Modeling a 4G LTE System in MATLAB

Idin Motedayen-Aval
Senior Applications Engineer
MathWorks
Idin.motedayen-aval@mathworks.com
Agenda

- 4G LTE and LTE Advanced
  - True Global standard
  - True Broadband mobile communications
  - How it was achieved?
  - What are the challenges?

- MATLAB and communications system design
  - Modeling and simulation
  - Simulation acceleration
  - Path to implementation

- Case study:
  - Physical layer modeling of an LTE system in MATLAB

- Summary
Modeling a 4G LTE System in MATLAB

Part 1:
Modeling & simulation
4G LTE and LTE Advanced
4G LTE and LTE Advanced
Distinguishing Features

- Motivation
  - Very high capacity & throughput
  - Support for video streaming, web browsing, VoIP, mobile apps

- A true global standard
  - Contributions from all across globe
  - Deployed in AMER, EMEA, APLA
4G LTE and LTE Advanced
Distinguishing Features

- A true broadband mobile standard
  - From 2 Mbps (UMTS)
  - To 100 Mbps (LTE)
  - To 1 Gbps (LTE Advanced)

<table>
<thead>
<tr>
<th>Standard</th>
<th>Low Mobility</th>
<th>High Mobility</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDGE</td>
<td>~250 kbps</td>
<td></td>
</tr>
<tr>
<td>WCDMA/UMTS</td>
<td>2 Mbps</td>
<td>384 kbps</td>
</tr>
<tr>
<td>HSDPA</td>
<td>14 Mbps</td>
<td></td>
</tr>
<tr>
<td>HSPA+</td>
<td>42 Mbps</td>
<td></td>
</tr>
<tr>
<td>LTE (R8 or R9)</td>
<td>100 Mbps</td>
<td></td>
</tr>
<tr>
<td>4G Requirement(*)</td>
<td>1 Gbps</td>
<td>100 Mbps</td>
</tr>
<tr>
<td>LTE Advanced(*)</td>
<td>&gt; 1 Gbps</td>
<td>&gt; 100 Mbps</td>
</tr>
</tbody>
</table>
How is this remarkable advance possible?

- Integration of enabling technologies with sophisticated mathematical algorithms
  - OFDM
  - MIMO (multiple antennas)
  - Turbo Coding

- Smart usage of resources and bandwidth
  - Adaptive modulation
  - Adaptive coding
  - Adaptive MIMO
  - Adaptive bandwidth
What MATLAB users care about LTE?

- **Academics**
  - Researchers contributing to future standards
  - Professors
  - Students

- **Practitioners**
  - System Engineers
  - Software designers
  - Implementers of wireless systems

- Challenge in interaction and cooperation between these two groups

- MATLAB is their common language
Challenges: From specification to implementation

- Simplify translation from specification to a model as blue-print for implementation
- Introduce innovative proprietary algorithms
- System-level performance evaluation
- Accelerate large scale simulations
- Address gaps in the implementation workflow
Where does MATLAB fit in addressing these challenges?

- MATLAB and Communications System Toolbox are ideal for LTE algorithm and system design
- MATLAB and Simulink provide an environment for dynamic & large scale simulations
- Accelerate simulation with a variety of options in MATLAB
- Connect system design to implementation with
  - C and HDL code generation
  - Hardware-in-the-loop verification
Communications System Toolbox

Over 100 algorithms for

- Modulation, Interleaving, Channels, Source Coding
- Error Coding and Correction
- MIMO, Equalizers, Synchronization
- Sources and Sinks, SDR hardware

Algorithm libraries in MATLAB

Algorithm libraries in Simulink
4G LTE and LTE Advanced

Existing demos of Communications System toolbox

Work in-progress
Case Study: Downlink physical layer of LTE (Release 10)
What we will do today

1. Start modeling in MATLAB
2. Iteratively incorporate components such as Turbo Coding, MIMO, OFDM and Link Adaptation
3. Put together a full system model with MATLAB & Simulink
4. Accelerate simulation speed in MATLAB
5. Path to implementation
Modeling and Simulation
LTE Physical layer model in standard

- Downlink Shared Channel
  - Transport channel
  - Physical channel

Reference: 3GPP TS 36 211 v10 (2010-12)
LTE Physical layer model in MATLAB
LTE Physical layer model in MATLAB

>> Open LTE system model

Turbo Channel Coding

MIMO

OFDMA

Adaptation of everything

Channel bandwidth (MHz):
20

Antenna configuration:
2x2
Overview of Turbo Coding

- Error correction & coding technology of LTE
- Performance: Approach channel capacity (Shannon bound)
- Evolution of convolutional coding
- Based on an iterative decoding scheme
MATLAB Demo

Modeling and Simulation of a transceiver system, showcasing:

Coding Gain:
- Convolutional Encoding & Viterbi decoding
- Hard and Soft Decision decoding
- Turbo Encoding & Decoding

```
function [ber bits]=fcn_04PSK_Viterbi(EbNo,MaxNumErrs,MaxNumBits)

%% Initialization
% Components
persistent hModulator hAWGN hDeModulator hBitError
persistent hConvEncoder hViterbi
if isempty(hModulator)
    hModulator = comm.PSKModulator(...
        'ModulationOrder',4, ...
        'PhaseOffset',0, ...
        [BITs,...]
    hBitError = comm.ErrorRate;
end

% Parameters

%=comm.ConvolutionalEncoder(

% = comm.ViterbiDecoder(…
  ‘InputFormat’, ‘Hard’,…
```

Convolutional Coding + Viterbi Decoding

```
  theory - Hard decision
  simulation - Hard decision

BER

$E_b/N_0$ (dB)
```
MATLAB Demo

Modeling and Simulation of a transceiver system, showing Coding Gain:
- Convolutional Encoding & Viterbi decoding
- Hard and Soft Decision decoding
- Turbo Encoding & Decoding

```
function [ber_bits]=fcn_04PSK_Viterbi_Soft(EbNo,MaxNumErrs,MaxNumBits)
%% Initialization
persistent hModulator hAWGN hDeModulator hBitError
persistent hConvEncoder hViterbi hQuantizer
if isempty(hModulator)
    hModulator = comm.PSKModulator(...
        'ModulationOrder',4,...
        'PhaseOffset',0,...
    );
end
hDeModulator = comm.QPSKDemodulator('BitOutput',true,...
    'DecisionMethod','Log-likelihood ratio',...
    'PhaseOffset',0,...
    'VarianceSource', 'Input port');
hBitError = comm.ErrorRate;
hQuantizer = dsp.ScalarQuantizerEncoder(...
    'Partitioning', 'Unbounded',...
    'BoundaryPoints', QuantizerBoundaries,...
    'InputFormat', 'Soft',...
    'OutputDataType', 'double',...
    'TerminationMethod', 'Terminated');
end
```

= comm.QPSKDemodulator('DecisionMethod', 'Log-Likelihood ratio')

= comm.ViterbiDecoder... 'InputFormat', 'Soft',...

![](image)
MATLAB Demo

Modeling and Simulation of a transceiver system, showcasing Coding Gain:
- Convolutional Encoding & Viterbi decoding
- Hard and Soft Decision decoding
- Turbo Encoding & Decoding

```
function [ber_bits]=fcn_04PSK_zTurbo_Soft(EbNo,MaxNumErrs,MaxNumBits)

%% Initialization
persistent hModulator hAWGN hDeModulator hBitError
persistent hTurboEncoder hTurboDecoder
if isempty(hModulator)
    hModulator = comm.QPSKModulator('PhaseOffset',0,'BitInput',true);
    hAWGN = comm.AWGNChannel(...
        'NoiseMethod','Random',
        'Seed',0);
end

```

```matlab
% Turbo Encoder
hTurboEncoder = comm.TurboEncoder(...
    'TrellisStructure',Trellis,...
    'InterleaverIndices',Indices);
```

```matlab
% Turbo Decoder
hTurboDecoder = comm.TurboDecoder(...
    'DecisionMethod','Log-Likelihood ratio',
    'NumIterations', 6,...
```

Turbo Coding

BER vs. $E_b/N_0$ (dB)
DL-SCH transport channel processing

Transport block
- CRC attachment

Code block segmentation
- Code block CRC attachment

Channel coding
- $a_0, a_1, ..., a_{A-1}$

Rate matching
- $b_0, b_1, ..., b_{B-1}$

Code block concatenation
- $c_{r_0}, c_{r_1}, ..., c_{r(E-1)}$

Channel encoding
- $d_{r_0}^{(i)}, d_{r_1}^{(i)}, ..., d_{r(D-1)}^{(i)}$

Code block CRC attachment
- $e_{r_0}, e_{r_1}, ..., e_{r(E-1)}$

Transport block
- $f_0, f_1, ..., f_{G-1}$

3GPP TS 36.212 v10.0.0

>>commppccc
Python Function block

```python
# Channel coding the TB
if (prmLTEDLSC.H.C=1)  % single CB, no CB CRC used
    % Turbo encode
tEncCbData = step(RCBTEnc, in);

    % Rate matching, minus bit selection
    rmCbData = lteCbRateMatching(tEncCbData, prmLTEDLSC.Kplus);

    % unify code paths
    out = rmCbData;
else  % multiple CBs in TB
    for cbIdx = 1:prmLTEDLSC.C
        % Code-block segmentation
        cbData = in((1:prmLTEDLSC.Kplus-24)) + (cbIdx-1)*(prmLTEDLSC.Kplus-24));

        % Append checksum to each CB
crcCbData = step(hCRCGenerator, cbData);

        % Turbo encode each CB
tEncCbData = step(RCBTEnc, crcCbData);

        % Rate matching, minus bit selection
        rmCbData = lteCbRateMatching(tEncCbData, prmLTEDLSC.Kplus);

        % Code-block concatenation
        out((1:prmLTEDLSC.Kplus*3+12)) + (cbIdx-1)*(prmLTEDLSC.Kplus*3+12) = rmCbData;
    end
end
```
OFDM Overview

- Orthogonal Frequency Division Multiplexing
  - Multicarrier modulation scheme (FFT-based)

- Split available spectrum into uniform intervals called sub-carriers
  - Transmit data independently at each sub-carrier

- Most important feature
  - Robust against multi-path fading
  - Using low-complexity frequency-domain equalizers
OFDM & Multi-path Fading

- Multi-path propagation leads to frequency selective fading
- Frequency-domain equalization is less complex and perfectly matches OFDM
- We need to know channel response at each sub-carrier – pilots

\[
y(n) = \sum_{n=0}^{N} h_n x(n - d_n)
\]

\[
Y(\omega) = H(\omega)X(\omega)
\]

If \( G(\omega_k) \approx H^{-1}(\omega_k) \) \( G(\omega_k) Y(\omega_k) \approx X(\omega_k) \)
How Does LTE Implement OFDM?

- **Frequency Nmax=2048**
- **Resource block**
- **Resource element**
- **Resource grid**

**MathWorks**

- **Interpolate vertically (Frequency)**
- **Interpolate horizontally (Time)**
- **Pilots**

**How Does LTE Implement OFDM?**

- **Pilots**
  - Interpolate horizontally (Time)
  - Interpolate vertically (Frequency)

**Resource grid**

- **1 sub-frame = 1 ms**

**Time (msec)**

- **Slot 1** 0.5
- **Slot 2** 1.0
- **1.5**
- **2.0**
How to Implement LTE OFDM in MATLAB

Depending on Channel Bandwidth

\[
\begin{align*}
\text{switch} & \quad \text{prmLTEPDSCH.Nrb} \\
& \quad \text{Case 25, } N=512; \\
& \quad \text{Case 100, } N=2048;
\end{align*}
\]

Set Frequency-domain FFT size

\[
\text{Transmitter:} \quad \text{Place pilots in regular intervals}
\]

\[
= 6 \times \text{numRb} + \text{mod}(v+vsh, 6) + 1;
\]

Create OFDM signal

\[
X = \text{ifft}(\text{tmp}, N, 1);
\]

Receiver: Estimate channel by interpolating in time & frequency

\[
\begin{align*}
& = \text{mean}([h(\cdot, 1, 1, n) \ h(\cdot, 3, 1, n)]) \\
& = \text{mean}([h(\cdot, 2, 1, :) \ h(\cdot, 4, 1, :)])
\end{align*}
\]
MIMO Overview

- Multiple Input Multiple Output
- Using multiple transmit and receive antennas

\[ Y = H^*X + n \]
Where is MIMO being used?

- Several wireless standards
  - 802.11n: MIMO extensions of WiFi as of 2008
  - 802.16e: As of 2005 in WiMax Standard
  - 3G Cellular: 3GPP Release 6 specifies transmit diversity mode
  - 4G LTE

- Two main types of MIMO
  - Spatial multiplexing
  - Space-Time Block Coding (STBC)
Space-Time Block Codes (STBC)

- STBCs insert redundant data at transmitter
- Improves the BER performance
- Alamouti code (2 Tx, 2 Rx) is one of simplest examples of orthogonal STBCs
Spatial Multiplexing

- MIMO technique used in LTE standard

- Divide the data stream into independent sub-streams and use multiple transmit antennas

- MIMO is one of the main reasons for boost in data rates
  - More transmit antennas leads to higher capacity

- MIMO Receiver essentially solves this system of linear equations

\[ Y = HX + n \]
MIMO-OFDM overview

$Y = H \ast X + n$

What if 2 rows are linearly dependent?

$H = \begin{bmatrix}
h_1 & h_2 & h_3 & h_4 \\
h_1 & h_2 & h_3 & h_4 \\
h_{31} & h_{32} & h_{33} & h_{34} \\
h_{41} & h_{42} & h_{43} & h_{44}
\end{bmatrix}$

Dimension = 4; Rank = 3; $H$ = singular (not invertible)

$H = UDV^H$

Singular Value Decomposition

To avoid singularity:
1. Precode input with pre-selected $V$
2. Transmit over antennas based on Rank
Adaptive MIMO: Closed-loop Pre-coding and Layer Mapping

- Layer Mapping
- Precoding
- OFDM
- Channel Rank Estimation
- Precoder Matrix Estimation

Base station

- Rank Indicator
- Precoder Matrix Indicator

mobile
Adaptive MIMO in MATLAB

- In Receiver:
  - Detect \( V = \text{Rank of the H Matrix} \)
  - \( = \text{Number of layers} \)

- In Transmitter: (next frame)
  - Based on number of layers
  - Fill up transmit antennas with available rank

```
V = prmLTEPDSCH.numLayers;

switch V
  case 1
    out = complex(zeros(inLen1/2, v));
  case 2
    out(:,1) = in1;
  case 3
    out(:,2) = in2;
  case 4
    out = complex(zeros(inLen1/2, v));
    out(:,1:2) = reshape(in1, 2, inLen1/2).';
    out(:,3:4) = reshape(in2, 2, inLen2/2).';
end
```
**Cell-Specific Reference Signal Mapping**

- Null transmissions allow for separable channel estimation at Rx
- Use clustering or interpolation for RE channel estimation

4 CSR, 0 nulls /slot /RB  
=> 8 CSR/160 RE

4 CSR, 4 nulls /slot /RB  
=> 8 CSR/152 RE

Varies per antenna port:  
4 CSR, 8 nulls /slot /RB  
=> 8 CSR / 144 RE

2 CSR, 10 nulls /slot /RB  
=> 4 CSR / 144 RE

*Figure 6.10.1.2-1, 3GPP TS 36.211 v10.0.0*
Link Adaptation Overview

- Examples of link adaptations
  - Adaptive modulation
    - QPSK, 16QAM, 64QAM
  - Adaptive coding
    - Coding rates from (1/13) to (12/13)
  - Adaptive MIMO
    - 2x1, 2x2, ..., 4x2, ..., 4x4, 8x8
  - Adaptive bandwidth
    - Up to 100 MHz (LTE-A)
LTE Physical layer model in MATLAB

Turbo Channel Coding

MIMO

OFDMA

Adaptation of everything
Put it all together …
References

**Standard documents** – [3GPP link](#)

**Books**

**Selected papers**
Summary

- MATLAB is the ideal language for LTE modeling and simulation
- Communications System Toolbox extend breadth of MATLAB modeling tools
- You can accelerate simulation with a variety of options in MATLAB
  - Parallel computing, GPU processing, MATLAB to C
- Address implementation workflow gaps with
  - Automatic MATLAB to C/C++ and HDL code generation
  - Hardware-in-the-loop verification
Modeling a 4G LTE System in MATLAB

Part 2:
Simulation acceleration
Why simulation acceleration?

- From algorithm exploration to system design
  - Size and complexity of models increases
  - Time needed for a single simulation increases
  - Number of test cases increases
  - Test cases become larger

- Need to reduce simulation time during design

- Need to reduce time for large scale testing during prototyping and verification
MATLAB is quite fast

- Optimized and widely-used libraries
  - BLAS: Basic Linear Algebra Subroutines (multithreaded)
  - LAPACK: Linear Algebra Package

- JIT (Just In Time) Acceleration
  - On-the-fly multithreaded code generation for increased speed

- Built-in support for vector and matrix operations

- Parallel computing support to utilize additional cores
  - Parallel Computing Toolbox
  - MATLAB Distributed Computing Server
  - GPU support
Simulation acceleration options in MATLAB

- **System Objects**
- **Parallel Computing**
- **MATLAB to C**
- **GPU processing**

User’s Code:
```
for k=1:max
   x = fft(data);
   y = 20*log10;
```

>> Demo commacceleration
Parallel Simulation Runs

Task 1  Task 2  Task 3  Task 4

>> Demo
Summary

- `matlabpool` → available workers
- No modification of algorithm
- Use `parfor` loop instead of `for` loop
- Parallel computation or simulation leads to further acceleration
- More cores = more speed

---

Versions of the Transceiver | Elapsed Time (sec) | Acceleration Ratio
---|---|---
1. Baseline | 40.0462 | 1.0000
2. + preallocation & vectorization | 24.1367 | 1.6591
3. + using Viterbi & MIMO-Decoder System Objects | 19.0918 | 2.0976
4. + using All available System Objects | 14.0926 | 2.8417
5. + MATLAB to C code generation | 7.7013 | 5.1999
6. + Parallel simulation runs with `parfor` | 4.6079 | 8.6908
Simulation acceleration options in MATLAB

- System Objects
- Parallel Computing
- MATLAB to C
- GPU processing

User’s Code

```matlab
for k=1:max
    x = fft(data);
    y = 20*log10;
```
What is a Graphics Processing Unit (GPU)

- Originally for graphics acceleration, now also used for scientific calculations
- Massively parallel array of integer and floating point processors
  - Typically hundreds of processors per card
  - GPU cores complement CPU cores
- Dedicated high-speed memory
Why would you want to use a GPU?

- Speed up execution of computationally intensive simulations
- For example:
  - Performance: \( A \backslash b \) with Double Precision
Options for Targeting GPUs

1) Use GPU with MATLAB built-in functions

2) Execute MATLAB functions elementwise on the GPU

3) Create kernels from existing CUDA code and PTX files
% Push data from CPU to GPU memory
Agpu = gpuArray(A)

% Bring results from GPU memory back to CPU
B = gather(Bgpu)
GPU Processing with Communications System Toolbox

- Alternative implementation for many System objects take advantage of GPU processing
- Use Parallel Computing Toolbox to execute many communications algorithms directly on the GPU
- Easy-to-use syntax
- Dramatically accelerate simulations

### GPU System objects

- `comm.gpu.TurboDecoder`
- `comm.gpu.ViterbiDecoder`
- `comm.gpu.LDPCDecoder`
- `comm.gpu.PSKDemodulator`
- `comm.gpu.AWGNChannel`
Example: Turbo Coding

- Impressive coding gain
- High computational complexity
- Bit-error rate performance as a function of number of iterations

Example: Turbo Coding

```matlab
function [ber_bits]=Fcn4_04PSK_Turbo_Soft(EbNo,MaxNumErrs,MaxNumBits)
    # codegen
    % Initialization
    FRM=65536;
    Trellis=poly2trunc(FRM,[37 21]);
    % Turbo decoder
    = comm.TurboDecoder(…
        'NumIterations', numIter, …
```
Acceleration with GPU System objects

- Same numerical results

```matlab
% comm.gpu.TurboDecoder(...
% 'NumIterations', N,...
% = comm.gpu.AWGNChannel(...
```
Key Operations in Turbo Coding Function

**CPU**

% Turbo Encoder
hTEnc = comm.TurboEncoder('TrellisStructure', poly2trellis(4, [13 15], 13), ...
'InterleaverIndices', intrlvrIndices)

% AWG Noise
hAWGN = comm.AWGNChannel('NoiseMethod', 'Variance');

% BER measurement
hBER = comm.ErrorRate;

% Turbo Decoder
hTDec = comm.TurboDecoder(...
'TrellisStructure', poly2trellis(4, [13 15], 13), ...
'InterleaverIndices', intrlvrIndices, 'NumIterations', numIter);

ber = zeros(3,1); %initialize BER output

while ( ber(1) < MaxNumErrs & & ber(2) < MaxNumBits)
    data = randn(blkLength, 1)>0.5;
    % Encode random data bits
    yEnc = step(hTEnc, data);
    %Modulate, Add noise to real bipolar data
    modout = 1-2*yEnc;
    rData = step(hAWGN, modout);
    % Convert to log-likelihood ratios for decoding
    llrData = (-2/noiseVar).*rData;
    % Turbo Decode
    decData = step(hTDec, llrData);
    % Calculate errors
    ber = step(hBER, data, decData);
end

**GPU Version 1**

% Turbo Encoder
hTEnc = comm.TurboEncoder('TrellisStructure', poly2trellis(4, [13 15], 13), ...
'InterleaverIndices', intrlvrIndices)

% AWG Noise
hAWGN = comm.AWGNChannel('NoiseMethod', 'Variance');

% BER measurement
hBER = comm.ErrorRate;

% Turbo Decoder
hTDec = comm.gpu.TurboDecoder(...
'TrellisStructure', poly2trellis(4, [13 15], 13), ...
'InterleaverIndices', intrlvrIndices, 'NumIterations', numIter);

ber = zeros(3,1); %initialize BER output

while ( ber(1) < MaxNumErrs & & ber(2) < MaxNumBits)
    data = randn(blkLength, 1)>0.5;
    % Encode random data bits
    yEnc = step(hTEnc, data);
    %Modulate, Add noise to real bipolar data
    modout = 1-2*yEnc;
    rData = step(hAWGN, modout);
    % Convert to log-likelihood ratios for decoding
    llrData = (-2/noiseVar).*rData;
    % Turbo Decode
    decData = step(hTDec, llrData);
    % Calculate errors
    ber = step(hBER, data, decData);
end
Profile results in Turbo Coding Function

**CPU**

```matlab
% Turbo Encoder
<0.01 hTEnc = comm.TurboEncoder('TrellisStructure',poly2trellis(4, [13 15], 13),...
   'InterleaverIndices', intrvlrIndices)
% AWG Noise
<0.01 hAWGN = comm.AWGNChannel('NoiseMethod', 'Variance');
% BER measurement
<0.01 hBER = comm.ErrorRate;

% Turbo Decoder
<0.01 hTDec = comm.TurboDecoder(...
   'TrellisStructure',poly2trellis(4, [13 15], 13),...
   'InterleaverIndices', intrvlrIndices, 'NumIterations', numIter);
<0.01 ber = zeros(3,1); %initialize BER output
% Processing loop
while (ber(1) < MaxNumErrs & & ber(2) < MaxNumBits)
    0.30 data = randn(blkLength, 1)>0.5;
    % Encode random data bits
    2.33 yEnc = step(hTEnc, data);
    % Modulate, Add noise to real bipolar data
    0.05 modout = 1-2*yEnc;
    1.50 rData = step(hAWGN, modout);
    % Convert to log-likelihood ratios for decoding
    0.03 llrData = (-2/noiseVar).*rData;
    330.54 decData = step(hTDec, llrData);
    % Calculate errors
    0.17 ber = step(hBER, data, decData);
end
```

**GPU Version 1**

```matlab
% Turbo Encoder
<0.01 hTEnc = comm.TurboEncoder('TrellisStructure',poly2trellis(4, [13 15], 13),...
   'InterleaverIndices', intrvlrIndices)
% AWG Noise
<0.01 hAWGN = comm.AWGNChannel('NoiseMethod', 'Variance');
% BER measurement
<0.01 hBER = comm.ErrorRate;

% Turbo Decoder
0.02 hTDec = comm.gpu.TurboDecoder(...
   'TrellisStructure',poly2trellis(4, [13 15], 13),...
   'InterleaverIndices', intrvlrIndices, 'NumIterations', numIter);
<0.01 ber = zeros(3,1); %initialize BER output
% Processing loop
while (ber(1) < MaxNumErrs & & ber(2) < MaxNumBits)
    0.28 data = randn(blkLength, 1)>0.5;
    % Encode random data bits
    2.38 yEnc = step(hTEnc, data);
    % Modulate, Add noise to real bipolar data
    0.05 modout = 1-2*yEnc;
    1.45 rData = step(hAWGN, modout);
    % Convert to log-likelihood ratios for decoding
    0.04 llrData = (-2/noiseVar).*rData;
    98.18 decData = step(hTDec, llrData);
    % Calculate errors
    0.17 ber = step(hBER, data, decData);
end
```
Key Operations in Turbo Coding Function

### CPU

% Turbo Encoder
hTEnc = comm.TurboEncoder('TrellisStructure',poly2trellis(4, [13 15], 13),...
'InterleaverIndices', intrlvrIndices)

% AWG Noise
hAWGN = comm.AWGNChannel('NoiseMethod', 'Variance');

% BER measurement
hBER = comm.ErrorRate;

% Turbo decoder
hTDec = comm.TurboDecoder('TrellisStructure',poly2trellis(4, [13 15], 13),...
'InterleaverIndices', intrlvrIndices, 'NumIterations', numIter);

%% Processing loop
while (ber(1) < MaxNumErrs && ber(2) < MaxNumBits)
data = randn(blkLength, 1)>0.5;
% Encode random data bits
yEnc = step(hTEnc, data);
% Modulate, Add noise to real bipolar data
modout = 1-2*yEnc;
% Convert to log-likelihood ratios for decoding
llrData = (-2/noiseVar).*rData;
% Turbo Decode
decData = step(hTDec, llrData);
% Calculate errors
ber = step(hBER, data, decData);
end

### GPU Version 2

% Turbo Encoder
hTEnc = comm.TurboEncoder('TrellisStructure',poly2trellis(4, [13 15], 13),...
'InterleaverIndices', intrlvrIndices)

% AWG Noise
hAWGN = comm.gpu.AWGNChannel('NoiseMethod', 'Variance');

% BER measurement
hBER = comm.ErrorRate;

% Turbo decoder
hTDec = comm.gpu.TurboDecoder('TrellisStructure',poly2trellis(4, [13 15], 13),...
'InterleaverIndices', intrlvrIndices, 'NumIterations', numIter, ...
'NumFrames', numFrames);

%% Processing loop
while (ber(1) < MaxNumErrs && ber(2) < MaxNumBits)
data = randn(numFrames*blkLength, 1)>0.5;
% Encode random data bits
yEnc = gpuArray(multiframeStep(hTEnc, data, numFrames));
% Modulate, Add noise to real bipolar data
modout = 1-2*yEnc;
% Convert to log-likelihood ratios for decoding
llrData = (-2/noiseVar).*rData;
% Turbo Decode
decData = step(hTDec, llrData);
% Calculate errors
ber = step(hBER, data, gather(decData));
end
## Profile results in Turbo Coding Function

### CPU

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Time (s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0.01</td>
<td>hTEnc = comm.TurboEncoder('TrellisStructure', poly2trellis(4, [13 15], 13), 'InterleaverIndices', intrvlrIndices)</td>
<td></td>
<td>Turbo Encoder</td>
</tr>
<tr>
<td>&lt;0.01</td>
<td>hAWGN = comm.AWGNChannel('NoiseMethod', 'Variance');</td>
<td></td>
<td>AWG Noise</td>
</tr>
<tr>
<td>&lt;0.01</td>
<td>hBER = comm.ErrorRate;</td>
<td></td>
<td>BER measure</td>
</tr>
<tr>
<td>&lt;0.1</td>
<td>hTDec = comm.TurboDecoder('TrellisStructure', poly2trellis(4, [13 15], 13), 'InterleaverIndices', intrvlrIndices, 'NumIterations', numIter);</td>
<td></td>
<td>Turbo Decoder</td>
</tr>
<tr>
<td>while (ber(1) &lt; MaxNumErrs &amp;&amp; ber(2) &lt; MaxNumBits)</td>
<td></td>
<td>Processing Loop</td>
<td></td>
</tr>
<tr>
<td>0.30</td>
<td>data = randn(blkLength, 1)&gt;0.5;</td>
<td></td>
<td>Encode random data bits</td>
</tr>
<tr>
<td>2.33</td>
<td>yEnc = step(hTEnc, data);</td>
<td></td>
<td>Modulate, Add noise to real bipolar data</td>
</tr>
<tr>
<td>0.05</td>
<td>modout = 1-2*yEnc;</td>
<td></td>
<td>Convert to log-likelihood ratios for decoding</td>
</tr>
<tr>
<td>1.50</td>
<td>rData = step(hAWGN, modout);</td>
<td></td>
<td>Turbo Decode</td>
</tr>
<tr>
<td>0.03</td>
<td>IlrData = (-2/noiseVar).*rData;</td>
<td></td>
<td>Calculate errors</td>
</tr>
<tr>
<td>330.54</td>
<td>decData = step(hTDec, IlrData);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.17</td>
<td>ber = step(hBER, data, decData);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### GPU Version 2

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Time (s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0.01</td>
<td>hTEnc = comm.TurboEncoder('TrellisStructure', poly2trellis(4, [13 15], 13), 'InterleaverIndices', intrvlrIndices)</td>
<td></td>
<td>Turbo Encoder</td>
</tr>
<tr>
<td>&lt;0.01</td>
<td>hAWGN = comm.gpu.AWGNChannel('NoiseMethod', 'Variance');</td>
<td>0.03</td>
<td>AWG Noise</td>
</tr>
<tr>
<td>&lt;0.01</td>
<td>hBER = comm.ErrorRate;</td>
<td></td>
<td>BER measure</td>
</tr>
<tr>
<td>numFrames</td>
<td>30;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hTDec</td>
<td>= comm.gpu.TurboDecoder('TrellisStructure', poly2trellis(4, [13 15], 13), 'InterleaverIndices', intrvlrIndices, 'NumIterations', numIter, 'NumFrames', numFrames);</td>
<td>0.01</td>
<td>Turbo Decoder - setup for Multi-frame or Multi-user processing</td>
</tr>
<tr>
<td>while (ber(1) &lt; MaxNumErrs &amp;&amp; ber(2) &lt; MaxNumBits)</td>
<td></td>
<td>Processing Loop</td>
<td></td>
</tr>
<tr>
<td>0.22</td>
<td>data = randn(numFrames*blkLength, 1)&gt;0.5;</td>
<td></td>
<td>Encode random data bits</td>
</tr>
<tr>
<td>2.45</td>
<td>yEnc = gpuArray(multiframeStep(hTEnc, data, numFrames));</td>
<td></td>
<td>Modulate, Add noise to real bipolar data</td>
</tr>
<tr>
<td>0.02</td>
<td>modout = 1-2*yEnc;</td>
<td></td>
<td>Convert to log-likelihood ratios for decoding</td>
</tr>
<tr>
<td>0.31</td>
<td>rData = step(hAWGN, modout);</td>
<td></td>
<td>Turbo Decode</td>
</tr>
<tr>
<td>0.01</td>
<td>IlrData = (-2/noiseVar).*rData;</td>
<td></td>
<td>Calculate errors</td>
</tr>
<tr>
<td>20.89</td>
<td>decData = step(hTDec, IlrData);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.09</td>
<td>ber = step(hBER, data, gather(decData));</td>
<td></td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Things to note when targeting GPU

- Minimize data transfer between CPU and GPU.

- Using GPU only makes sense if data size is large.

- Some functions in MATLAB are optimized and can be faster than the GPU equivalent (e.g. FFT).

- Use arrayfun to explicitly specify elementwise operations.
## Acceleration Strategies Applied in MATLAB

<table>
<thead>
<tr>
<th>Option</th>
<th>Technology / Product</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Best Practices in Programming</strong></td>
<td></td>
</tr>
<tr>
<td>• Vectorization &amp; pre-allocation</td>
<td>MATLAB, Toolboxes, System Toolboxes</td>
</tr>
<tr>
<td>• Environment tools. (i.e. Profiler, Code Analyzer)</td>
<td></td>
</tr>
<tr>
<td><strong>2. Better Algorithms</strong></td>
<td></td>
</tr>
<tr>
<td>• Ideal environment for algorithm exploration</td>
<td>MATLAB, Toolboxes, System Toolboxes</td>
</tr>
<tr>
<td>• Rich set of functionality (e.g. System objects)</td>
<td></td>
</tr>
<tr>
<td><strong>3. More Processors or Cores</strong></td>
<td></td>
</tr>
<tr>
<td>• High level parallel constructs (e.g. <code>parfor</code>, <code>matlabpool</code>)</td>
<td>Parallel Computing Toolbox, MATLAB Distributed Computing Server</td>
</tr>
<tr>
<td>• Utilize cluster, clouds, and grids</td>
<td></td>
</tr>
<tr>
<td><strong>4. Refactoring the Implementation</strong></td>
<td></td>
</tr>
<tr>
<td>• Compiled code (MEX)</td>
<td>MATLAB, MATLAB Coder, Parallel Computing Toolbox</td>
</tr>
<tr>
<td>• GPUs, FPGA-in-the-Loop</td>
<td></td>
</tr>
</tbody>
</table>
Summary

- MATLAB is the ideal language for LTE modeling and simulation
- Communications System Toolbox extend breadth of MATLAB modeling tools
- You can accelerate simulation with a variety of options in MATLAB
  - Parallel computing, GPU processing, MATLAB to C
- Address implementation workflow gaps with
  - Automatic MATLAB to C/C++ and HDL code generation
  - Hardware-in-the-loop verification
Modeling a 4G LTE System in MATLAB

Part 3: Path to implementation (C and HDL)
LTE Downlink processing

(a) Transport channel processing for DL-SCH
(b) Overview of downlink physical channel processing
Why Engineers translate MATLAB to C today?

Integrate MATLAB algorithms w/ existing C environment using source code or static libraries

Prototype MATLAB algorithms on desktops as standalone executables

Accelerate user-written MATLAB algorithms

Implement C/C++ code on processors or hand-off to software engineers
Algorithm Design and Code Generation in MATLAB

With MATLAB Coder, design engineers can

• Maintain one design in MATLAB
• Design faster and get to C/C++ quickly
• Test more systematically and frequently
• Spend more time improving algorithms in MATLAB
MATLAB Language Support for Code Generation

- visualization
- Java
- sparse
- nested functions
- classes
- graphics
- variable-sized data
- struct
- global
- complex
- fixed-point
- persistent
- System objects
- numeric
- functions
- malloc
- arrays
**Supported MATLAB Language Features and Functions**

- Broad set of language features and functions/system objects supported for code generation.

<table>
<thead>
<tr>
<th>Matrices and Arrays</th>
<th>Data Types</th>
<th>Programming Constructs</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Matrix operations</td>
<td>• Complex numbers</td>
<td>• Arithmetic, relational, and logical operators</td>
<td>• MATLAB functions and sub-functions</td>
</tr>
<tr>
<td>• N-dimensional arrays</td>
<td>• Integer math</td>
<td>• Program control (if, for, while, switch)</td>
<td>• Variable length argument lists</td>
</tr>
<tr>
<td>• Subscripting</td>
<td>• Double/single-precision</td>
<td></td>
<td>• Function handles</td>
</tr>
<tr>
<td>• Frames</td>
<td>• Fixed-point arithmetic</td>
<td></td>
<td>Supported algorithms</td>
</tr>
<tr>
<td>• Persistent variables</td>
<td>• Characters</td>
<td></td>
<td>• &gt; 400 MATLAB operators and functions</td>
</tr>
<tr>
<td>• Global variables</td>
<td>• Structures</td>
<td></td>
<td>• &gt; 200 System objects for</td>
</tr>
<tr>
<td></td>
<td>• Numeric classes</td>
<td></td>
<td>• Signal processing</td>
</tr>
<tr>
<td></td>
<td>• Variable-sized data</td>
<td></td>
<td>• Communications</td>
</tr>
<tr>
<td></td>
<td>• System objects</td>
<td></td>
<td>• Computer vision</td>
</tr>
</tbody>
</table>
Path to implementation

- Bring it all to Simulink

- Elaborate your design
  - Model System-level inaccuracies
  - Compensate
  - Add fixed-point
  - Generate HDL
MATLAB to Hardware
Why do we use FPGAs?

- Customized interfaces to peripherals
- High-speed communication interfaces to other processors
- Finite state machines, digital logic, timing and memory control
- High speed, highly parallel DSP Algorithms

We are going to focus on this use case today.
Separate Views of DSP Implementation

System Designer

Algorithm Design
- Fixed-Point
- Timing and Control Logic
- Architecture Exploration
- Algorithms / IP

System Test Bench
- Environment Models
- Analog Models
- Digital Models
- Algorithms / IP

FPGA Requirements
- Hardware Specification
- Test Stimulus

FPGA Designer

RTL Design
- IP Interfaces
- Hardware Architecture

RTL Verification
- Behavioral Simulation
- Functional Simulation
- Static Timing Analysis
- Timing Simulation
- Back Annotation

Implement Design
- Synthesis
- Map
- Place & Route

Hardware
What we will learn today

MATLAB to Hardware

- Convert design to fixed-point
  - Use Fixed-point Tool
  - Use `NumericTypeScope` in MATLAB
  - Verify against floating-point design
- Serialize design
- Implementation using HDL Coder
  - Verify through software and/or hardware co-simulation
OFDM Transmitter

% IFFT processing
45  x = ifft(tmp, N, 1);
46  x = x.*sqrt(N/len);
47
% Add cyclic prefix per OFDM symbol per antenna port
48  % and serialize over the subframe (equal to 2 slots)
49
% For a subframe of data
50  y = complex(zeros(subframeLen, numLayers));
for j = 1:2 % Over the two slots
51  % First OFDM symbol
52  y((j-1)*slotLen+1:cpLen0), :) = x((N-cpLen0+1):N, :)
53  y((j-1)*slotLen+cpLen0+(1:N), :) = x(1:N, (j-1)*7+1,
54
% Next 6 OFDM symbols
55  for k = 1:6
56  y((j-1)*slotLen+cpLen0+k*N+(k-1)*cpLenR+(1:cpLenR), :) =
57
end
MATLAB to Hardware

ifft(x, 2048, 1)

- **Issue #1**
  - ‘x’ is 2048x4 matrix
  - MATLAB does 2048-pt FFT along first dimension
  - Output is also 2048x4
  - Cannot process samples this way in hardware!
  - Serialize design

- **Issue #2**
  - MATLAB does double-precision floating-point arithmetic
  - Floating point is expensive in hardware (power and area)
  - Convert to fixed-point
HDL Workflow

- **Floating Point Model**
  - Satisfies System Requirements
    - Executable Specification
  - MATLAB and/or Simulink Model

- **Model Elaboration**
  - Develop Hardware Friendly Architecture in Simulink
  - Convert to Fixed-Point
    - Determine Word Length
    - Determine Binary Point Location

- **Implement Design**
  - Generate HDL code using Simulink HDL Coder
  - Import Custom and Vendor IP

- **Verification**
  - Software co-simulation with HDL simulator
  - Hardware co-simulation
Divide and conquer

Save simulation data to use in development
MATLAB-based FFT

- Can be done
- Resources
  - 120 LUTs (1%)
  - 32 Slices (1%)
  - 0 DSP48s
- Need 2048-point FFT
- 32-point FFT chokes synthesis tool!

```matlab
18 - u = complex(bu_r, bu_i);
19
20 - y = fft4(N, u);
21
22 - end
23
24 - function x = fft4(N, u)
25 -
26 - nt = numerictype(u);
27 - fm = fimath(u);
28
29 - x = complex(fi(zeros(4,1), nt, fm));
30
31 - tu1 = complex(fi(zeros(2,1), nt, fm));
32 - tu2 = complex(fi(zeros(2,1), nt, fm));
33
34 - [tu1(1), tu1(2)] = bfly2(u(1), u(2));
35 - [tu2(1), tu2(2)] = bfly2(u(3), u(4));
36
37 - % typical butterfly structure in an FFT
38 - %
39 - % u0 -------------> tu0
40 - %
41 - %
42 - %
43 - %
44 - %
45 - % tu1 -------------> tu1
```
Re-implement using Simulink blocks
compare against original code
Convert to fixed-point
compare against original code

OFDM Transmitter
Step 2 - Fixed-point

Demo
What you just saw

- Simulink Fixed-Point to model fixed-point data types
  - Word lengths
  - Fraction lengths

- Fixed-Point Tool
  - monitoring signal min/max, overflow
  - optimization of data types
Use original testbench to optimize settings
Analyze BER to determine word length

- Anything beyond 8 bits is “good enough”
Recall

\[ \text{ifft}(x, 2048, 1) \]

- **Issue #1**
  - ‘x’ is 2048x4 matrix
  - MATLAB does 2048-pt FFT along first dimension
  - Output is also 2048x4
  - Cannot process samples this way in hardware!
  - Serialize design

- **Issue #2**
  - MATLAB does double-precision floating-point arithmetic
  - Floating point is expensive in hardware (power and area)
  - Convert to fixed-point
Serial & Fixed-point

“HDL ready”
Automatically Generate HDL Code
Simulink HDL Coder

HDL Workflow Advisor
1. Set Target
   1.1. Set Target Device and Synthesis Tool
   1.2. Set Target Interface
2. Prepare Model For HDL Code Generation
   2.1. Check Global Settings
   2.2. Check Algebraic Loops
   2.3. Check Block Compatibility
   2.4. Check Sample Times
3. HDL Code Generation
   3.1. Set Code Generation Options
   3.1.1. Set Basic Options
   3.1.2. Set Advanced Options
   3.1.3. Set Testbench Options
   3.2. Generate RTL Code and Testbench
4. FPGA Synthesis and Analysis
   4.1. Create Project
   4.2. Perform Synthesis and P/R
   4.2.1. Perform Logic Synthesis
   4.2.2. Perform Mapping
   4.2.3. Perform Place and Route
5. Download to Target
   5.1. Generate Programming File
   5.2. Program Target Device

5.2. Program Target Device
Analysis
Program target FPGA device

Result: Passed
Passed Program target FPGA device.

Synthesis Tool Log:
Release 13.2 - IMPACT 0.6.1
Copyright (c) 1984-2011 The MathWorks, Inc. All rights reserved.
Preference Table
Name
StartupClock
AutoSignature
KeepsSVP
ConcurrentMode
UseHighs
ConfigOnFailure
UseLevel
MessageLevel
autoUseTime
ByteSwap
Auto_Infer
ShowDisplayComments
INFO:IMPACT - Digilent Plugin: no JTAG device was found.
AutoDetecting cable. Please wait.
Connecting to cable (USB Port - USB21).
Checking cable driver:
Driver file USBdrv.sys found.
Driver version: sig-1027, dest-1027.
What You Just Saw - Workflow Advisor

Select ASIC, FPGA, Or FPGA Board Target

Prepare Model For HDL Code Generation

Generate HDL Code

Physical Design and Critical Path Highlighting

Program FPGA
What You Just Saw – Generated HDL Code

```vhdl
-- Component Configuration Statements
FOR ALL : filter_bank_left
  USE ENTITY work.filter_bank_left(rtl);

-- Signals
SIGNAL filter_bank_left_out1 : std_logic_vector(15 DOWNTO 0); -- ufix16
SIGNAL filter_bank_right_out1 : std_logic_vector(15 DOWNTO 0); -- ufix16

BEGIN
  -- <S6>/filter_bank_left
  u_filter_bank_left : filter_bank_left
    PORT MAP( clk => clk,
             reset => reset,
             enb => enb,
             input => LeftIn, -- sfix16_En15
             parameters => Parameters, -- uint8 [10]
             gain => Gain, -- sfix16_En11
             data_out => filter_bank_left_out1 -- sfix16_En8
    );

  -- <S6>/filter_bank_right
  u_filter_bank_right : filter_bank_left
    PORT MAP( clk => clk,
               reset => reset,
               enb => enb,
               input => RightIn, -- sfix16_En15
               parameters => Parameters, -- uint8 [10]
               gain => Gain, -- sfix16_En11
               data_out => filter_bank_right_out1 -- sfix16_En8
    );

  LeftOut <= filter_bank_left_out1;
  RightOut <= filter_bank_right_out1;
```

Readable, Portable HDL Code
LTE Frame Structure (FDD)

- 1 Frame = 10ms
- 10 sub-frames per frame
- 2 slots per sub-frame (1 slot = .5ms)
- 7 OFDM symbols per slot
  - 2048 subcarriers in our simulation
  - IFFT output sample time
    \[
    \frac{0.5ms}{\frac{7}{2048}} = 3.4877 \times 10^{-8} \text{s OR } 28.672MHz
    \]

(30.72MHz after cyclic prefix)
Frame and Slot Structure

30.72 MHz = 15000 * 2048

25 * cdma2000 = 8 * W-CDMA = 30.72 MHz

- If we do 2048-length FFT, we can have 15 of them per millisecond
- But we need a cyclic prefix

=> Choose to have 14 symbols/ms (1 sub-frame)
  - Split into two slots of 7 symbols
Review: Automatic HDL Code Generation

- Readable, portable HDL code
- Target ASIC and FPGA
- Standard Simulink libraries
- Push-button programming of Xilinx and Altera FPGA
- Optimize for area and speed
- Code traceability between model and code
FPGA Prototyping

- **Shorter iteration cycles**
  - Automatic HDL code generation
  - Integrated HDL verification

- **Flexible automatic HDL Code generation**
  - Speed Optimization
  - Area Optimization
  - Power saving options
  - Resource utilization
  - Validation models
HDL Coder
Generate VHDL and Verilog Code for FPGA and ASIC designs

MATLAB  Simulink

HDL Coder

Verilog and VHDL

New: MATLAB to HDL

- Automatic floating-point to fixed-point conversion
- HDL resource optimizations and reports
- Algorithm-to-HDL traceability
- Integration with simulation & synthesis tools
HDL Verifier
Verify VHDL and Verilog code using cosimulation and FPGAs

- Support for 15 Altera and Xilinx FPGA boards
- Use with:
  - HDL Coder
  - Hand-written HDL code

New: FPGA Hardware-in-the-Loop Verification
HDL Workflow

- **Floating Point Model**
  - Satisfies System Requirements
    - Executable Specification
  - MATLAB and/or Simulink Model

- **Model Elaboration**
  - Develop Hardware Friendly Architecture
  - Convert to Fixed-Point
    - Determine Word Length
    - Determine Binary Point Location

- **Implement Design**
  - Generate HDL code using HDL Coder
  - Import Custom and Vendor IP

- **Verification**
  - Software co-simulation with HDL simulator
  - Hardware co-simulation
Summary

- MATLAB is an ideal language for LTE modeling and simulation
- Communications System Toolbox extend breadth of MATLAB modeling tools
- You can accelerate simulation with a variety of options in MATLAB
  - Parallel computing, GPU processing, MATLAB to C
- Address implementation workflow gaps with
  - Automatic MATLAB to C/C++ and HDL code generation
  - Hardware-in-the-loop verification
Thank You

Q & A