx Modulator

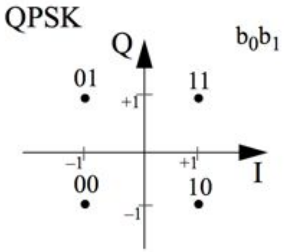




Pilots at: -21, -7, 7, 21

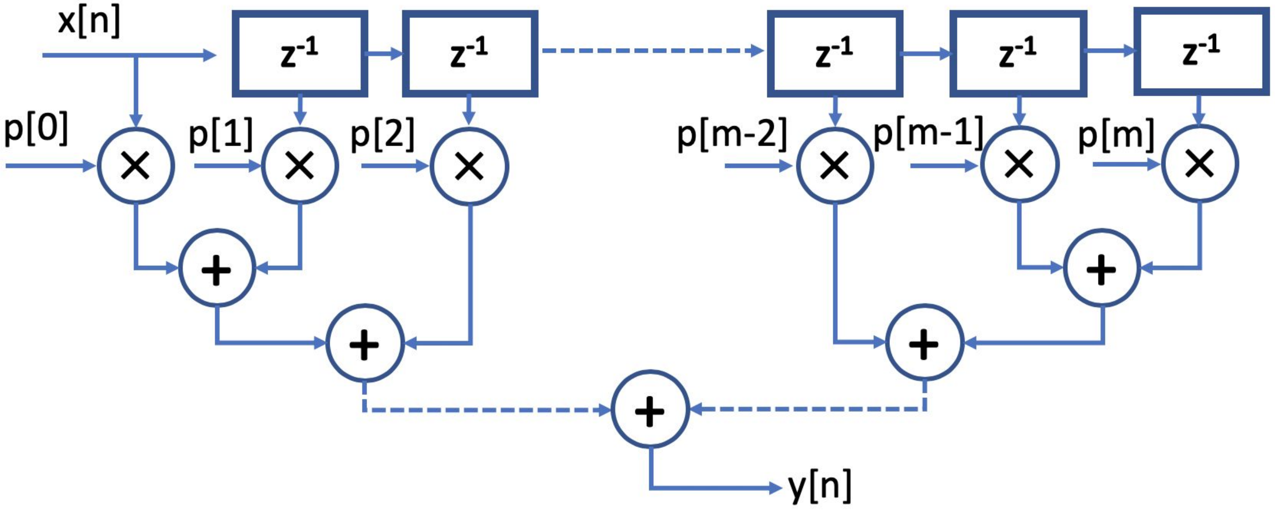




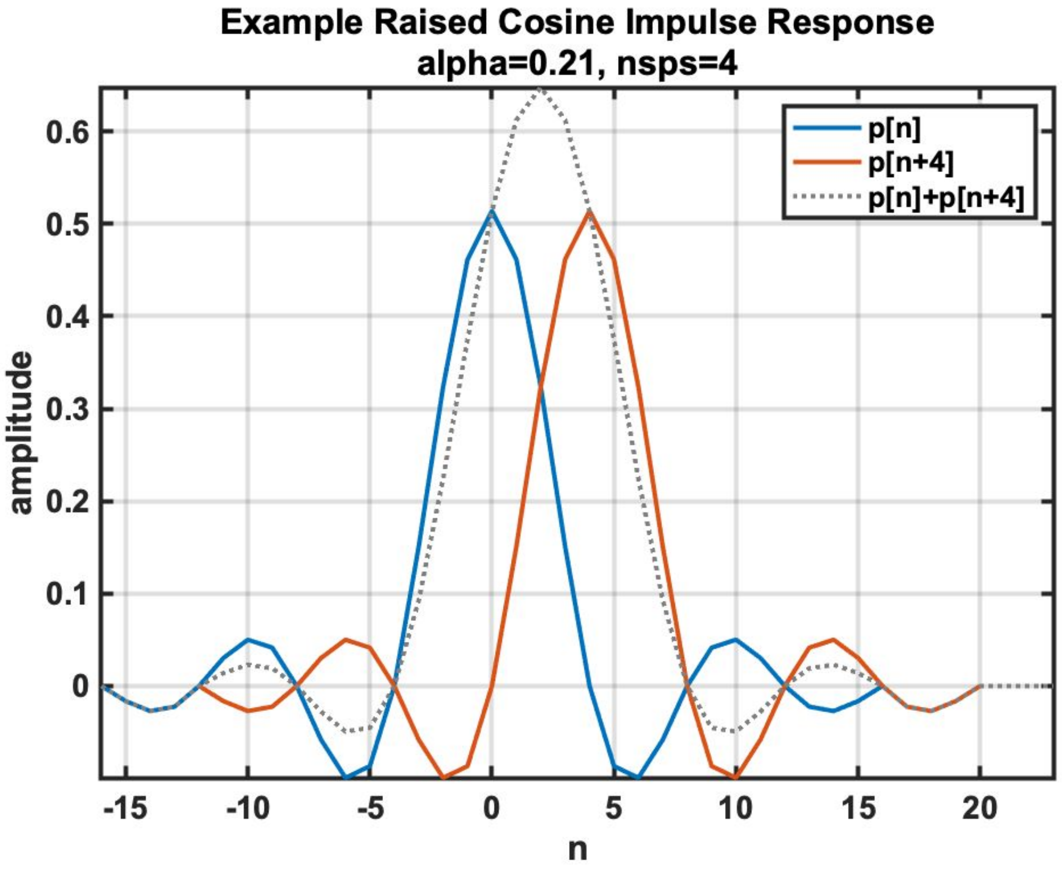


1) interleaver: operate at an OFDM symbol level with a block size of Ncbps
2) Mapper: operates at the OFDM symbol level with a block size of (Ncbps). Each block is divided into sub-blocks at the OFDM sub-carrier level .

Raised-Cosine Filter



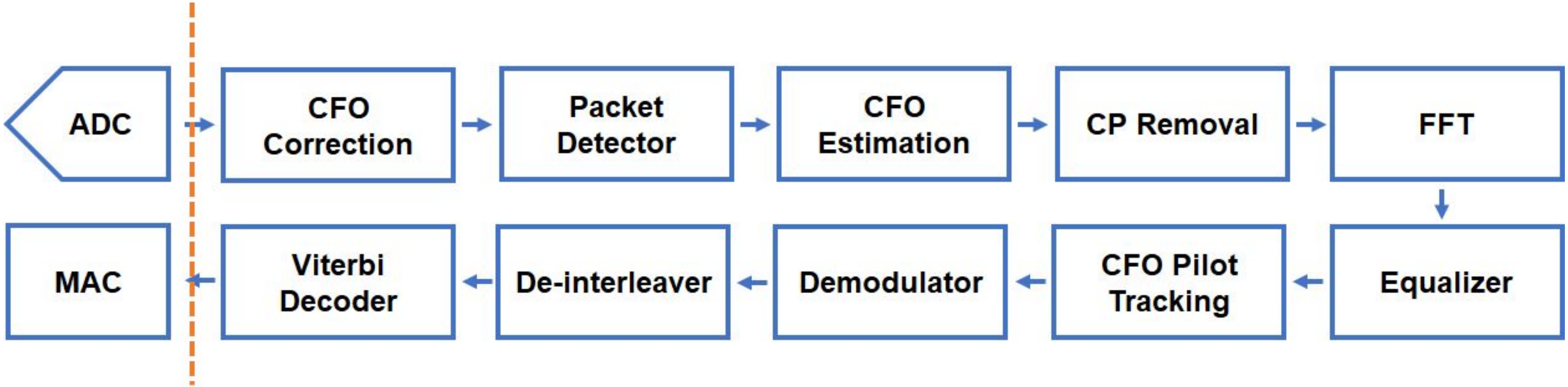
- Direct form FIR architecture
- TreeReduce addition to reduce critical path delay



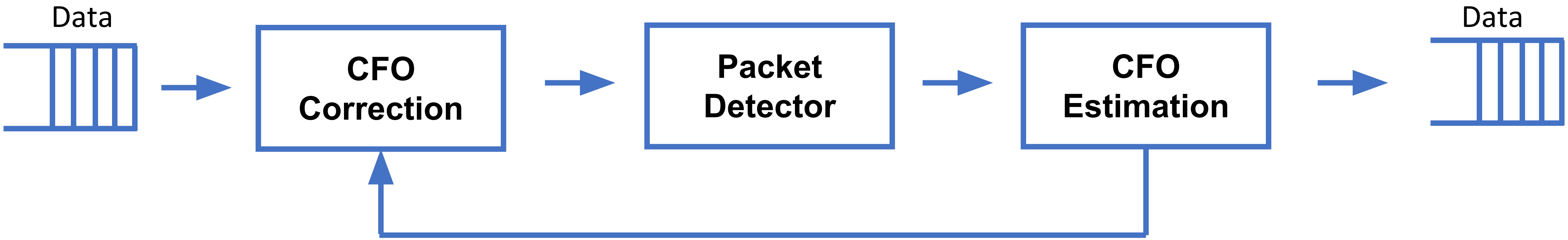
$$p[n] = \frac{\mathrm{sinc}\left(\frac{n}{nsps}\right) * \cos\left(\frac{\pi \alpha n}{nsps}\right)}{1 - \left(\frac{2\alpha n}{nsps}\right)^2}$$

$$n = -(nsps \times npulse), \ldots, (nsps \times npulse)$$

802.11a Receive Chain



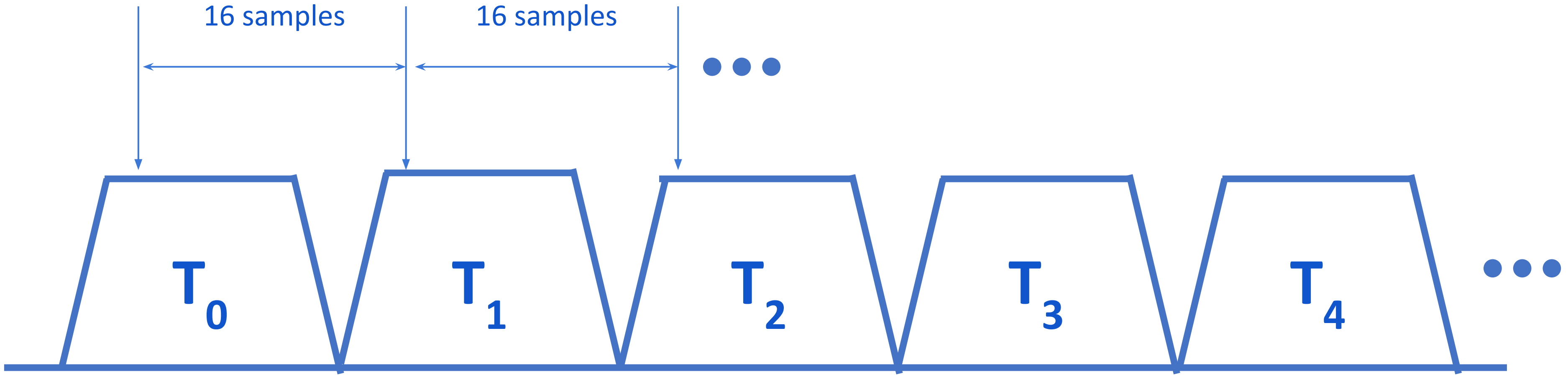
802.11a RX CFO Estimation and Correction



- Data arrives in CFO Correction block and is rotated by phase error
- Data arrives at Packet Detector and packet start/end flags generated
- CFO Estimation gets new phase error estimation and updates correction

E. Sourour, H. El-Ghoroury, and D. McNeill, “Frequency offset estimation and correction in the ieee 802.11a wlan,” in IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004, vol. 7, pp. 4923–4927 Vol. 7, Sept 2004.

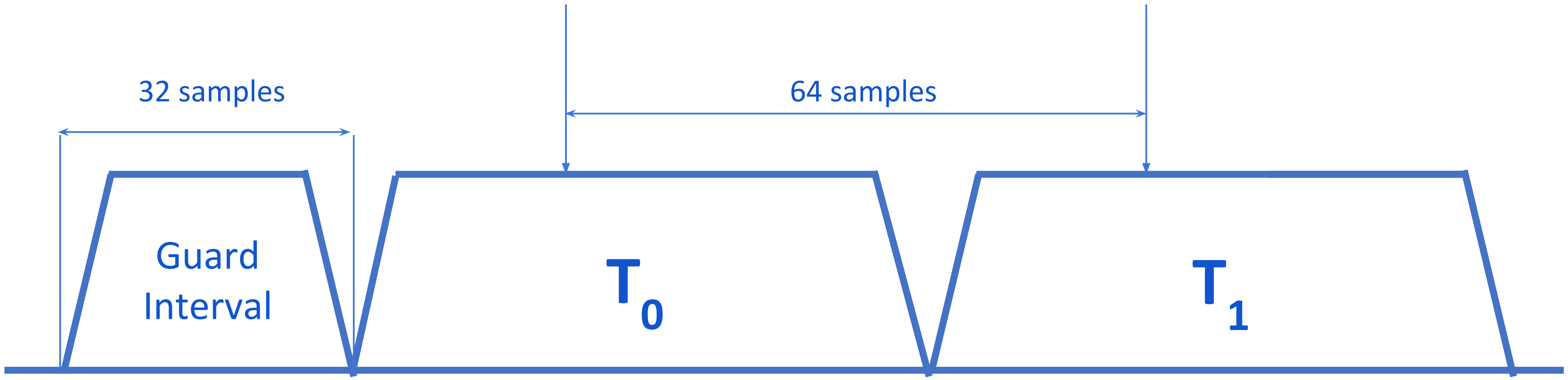
Coarse Offset Estimation



- 802.11a Short Training Field (STF) has 10 repeating sets of a 16-sample pattern
- Every 16 samples the pattern repeats
- By averaging the phase difference between all of the corresponding pairs, can extract per-sample phase estimate
- CFO Estimation is the last block that uses this field, so this is dropped at this stage

E. Sourour, H. El-Ghoroury, and D. McNeill, "Frequency offset estimation and correction in the ieee 802.11a wlan," in IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004, vol. 7, pp. 4923–4927 Vol. 7, Sept 2004.

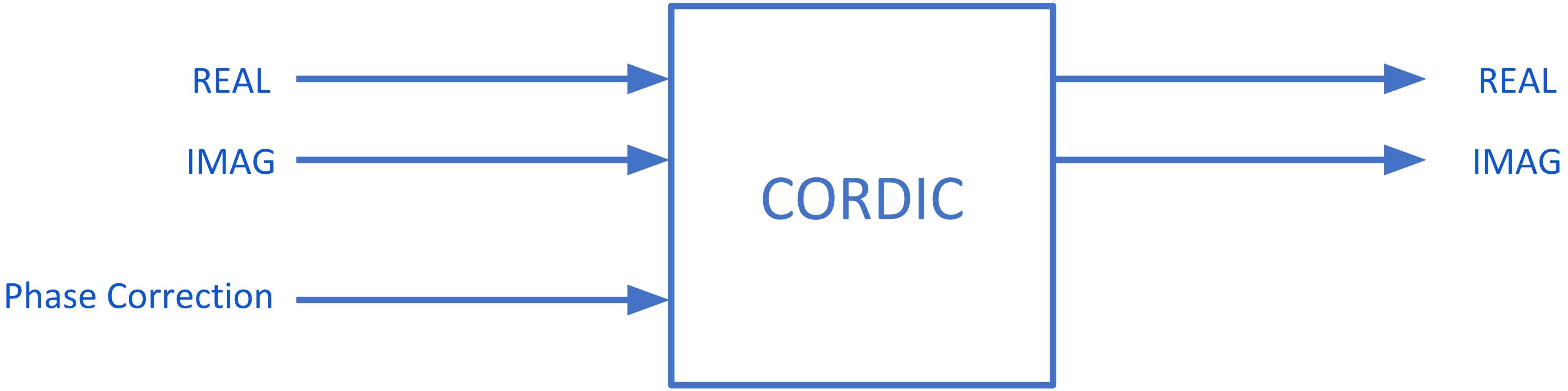
Fine Offset Estimation



- 802.11a Long Training Field (LTF) has 2 repeating sets of a 64-sample pattern
- 32 sample Guard Interval between STF and LTF
- Every 64 samples the pattern repeats
- These samples have the coarse correction applied already, so the long fields are required to get the small residual offset still present after the coarse correction
- Equalizer also uses LTF, so this field is passed through

E. Sourour, H. El-Ghoroury, and D. McNeill, “Frequency offset estimation and correction in the ieee 802.11a wlan,” in IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004, vol. 7, pp. 4923–4927 Vol. 7, Sept 2004.

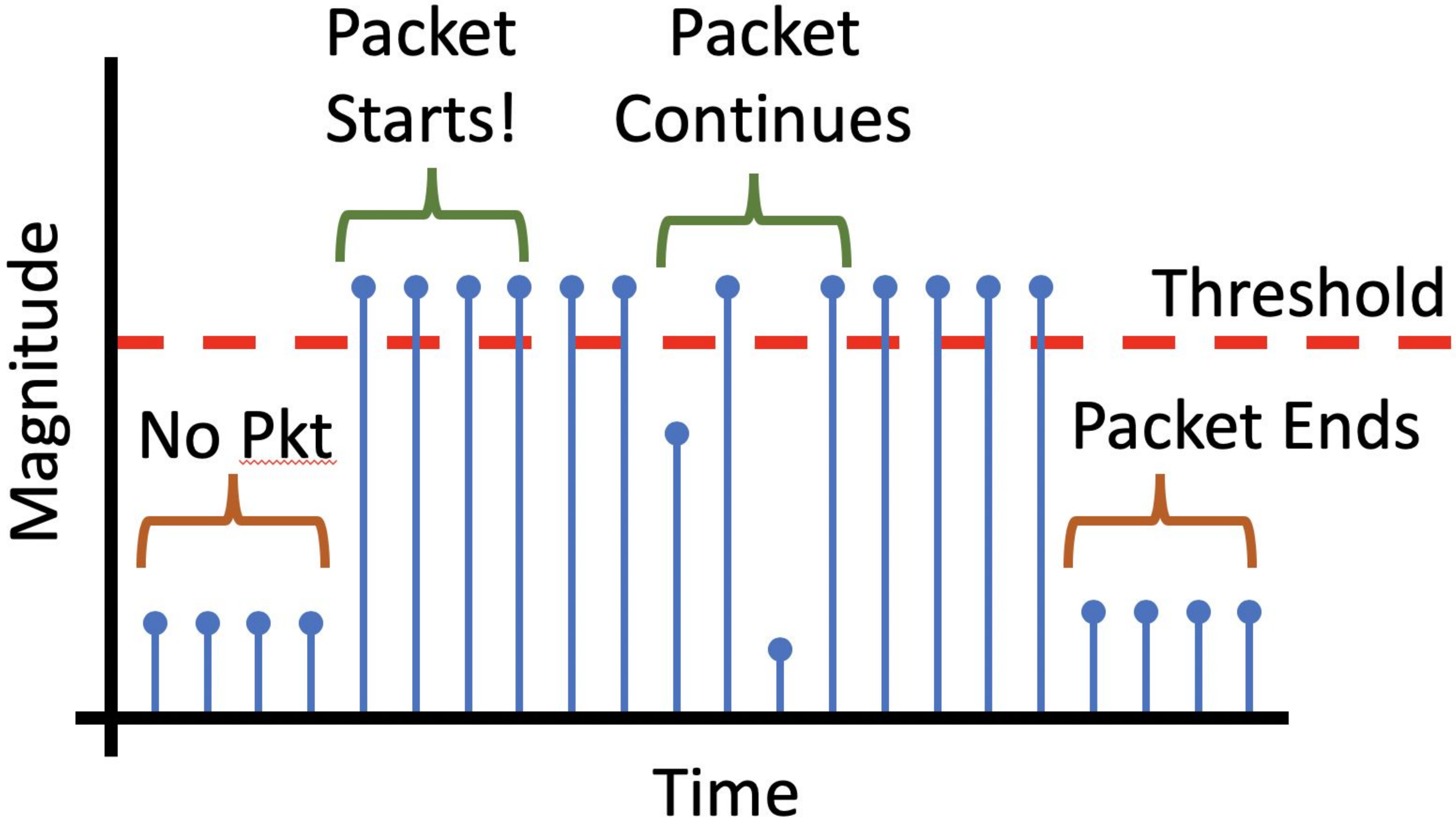
Offset Correction



- Correction is done by CORDIC rotation of input I/Q Vector
- Estimation block gives a per-sample phase correction estimate ($\boldsymbol{\varphi}_0$)
- Each new sample requires more rotation as phase error accumulates
- The next phase correction follows the piecewise function

$$\phi_{i+1} = \begin{cases} (\phi_i + \phi_0) - 2\pi & \phi_i + \phi_0 > \pi \\ (\phi_i + \phi_0) & |\phi_i + \phi_0| \leq \pi \\ (\phi_i + \phi_0) + 2\pi & \phi_i + \phi_0 < -\pi \end{cases}$$

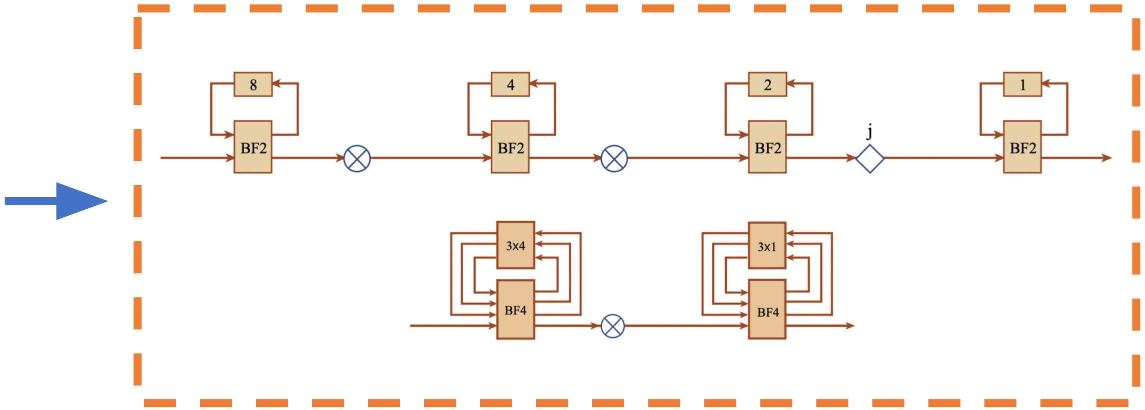
Packet Detector



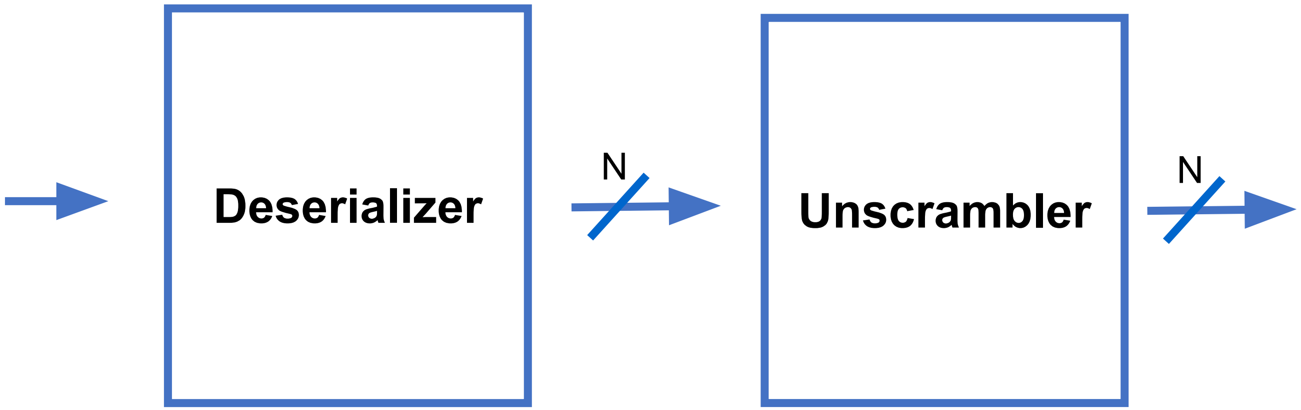
FFT (& IFFT)

- Single-path delay feedback architecture
  - Supports radix-2 and radix-4
  - FFT sizes: powers of 2

SDF Chain



FFT block diagram


Deserializer

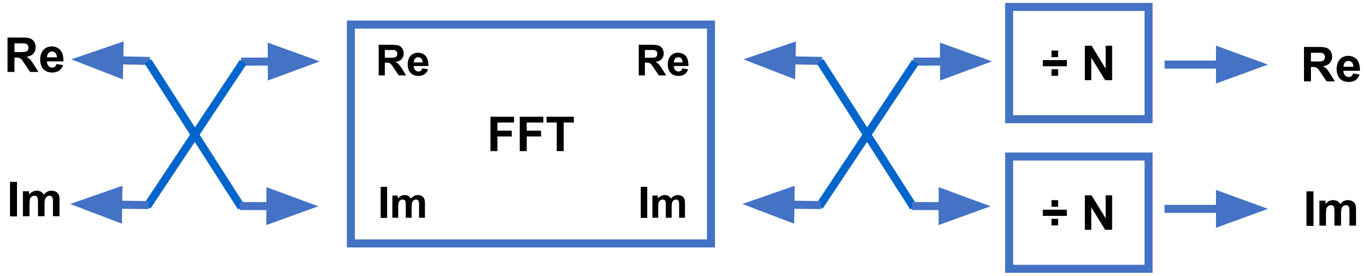
N

Unscrambler

N

- IFFT reuses FFT architecture

IFFT block diagram


Re

Im

Re

Im

FFT

Re

Im

÷ N

÷ N

Re

Im

S. He and M. Torkelson, “A new approach to pipeline FFT processor,” in Proceedings of International Conference on Parallel Processing, pp. 766–770, April 1996.
V. Stojanovic, “Lecture 10: Fast Fourier transform: VLSI architectures,” in 6.973 Communication System Design, 2006. MIT OpenCourseWare.

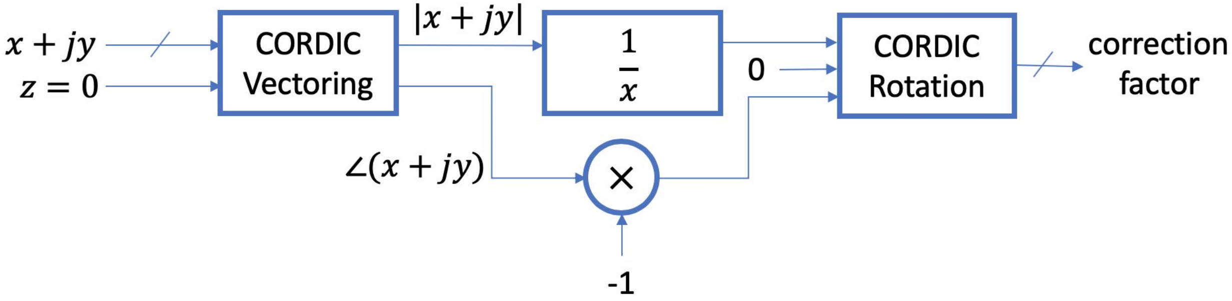
Zero-Forcing Equalizer

- Completely eliminates ISI

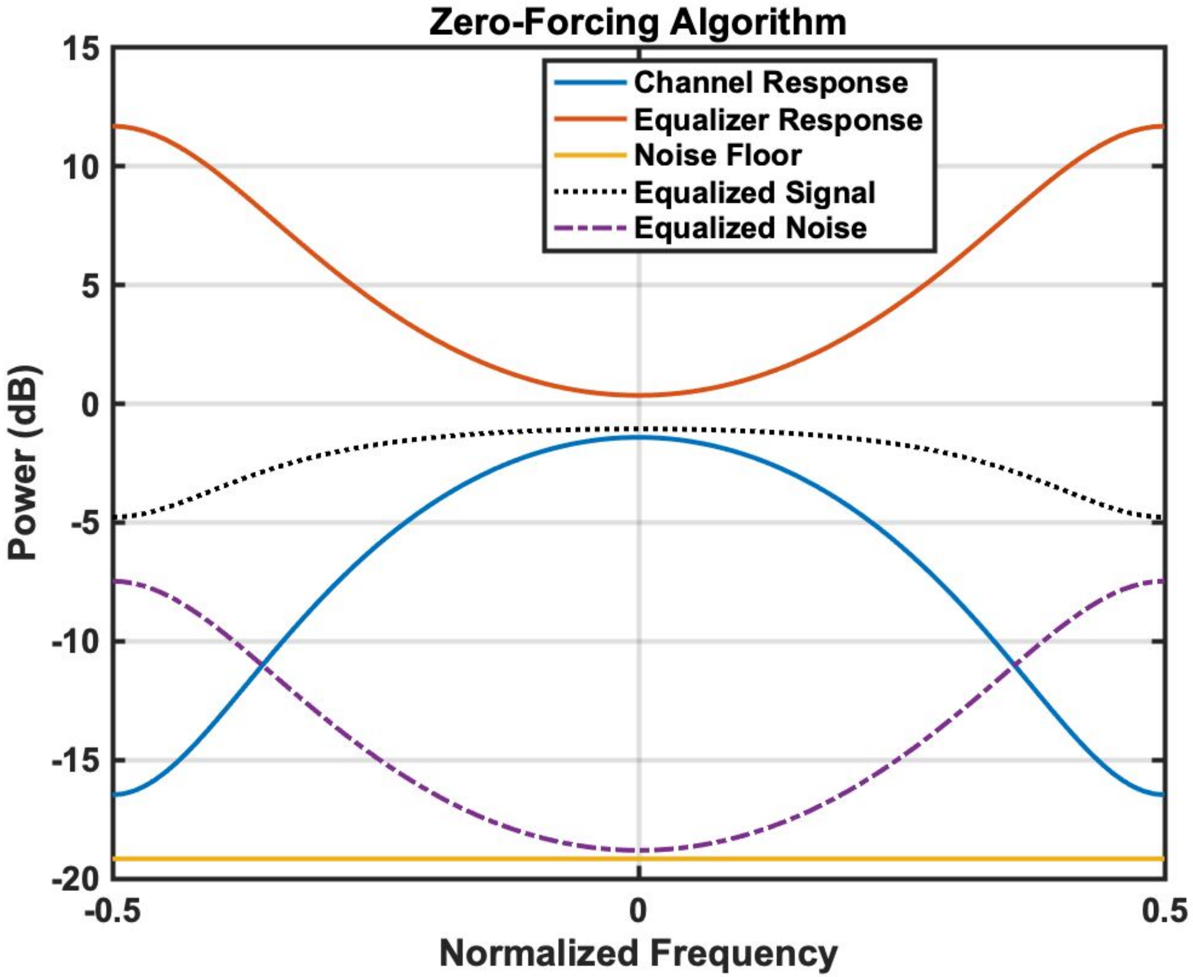
BUT…

- Amplifies noise
- Reduces overall SNR

Channel Inversion

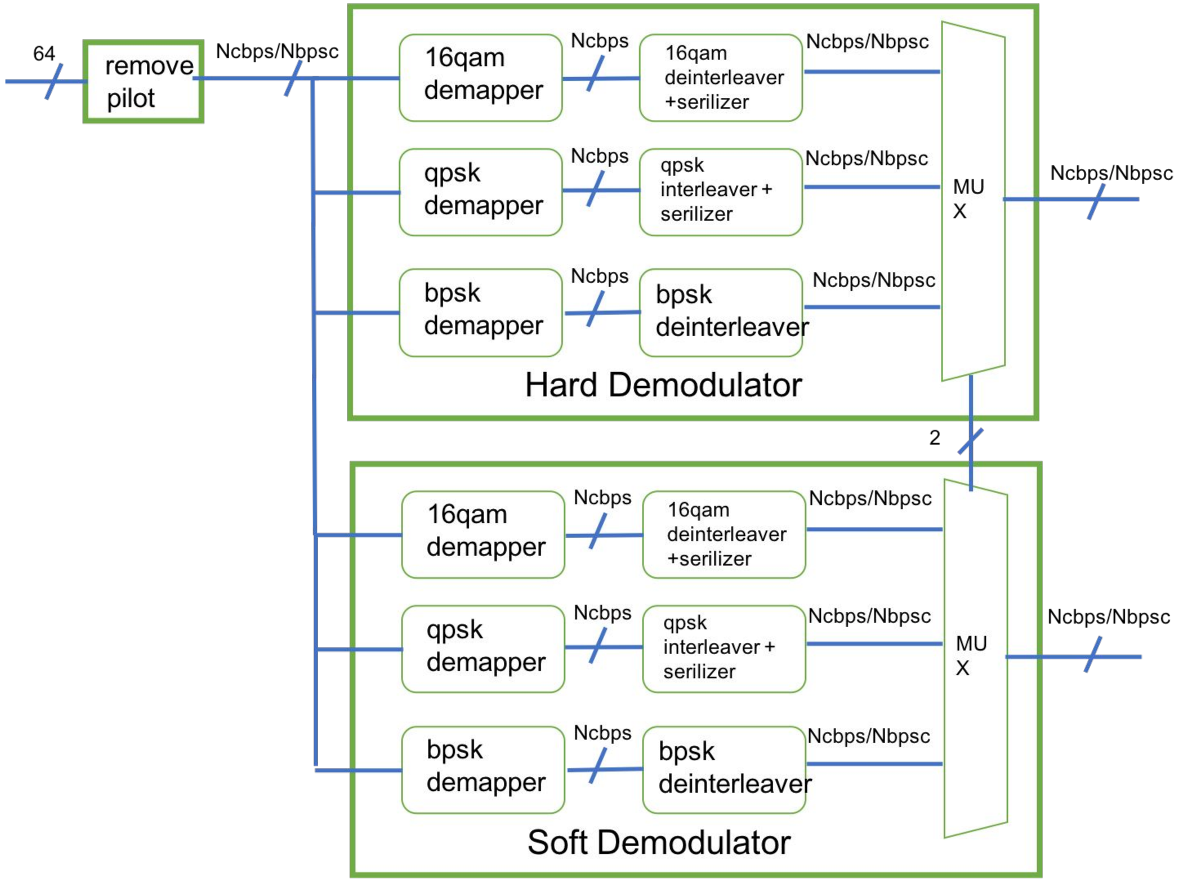


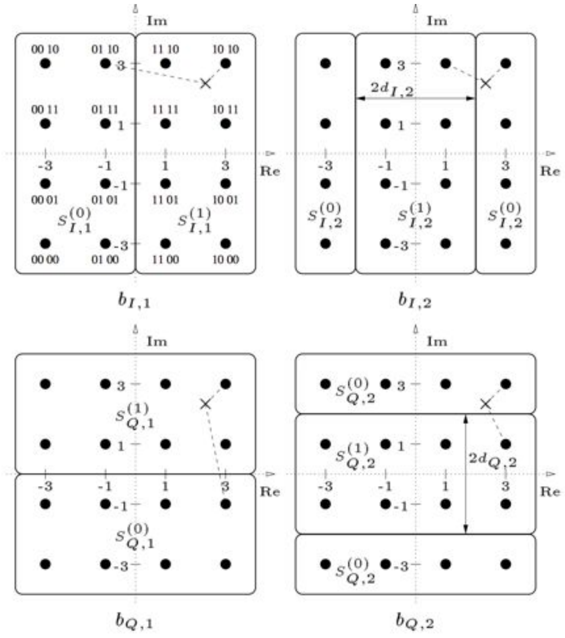
$$C(\omega) = \frac{Q(\omega)}{H(\omega)}$$



J. W. M. Bergmans, *Linear Equalization*. Springer-Science+Business Media, B.V., 1996.

Rx Demodulator





Partitions of 16qam constellation

Optimal soft demapping:

-include logarithmic and exponential functions

-not suitable for hardware implementation

Sub-optimal soft demapping

-low complexity

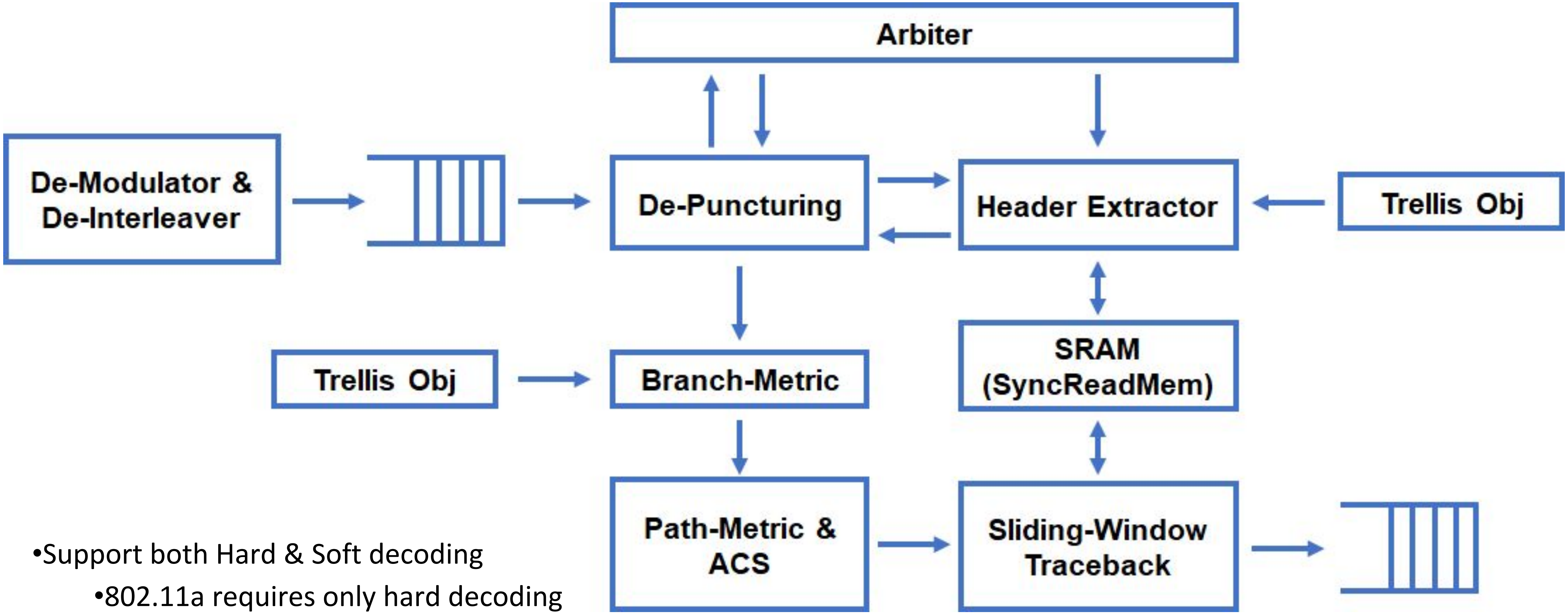
-negligible soft decoding performance loss

$$\begin{aligned} LLR(b_{I,k}) &= \log \frac{P\left[b_{I,k}=1 \mid r[i]\right]}{P\left[b_{I,k}=0 \mid r[i]\right]} \\ &= \log \frac{\sum_{\alpha \in S^{(1)}_{I,k}} P\left[a[i]=\alpha \mid r[i]\right]}{\sum_{\alpha \in S^{(0)}_{I,k}} P\left[a[i]=\alpha \mid r[i]\right]} \end{aligned}$$

$$LLR(b_{I,k}) = \\ = \frac{|G_{ch}(i)|^2}{4} \left\{ \min_{\alpha \in S^{(0)}_{I,k}} |y[i]-\alpha|^2 - \min_{\alpha \in S^{(1)}_{I,k}} |y[i]-\alpha|^2 \right\}$$

$$p\left(r[i] \mid a[i]=\alpha\right) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2}\frac{|r[i]-G_{ch}(i)\,\alpha|^2}{\sigma^2}\right\}$$

802.11a Rx Decoder



- Support both Hard & Soft decoding
    - 802.11a requires only hard decoding
- Two Viterbi decoding algorithms are used
    - Normal Viterbi Decoding
    - Sliding-Window Viterbi Decoding

802.11a PPDU Frame Structure

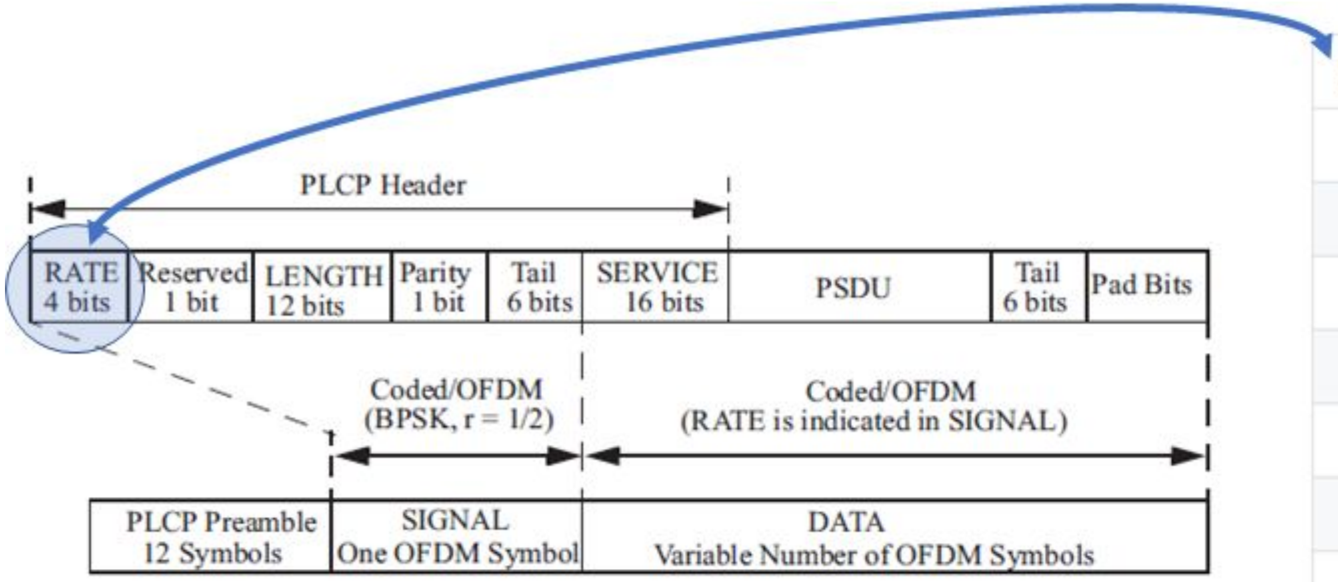
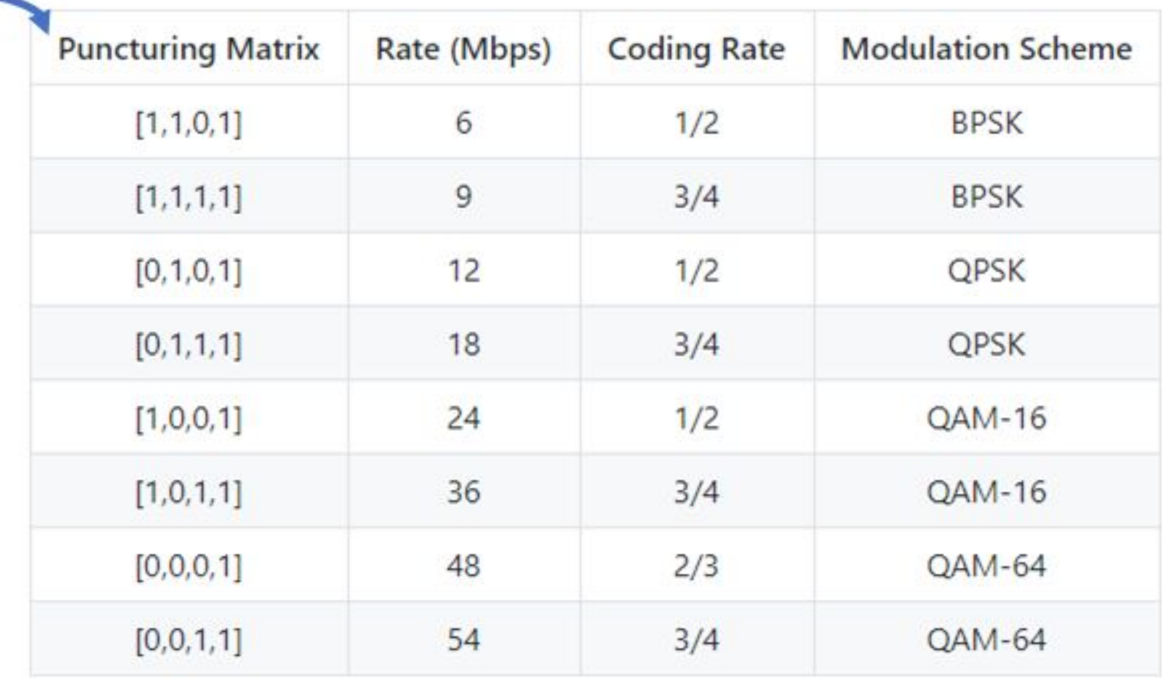


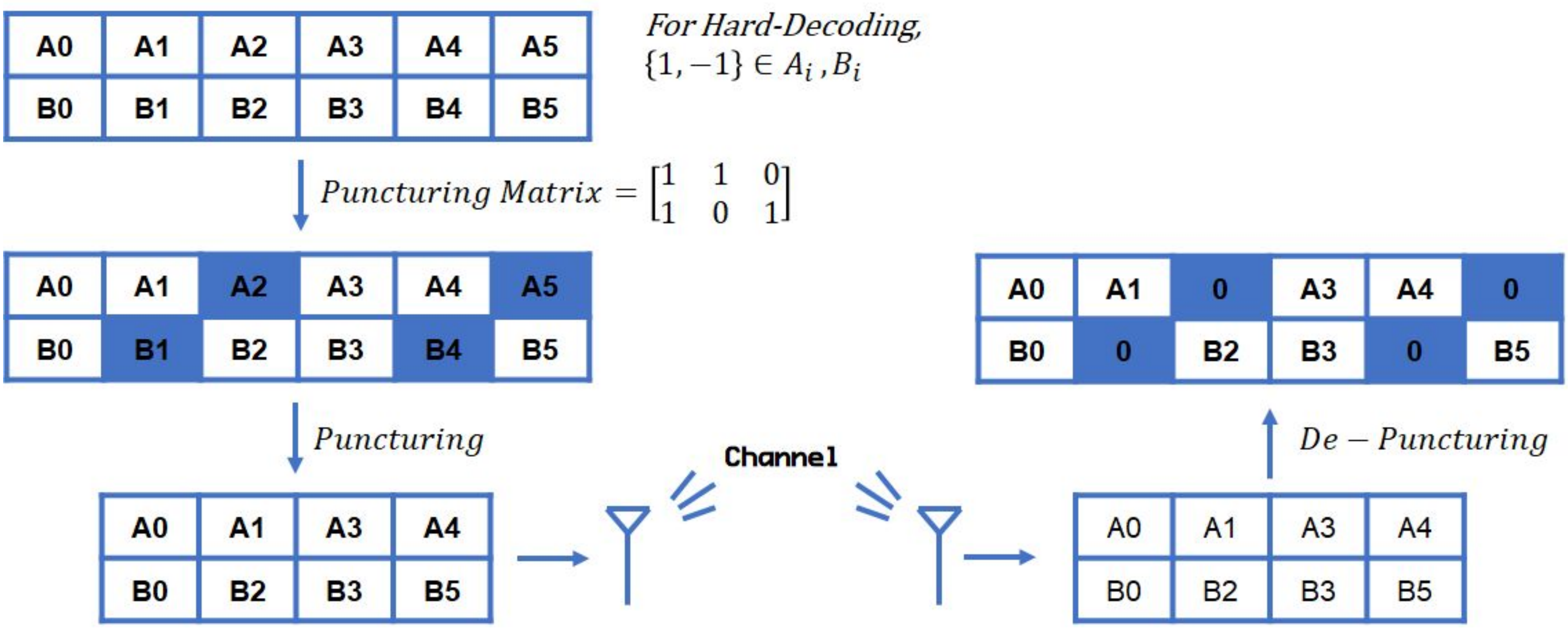
Image from [3]



Puncturing Matrix	Rate (Mbps)	Coding Rate	Modulation Scheme
[1,1,0,1]	6	1/2	BPSK
[1,1,1,1]	9	3/4	BPSK
[0,1,0,1]	12	1/2	QPSK
[0,1,1,1]	18	3/4	QPSK
[1,0,0,1]	24	1/2	QAM-16
[1,0,1,1]	36	3/4	QAM-16
[0,0,0,1]	48	2/3	QAM-64
[0,0,1,1]	54	3/4	QAM-64

- Length, modulation scheme, and coding rate are specified in header
    - Length is in octet
- PLCP Header is always coded with BPSK modulation and ½ coding rate
- Header information must be decoded before the next OFDM symbol is available
    - Requires minimum latency Viterbi Decoder for header extraction

De-Puncturing



For Hard-Decoding,
$\{1,-1\} \in A_i\,, B_i$

A0	A1	A2	A3	A4	A5
B0	B1	B2	B3	B4	B5

$$Puncturing\ Matrix = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

A0	A1	A2	A3	A4	A5
B0	B1	B2	B3	B4	B5

Puncturing

A0	A1	A3	A4
B0	B2	B3	B5

Channel

A0	A1	A3	A4
B0	B2	B3	B5

$De - Puncturing$

A0	A1	0	A3	A4	0
B0	0	B2	B3	0	B5

Maximum Likelihood Sequence Estimator

Generator Polynomial = (7 ,6)

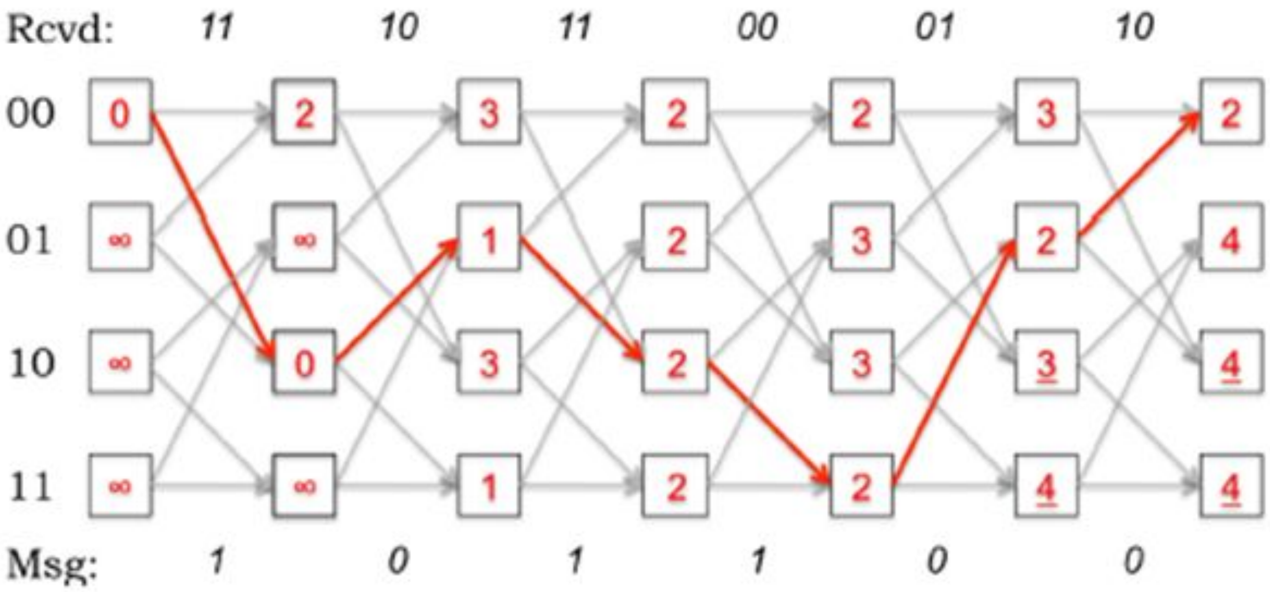


Image from [2]

Path metrics:

$$\begin{bmatrix}0 & 1 & 2 & 3 & 4 & 5 & 6\end{bmatrix}$$

$$\begin{bmatrix}0 & 2 & 3 & 2 & 2 & 3 & 2\\ \infty & \infty & 1 & 2 & 3 & 2 & 4\\ \infty & 0 & 3 & 2 & 3 & 3 & 4\\ \infty & \infty & 1 & 2 & 2 & 4 & 4\end{bmatrix}$$

Survival path:

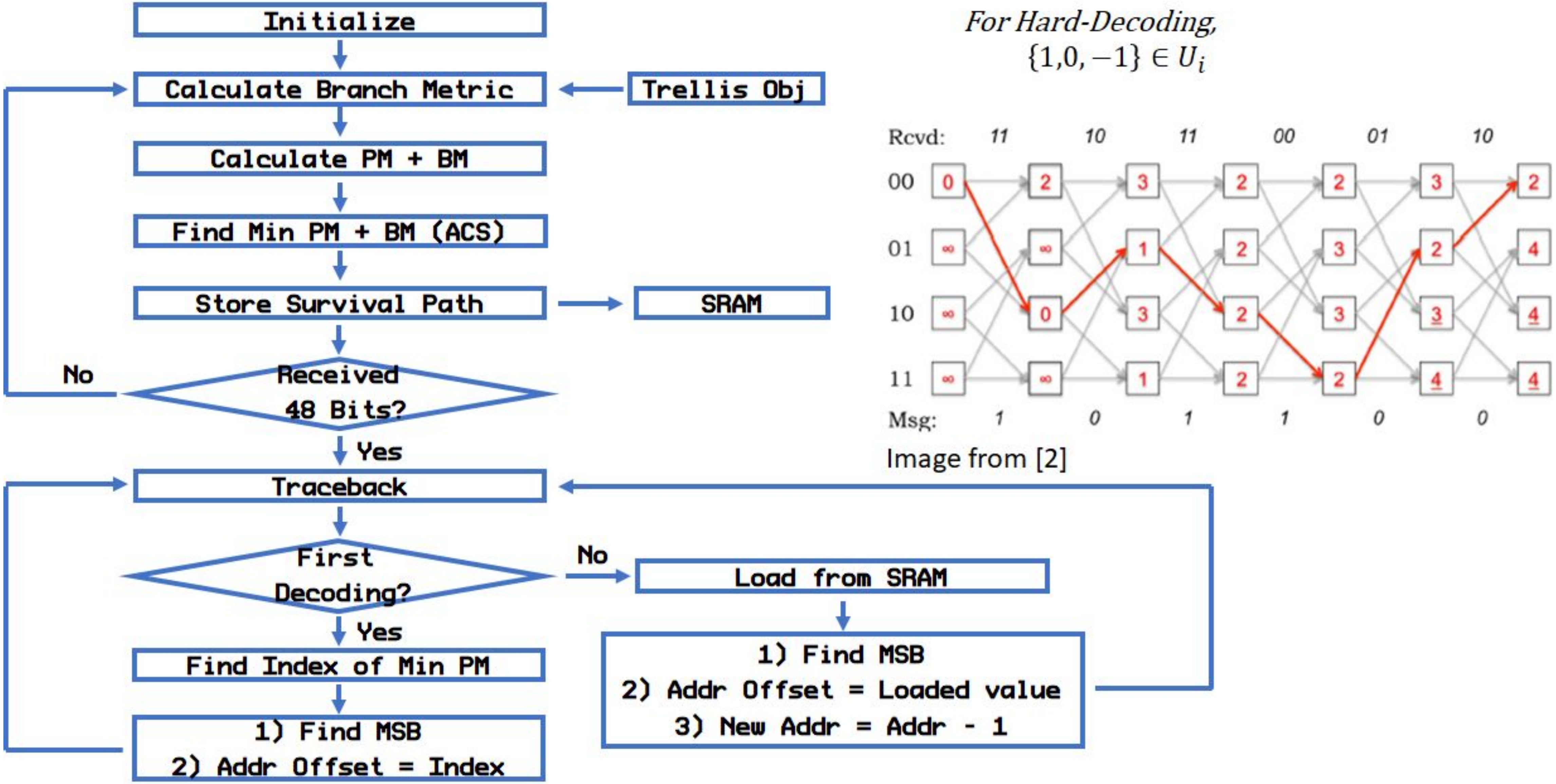
$$\begin{bmatrix}0 & 0 & 1 & 0 & 0 & 1\\ x & 2 & 3 & 3 & 3 & 2\\ 0 & 0 & 1 & 1 & 1 & 1\\ x & 2 & 3 & 2 & 3 & 3\end{bmatrix}$$

Take MSBs

Decoded results:

$$[1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0]$$

Viterbi Decoder Algorithm Flow Chart



Initialize

Calculate Branch Metric

Trellis Obj

Calculate PM + BM

Find Min PM + BM (ACS)

Store Survival Path

SRAM

Received 48 Bits?

No

Yes

Traceback

First Decoding?

No

Load from SRAM

1) Find MSB
2) Addr Offset = Loaded value
3) New Addr = Addr - 1

Yes

Find Index of Min PM

1) Find MSB
2) Addr Offset = Index

For Hard-Decoding,
$\{1,0,-1\} \in U_i$

Image from [2]

How to determine bit width for PM Register?

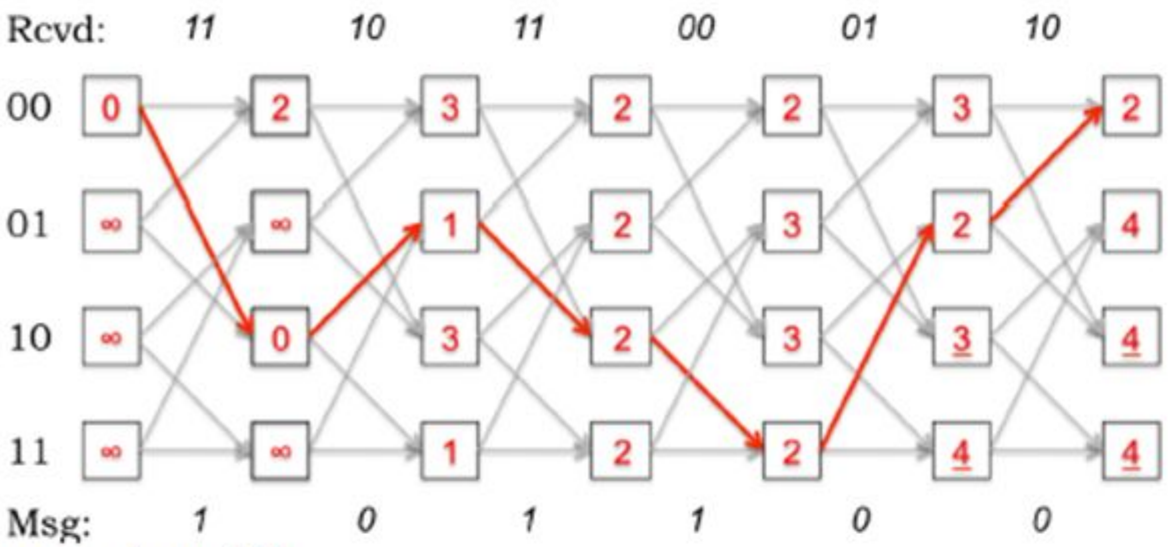


Image from [2]

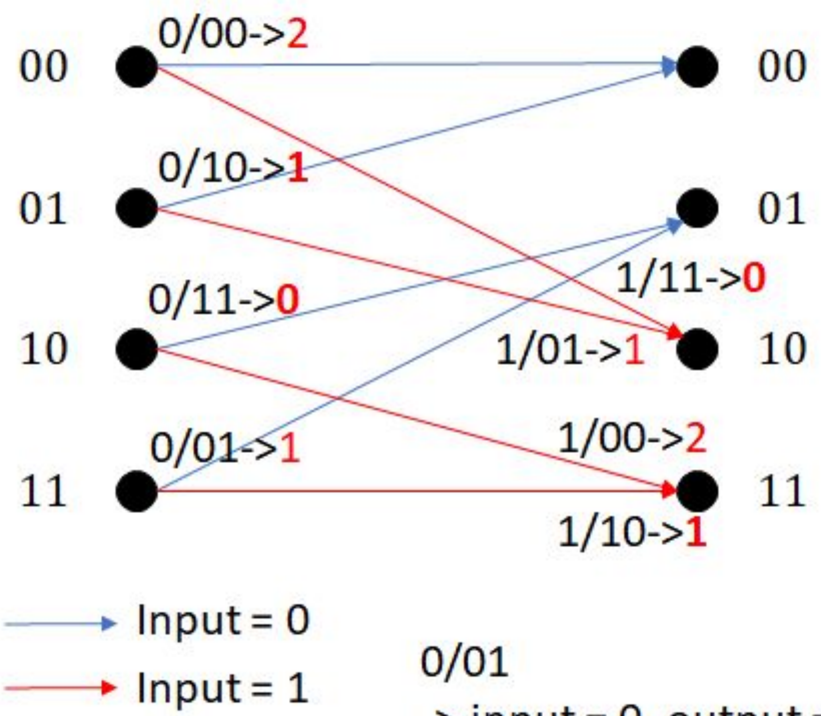
- PM values are accumulated over the trellis
- Length of bit sequence can vary from 1 to 32768 (2^{15})
- Only minimum PM value matters at the end of trellis
  - We do not care about absolute number
- Simple solution can be applied:
  - Whenver the min PM exceeds ‘x’, subtract ‘x’ from all elements
  - Currently x=16

Hard Decoding vs Soft Decoding

Hard Decision = Hamming distance calculation

Type = SInt(2.W)

Received input = 11



0/01

-> input = 0, output = 01

Soft Decision = Euclidean distance calculation:

Type = FixedPoint(bitWidth, (bitWidth-3).BP)

$$BM_{Soft} = \sum_{i=1}^{p} (u_i - v_i)^2$$

$$u_1, u_2, \dots, u_p = expected\ p\ parity\ bits$$

$$v_1, v_2, \dots, v_p = p\ analog\ samples$$

How can we reduce computation?

$$\sum_{i=1}^{p} (u_i - v_i)^2 = \sum_{i=1}^{p} u_i^2 - 2u_i v_i + v_i^2$$

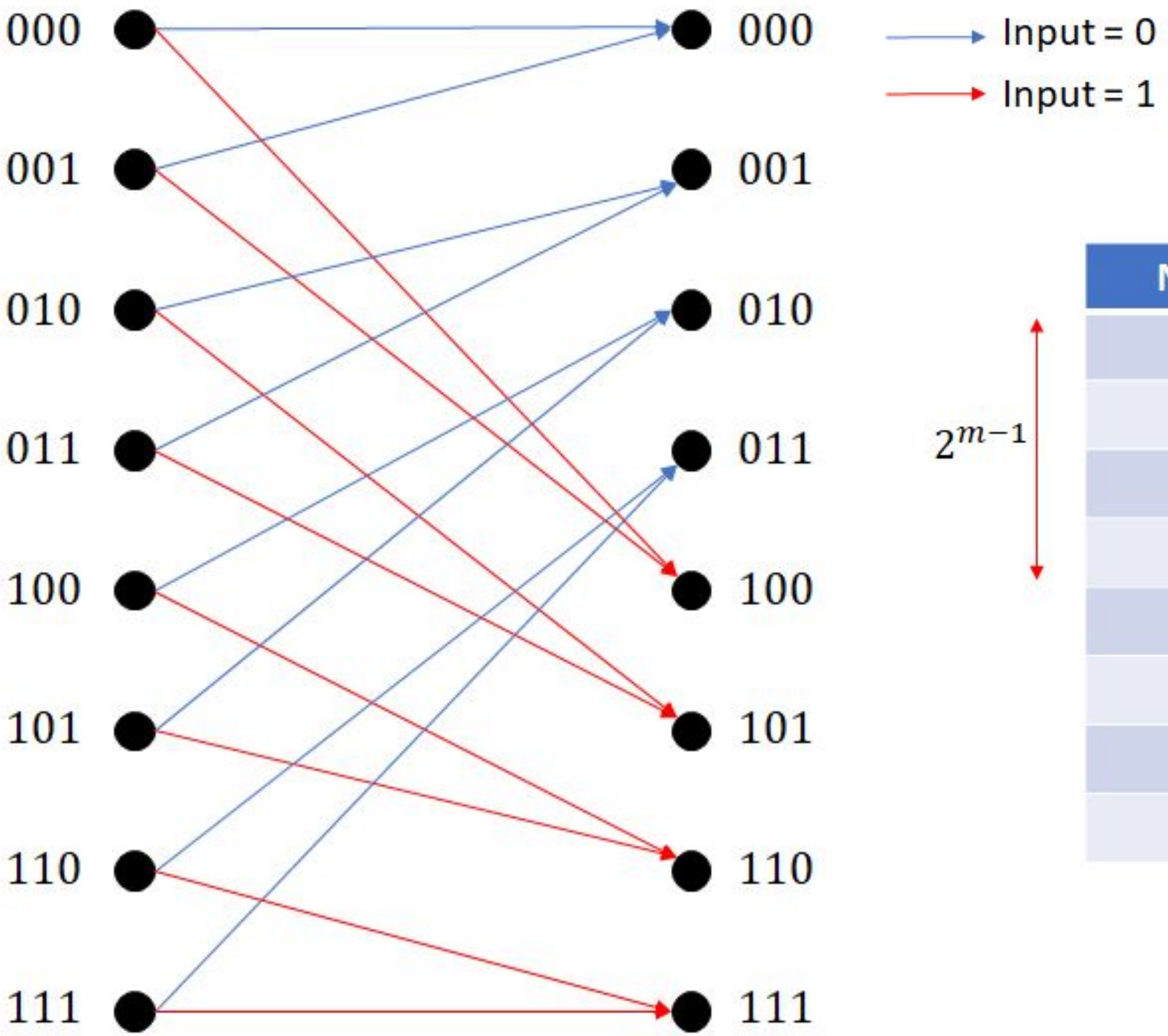
$$u_i\ are\ known\ and\ v_i\ are\ common\ term$$

$$\rightarrow \sum_{i=1}^{p} -u_i v_i$$

$$or$$

$$\rightarrow \min(\sum_{i=1}^{p} sgn(u_i) \cdot (v_i - u_i))$$

How to ease ACS computation?



2^{m-1}

Next State	Current State 1	Current State 2
0	0	1
1	2	3
2	4	5
3	6	7
4	0	1
5	2	3
6	4	5
7	6	7

Sliding Window Viterbi Decoder

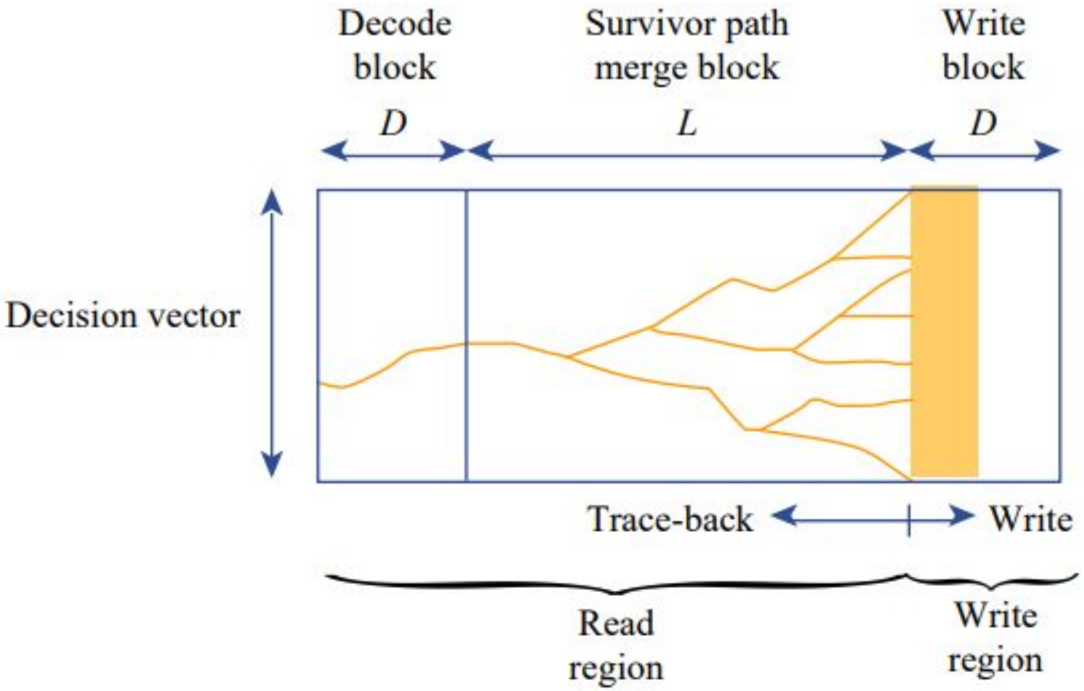


Image from [2]

- Survival path merges after 5*K steps
- Writing and Traceback should be working in parallel
  - $\# \; of \; read \; port \; \cong \frac{L}{D} + 2$
- Reduces memory size at the cost of complexity

The Chisel Difference

- Concise code
  - 8,700 lines of implementation
  - 8,700 lines of testing
  - 1-2 orders of magnitude less than the Verilog versions
- Agile generators
  - Can reconfigure for arbitrary number of subcarriers
  - Can reconfigure for arbitrary (convolutional) coding schemes
  - And more...
- Rapid unit testing!

Reference

[1] ISO/IEC/IEEE 8802-11:2012

[2] MIT opencourseware 6.973

[3] Deepthi Ancha, Implementation of Tail-Biting Convolutional codes, 2008