simDM Modulation Simulation

Contribution to 802.3dm Task Force Ad Hoc

December 19, 2024

Ragnar Jonsson - Marvell

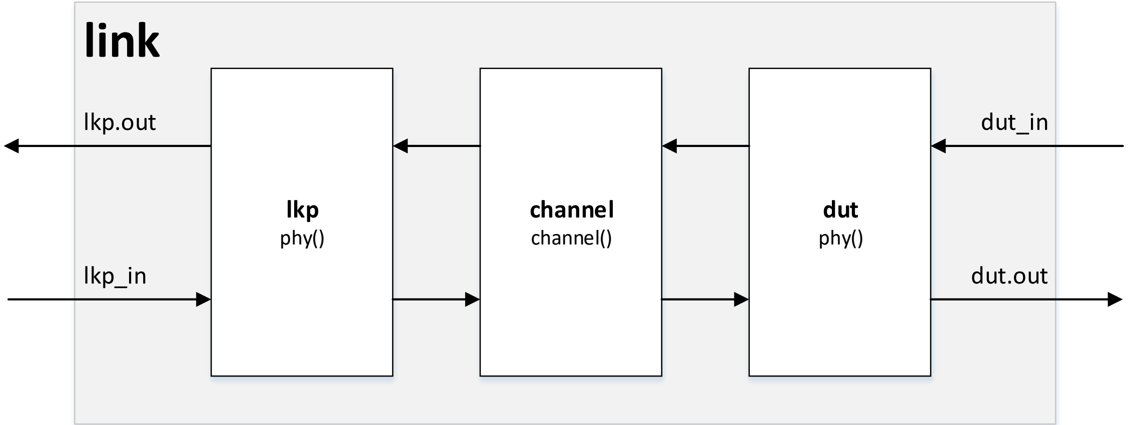
Introduction

- There are currently several modulation candidates being proposed for 802.3dm
- The following code is provided to facilitate simulation and more transparent evaluation of the individual proposal and more accurate comparisons of the proposals
- The following code is intended to be generic, but the code can then be configurated to represent either ACT or TDD
- Separate presentation uses this code to evaluate the performance of ACT in the presence of various impairments

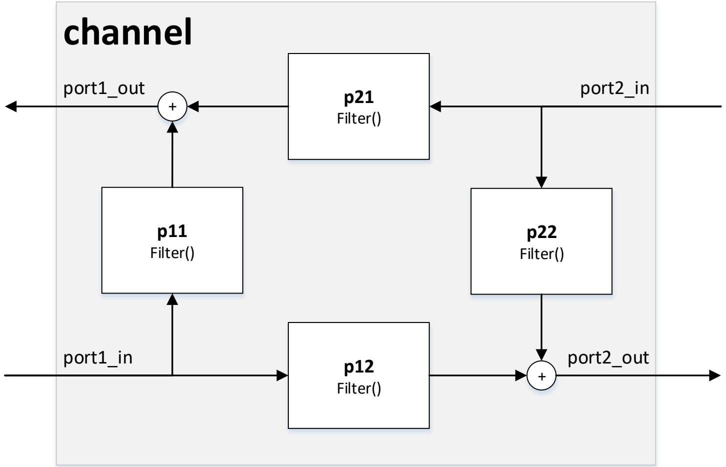
```
% This is simulation code provided to help with the development of
% IEEE 802.3dm.
%
% This code is provided for reference to allow independent evaluation
% of the accuracy and applicability of the simulation results shared in
% IEEE 802.3dm presentations by the author.
%
% Written by Ragnar Jonsson, affiliated with Marvell Technology, Inc.
% Version 1.0, December 16th, 2024
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
% OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
% FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
% THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
% LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
% FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
% DEALINGS IN THE SOFTWARE.
```

link – End-to-End link

```
function state = link(state,lkp_in,dut_in)
%  End-to-End link
    if(nargin<1)
         clear state;
         state.channel = channel();
         state.lkp = phy();
         state.dut = phy();
    else
         state.channel = channel(state.channel,state.lkp.tx_out,state.dut.tx_out);
         state.lkp = phy(state.lkp,state.channel.port1_out,lkp_in);
         state.dut = phy(state.dut,state.channel.port2_out,dut_in);
    end
end
```



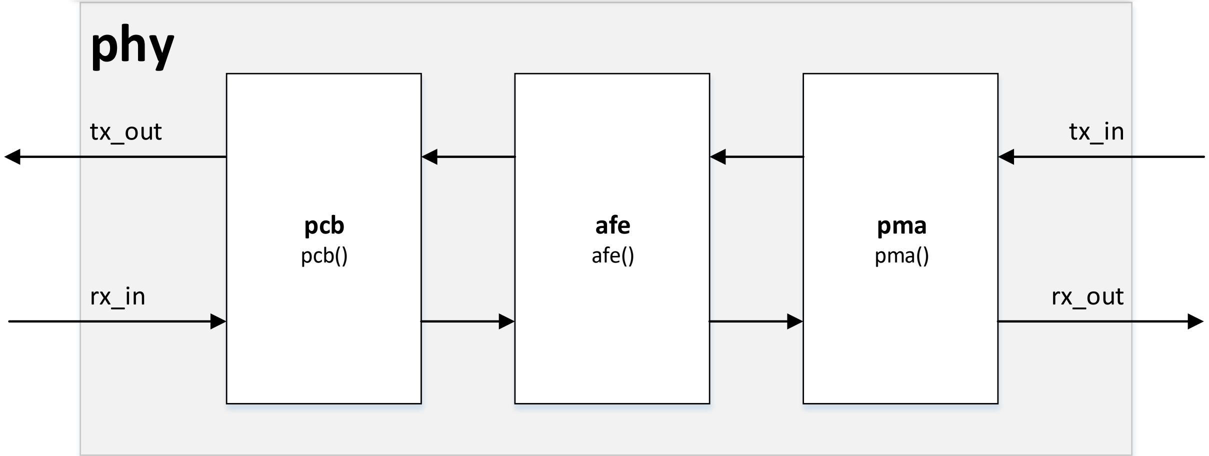
channel – Channel Model



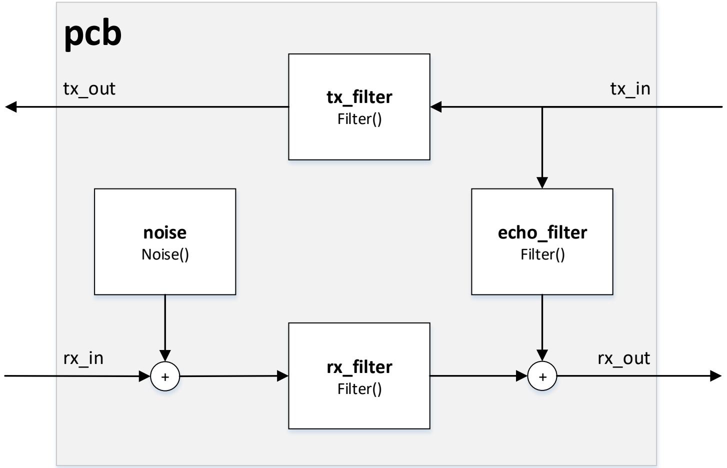

```
function state = channel(state,port1_in,port2_in)
%  Channel Model
    if((nargin<1)|(!isstruct(state)))
         clear state;
         state.p11 = Filter();
         state.p12 = Filter();
         state.p21 = Filter();
         state.p22 = Filter();
         state.port1_out = 0;
         state.port2_out = 0;
         state.p11.b = 0;
         state.p22.b = 0;
    else
         state.p11 = Filter(state.p11,port1_in);
         state.p12 = Filter(state.p12,port1_in);
         state.p21 = Filter(state.p21,port2_in);
         state.p22 = Filter(state.p22,port2_in);
         state.port1_out = Add(state.p11.out,state.p21.out);
         state.port2_out = Add(state.p12.out,state.p22.out);
    end
end
```

phy – PHY Top Level

```
function state = phy(state,rx_in,tx_in)
%  PHY top level
    if((nargin<1)|(!isstruct(state)))
         clear state;
         state.pma = pma();
         state.afe = afe();
         state.pcb = pcb();
         state.rx_out = 0;
         state.tx_out = 0;
    else
         state.pma = pma(state.pma,state.afe.rx_out,tx_in);
         state.afe = afe(state.afe,state.pcb.rx_out,state.pma.tx_out);
         state.pcb = pcb(state.pcb,rx_in,state.afe.tx_out);
         state.rx_out = state.pma.rx_out;
         state.tx_out = state.pcb.tx_out;
    end
end
```

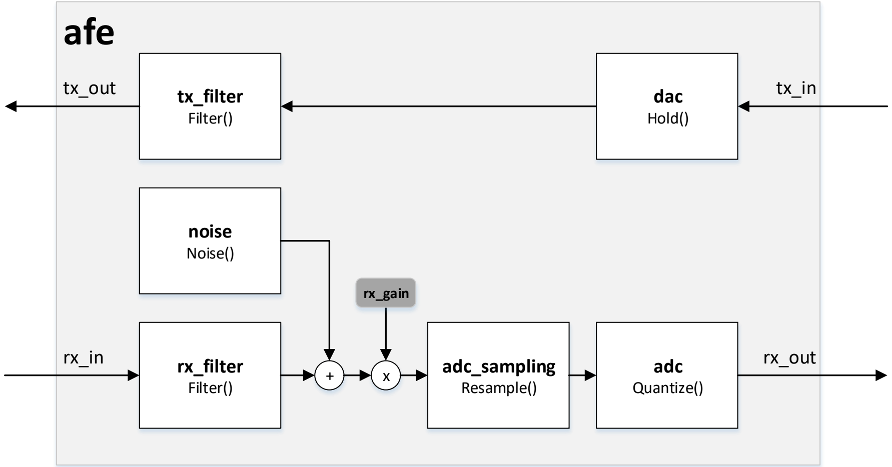


pcb - PCB and Packaging



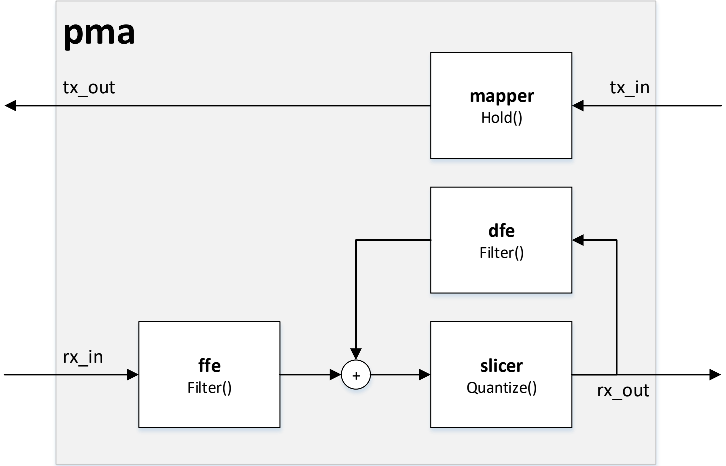

```
function state = pcb(state,rx_in,tx_in)
%  PCB and Packaging
    if((nargin<1)|(!isstruct(state)))
         clear state;
         state.noise = Noise();
         state.tx_filter = Filter();
         state.rx_filter = Filter();
         state.echo_filter = Filter();
         state.rx_out = 0;
         state.tx_out = 0;
         state.echo_filter.b = 0;
    else
         state.noise = Noise(state.noise,rx_in);
         signal_in = Add(rx_in,state.noise.out);
         state.tx_filter = Filter(state.tx_filter,tx_in);
         state.rx_filter = Filter(state.rx_filter,signal_in);
         state.echo_filter = Filter(state.echo_filter,tx_in);
         state.rx_out = Add(state.rx_filter.out,state.echo_filter.out);
         state.tx_out = state.tx_filter.out;
    end
end
```

afe – Analog Front End




```
function state = afe(state,rx_in,tx_in)
%  Analog Front End
    if(nargin<1)
        clear state;
        state.rx_gain = 1;
        state.noise = Noise();
        state.dac = Hold();
        state.tx_filter = Filter();
        state.rx_filter = Filter();
        state.adc_in = 0;
        state.adc_sampling = Resample();
        state.adc = Quantize();
        state.rx_out = 0;
        state.tx_out = 0;
    else
        state.dac = Hold(state.dac,tx_in);
        state.tx_filter = Filter(state.tx_filter,state.dac.out);
        state.rx_filter = Filter(state.rx_filter,rx_in);
        state.noise = Noise(state.noise,state.rx_filter.out);
        state.add_noise = Add(state.rx_filter.out,state.noise.out);
        state.adc_in = Mult(state.add_noise,state.rx_gain);
        state.adc_sampling = Resample(state.adc_sampling,state.adc_in);
        state.adc = Quantize(state.adc,state.adc_sampling.out);
        state.rx_out = state.adc.out;
        state.tx_out = state.tx_filter.out;
    end
end
```

pma – PMA Digital Processing




```
function state = pma(state,rx_in,tx_in)
%  PMA digital processing
    if(nargin<1)
        clear state;
        state.mapper = Hold();
        state.ffe = Filter();
        state.dfe = Filter();
        state.slicer = Quantize();
        state.ffe_oversampling = 1;
        state.rx_out = 0;
        state.tx_out = 0;
    else
        state.mapper = Hold(state.mapper,tx_in);
        state.ffe = Filter(state.ffe,rx_in);
        ffe_out = state.ffe.out(1:state.ffe_oversampling:end);
        N = length(ffe_out);
        state.rx_out = zeros(1,N);
        for n = 1:N,
            slicer_in = Add(ffe_out(n),state.dfe.out);
            state.slicer = Quantize(state.slicer,slicer_in);
            state.dfe = Filter(state.dfe,state.slicer.out);
            state.rx_out(n) = state.slicer.out;
        end
        state.tx_out = state.mapper.out;
    end
end
```

Support Functions

```
function out = Add(a,b)
  out = a + b;
end
------------------------------------------------------------------------
function state = Filter(state,in)
if((nargin<1)|(!isstruct(state)))
    clear state;
    state.a = 1;
    state.b = 1;
    state.state = [];
    state.out = 0;
  else
    [state.out,state.state] = filter(state.b,state.a,in,state.state);
  end
end
------------------------------------------------------------------------
function state = Hold(state,x)
if((nargin<1)|(!isstruct(state)))
    clear state;
    state.hold = 1;
    state.out = 0;
  else
    state.out = kron(x,state.hold);
  end
end
------------------------------------------------------------------------
function out = Mult(a,b)
  out = a * b;
end
------------------------------------------------------------------------
function state = Noise(state,x)
if((nargin<1)|(!isstruct(state)))
    clear state;
    state.eval = '0';
    state.out = 0;
  else
    eval(state.eval);
  end
end
```

```
function out = Qformat(format)
  if(nargin < 1)
    format = 0;
  end
  ------------------------------------------------------------------------
  switch(format(1))
    case 'q'
      bits = strsplit(strsplit(format,'q'){end},'.');
      i = eval(bits{1});
      b = eval(bits{2});
      st = 0.5^b;
      mx = 2^i - st;
      mn = -2^i;
      out = [mn st mx];
    case 'p'
      pam_levels = str2num(format(4:end));
      st = 2/(pam_levels-1);
      out = [-1 st 1];
    otherwise
      out = format;
    end
end
------------------------------------------------------------------------
function state = Quantize(state,x)
  if((nargin<1)|(!isstruct(state)))
    clear state;
    state.quant = 0;
    state.out = 0;
  elseif(state.quant == 0)
    state.out = x;
  else
    mn = state.quant(1);
    st = state.quant(2);
    mx = state.quant(3);
    r = roundb((x-mn)/st)*st+mn;
    state.out = max(mn,min(mx,r));
  end
end
```

simDM_ACT – Configuration of ACT Link Simulation

```
function act = simDM_ACT(hdr_rate,is_coax,pcb_cutoff)
% This is simulation code provided to help with the development of
% IEEE 802.3dm.
%
% This code is provided for reference to allow independent evaluation
% of the accuracy and applicability of the simulation results shared in
% IEEE 802.3dm presentations by the autor.
%
% Written by Ragnar Jonsson, affiliated with Marvell Technology, Inc.
% Version 0.2, December 10th, 2024
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
% OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
% FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
% THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
% LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
% FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
% DEALINGS IN THE SOFTWARE.

  if(nargin < 1)
    hdr_rate = 10;
  end
  if(nargin < 2)
    is_coax = 1;
  end
  if(nargin < 3)
    pcb_cutoff = 10;
  end

  %%% create the simulation data structure %%%
  act = link();

  %%% set over sampling ration for both high and low data rate (22.5GHz)%%%
  act.hdr_oversampling = 40/hdr_rate;
  act.ldr_oversampling = 192;

  %%% emulate LDR Tx and Rx analog filters as 4th order Butterworth filters %%%
  [b,a]=butter(4,4.5/act.ldr_oversampling);
  act.lkp.afe.tx_filter.a = a;
  act.lkp.afe.tx_filter.b = b;
  act.dut.afe.rx_filter.a = a;
  act.dut.afe.rx_filter.b = b;

  %%% emulate HDR Tx and Rx analog filters as 4th order Butterworth filters %%%
  [b,a]=butter(4,1/act.hdr_oversampling);
  act.lkp.afe.rx_filter.a = a;
  act.lkp.afe.rx_filter.b = b;
  act.dut.afe.tx_filter.a = a;
  act.dut.afe.tx_filter.b = b;

  %%% emulate PCB trace as simple RC circuit %%%
  b_pcb = [0.25 0.25];
  a_pcb = [1 -0.5];

  %%% emulate AC-cap as RC circuit %%%
  R = 100;
  C = 10e-9; % 10nF, per Table 149C-2
  f_s = 22.5e9;
  f_0 = 1/(C*R);
  b_cap = 2*f_s/(2*f_s + f_0)*[1 -1];
  a_cap = [1 -(2*f_s-f_0)/(2*f_s+f_0)];

  %%% emulate PoC HP characteristics as second order Butterworth filters %%%
  [b_poc,a_poc] = butter(2,pcb_cutoff*2/22.5e3,'high');

  %%% emulate the total PCB transfer function as product of component transfer %%%
  a = conv(a_pcb,conv(a_poc,a_cap));
  b = conv(b_pcb,conv(b_poc,b_cap));
  if(is_coax)
    tx_scale = 0.5;
  else
    tx_scale = 1;
  end
  act.lkp.pcb.tx_filter.a = a;
  act.lkp.pcb.tx_filter.b = b*tx_scale;
  act.lkp.pcb.rx_filter.a = a;
  act.lkp.pcb.rx_filter.b = b;
  act.dut.pcb = act.lkp.pcb;

  %%% map LDR bits to Manchester code %%%
  act.lkp.pma.mapper.hold = [1 -1];

  %%% emulate LDR ADC as Zero-order-Hold %%%
  act.lkp.afe.dac.hold = ones(1,act.ldr_oversampling/2)*0.5;

  %%% emulate HDR ADC as Zero-order-Hold %%%
  act.dut.afe.dac.hold = ones(1,act.hdr_oversampling)*0.5;

end
```

MARVELL™

Essential technology, done right™