How to account for non-LTI of Tx analog buffer in IBIS AMI flow

Vladimir Dmitriev-Zdorov

October 13, 2009
IBIS AMI group

Mentor Graphics®

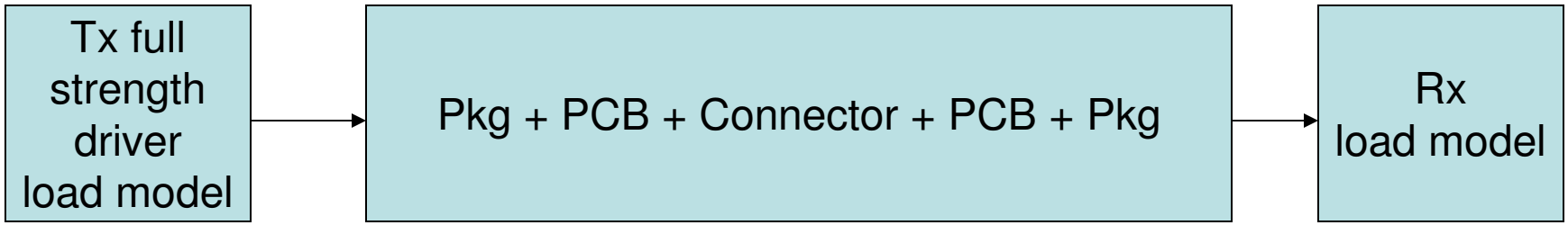
In this presentation we discuss the possibility to account for non-LTI behavior of the Tx output buffer in AMI flow:

- *Why this is important*
- *What should be added/changed in the current approach*
- *We consider the general issues, without final details. But if the concept is accepted, it does not seem difficult to specify changes in the flow and DLL's interface*

At the end, we briefly consider "true differential" version (optional)

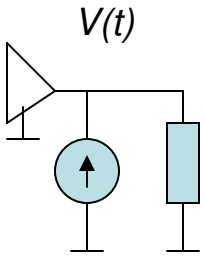
This is the current partitioning between DLLs and the 'channel'

Analog model includes:
Tx front end + channel + Rx front end
They all are assumed LTI.

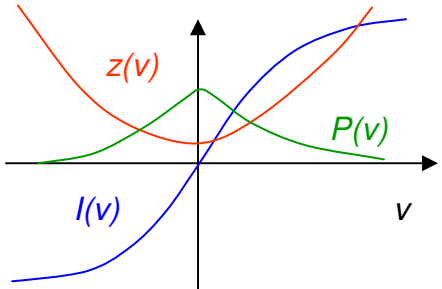


Is output buffer of Tx always close to linear?

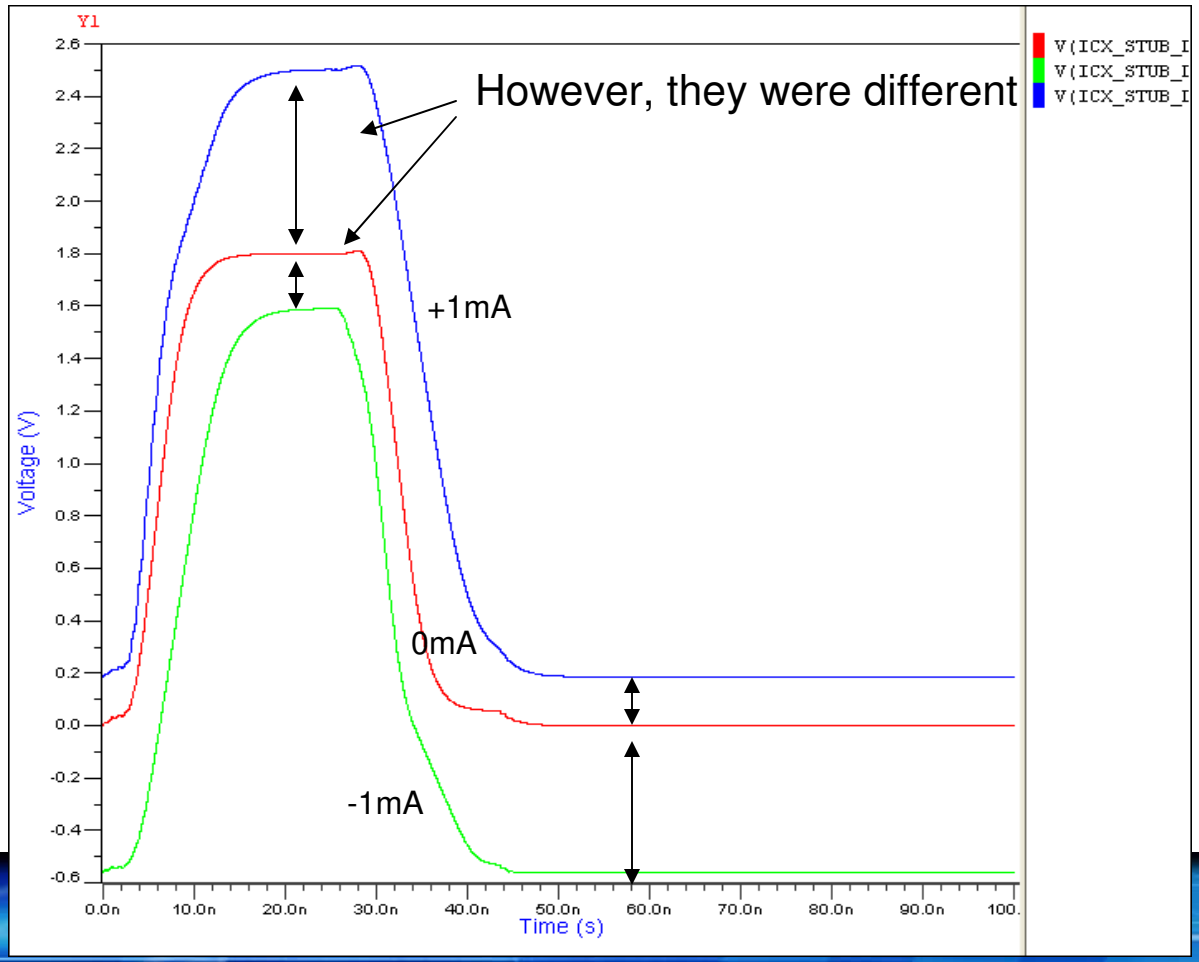
Linearity check:
Took a customer model at hand (IBIS)
Added plus/minus 1mA current source to ground, performed SPICE simulation.
In case of linear impedance, we should see identical offsets



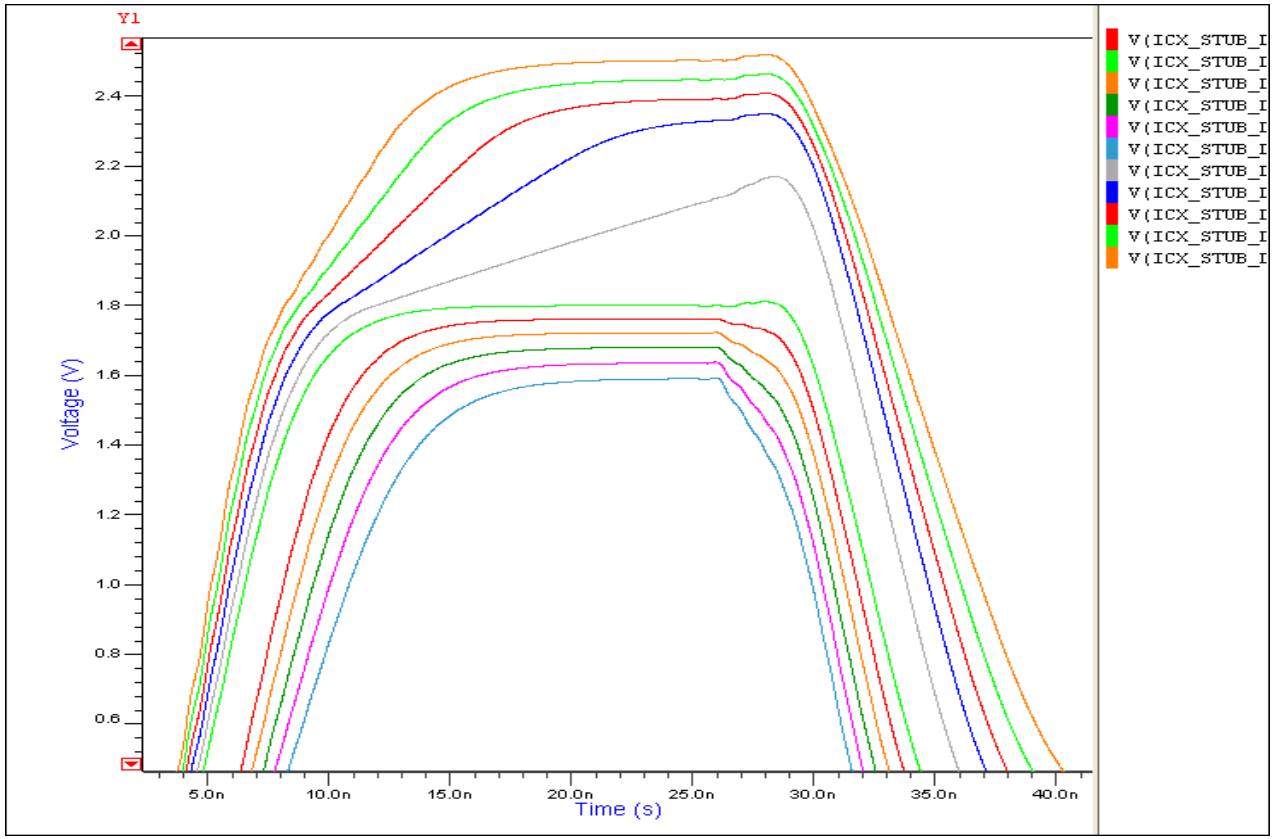
This check makes practical sense.
Current source with constant impedance in parallel could serve as a stamp for a linear channel at a given time step.
Different current values then show the accumulated effect of channel's state variables, that depend on prehistory (preceding bits).



At switching point the driver has its smallest impedance and hence has the ability to pump the energy faster (produce more power).
This effect is not non-idealness but a basic feature that allows the device switching fast but consuming little power at steady state



Another example: Tx waveforms with 200µA current step increments

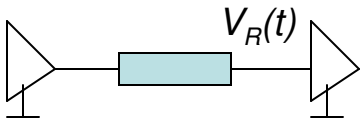


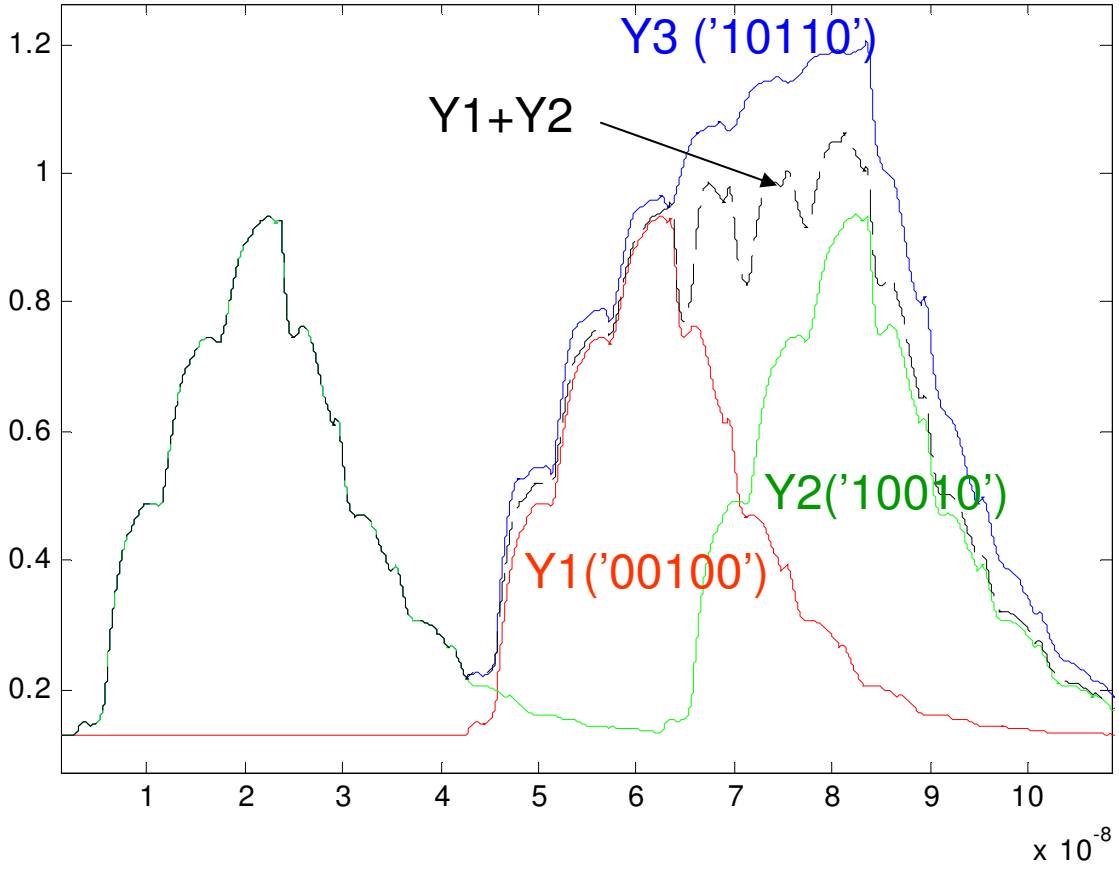
With LTI analog buffer, a constant current would only cause proportional vertical shifts. However, we observe changes in pulse width and shapes, too.

The above examples may seem extreme cases, but were not purposely selected: that was a typical cmos driver model. The effect from non-linearity can be smaller. But, in addition to that, there are time variations of the impedance (another side of non-LTI) that prevents from getting accurate results from superposition.

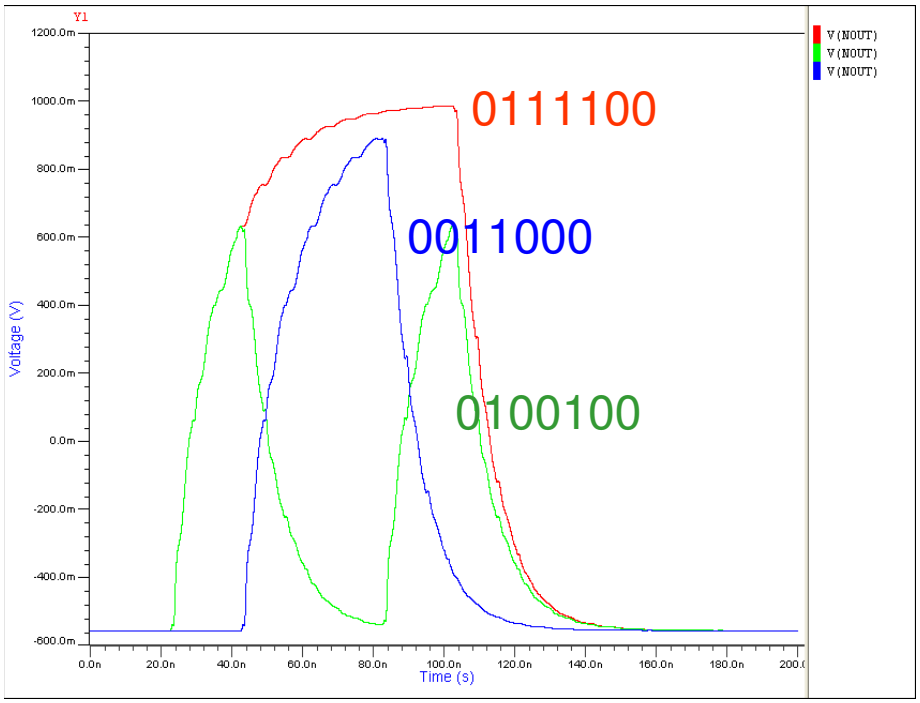
How big is the effect from non-LTI Tx at Rx input?

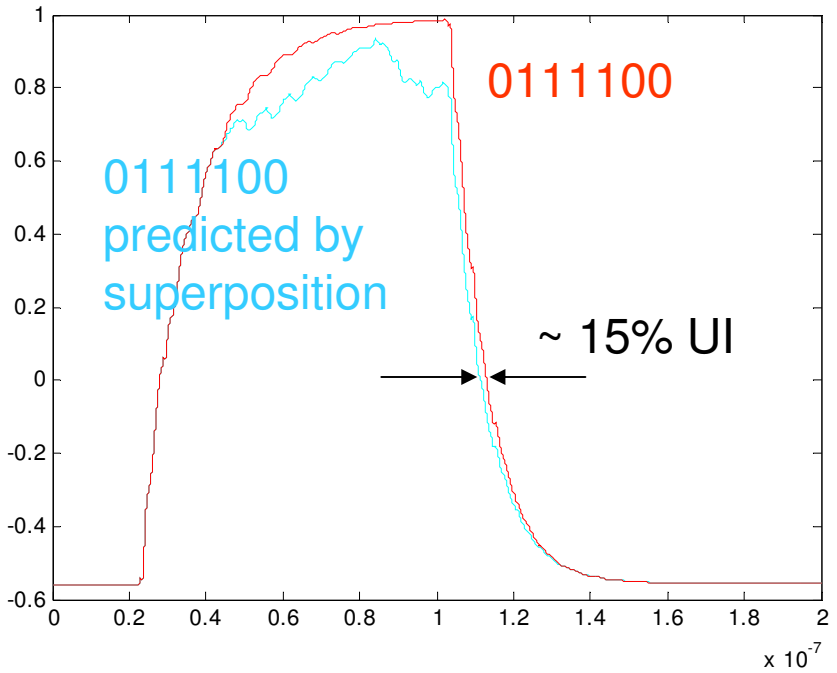
3 Spice simulations with inputs (x1, x2, x3) produced 3 outputs: y1, y2 and y3.
x1 = '00100'
x2 = '10010'
x3 = '10110'
As we see, x3=x1+x2. But is y3=y1+y2?
No, there is a substantial difference





The effect is typically smaller for differential buffers, but still considerable





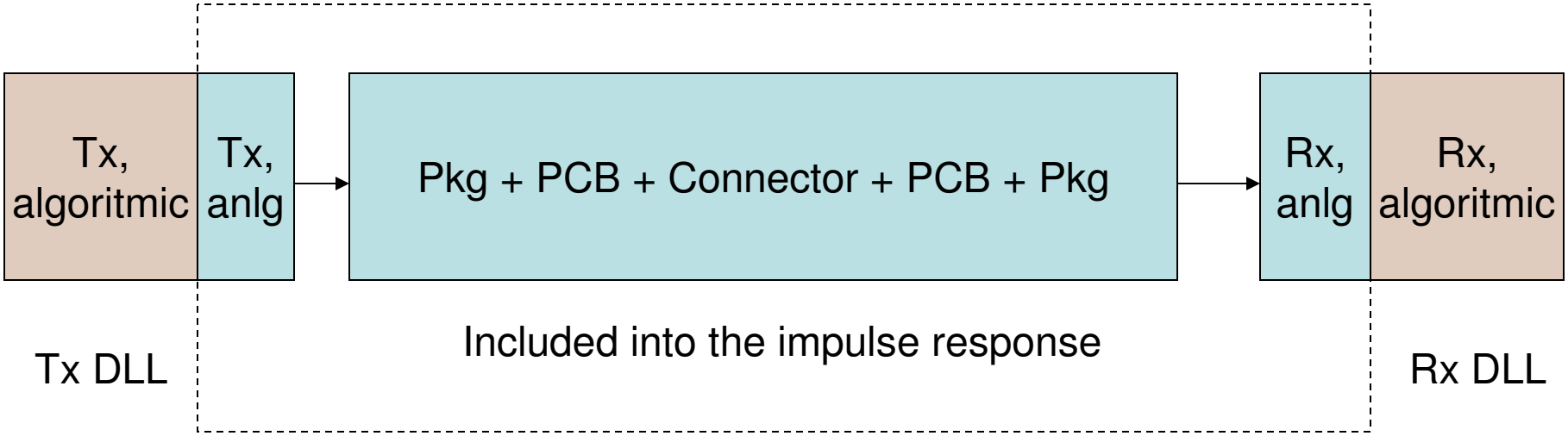
Some reports indicate that even though non-linearity of the Tx output buffer for every setting is not large, the output impedance is affected by tap settings in Tx equalization, that cannot be accounted for in the impulse response:

R. Mellitz, M Tsuk, T. Donisi, S. Pytel, Strategies for coping with non-linear and time variant behavior for high speed serial buffer modeling, DesignCon 2008.

How to handle that?

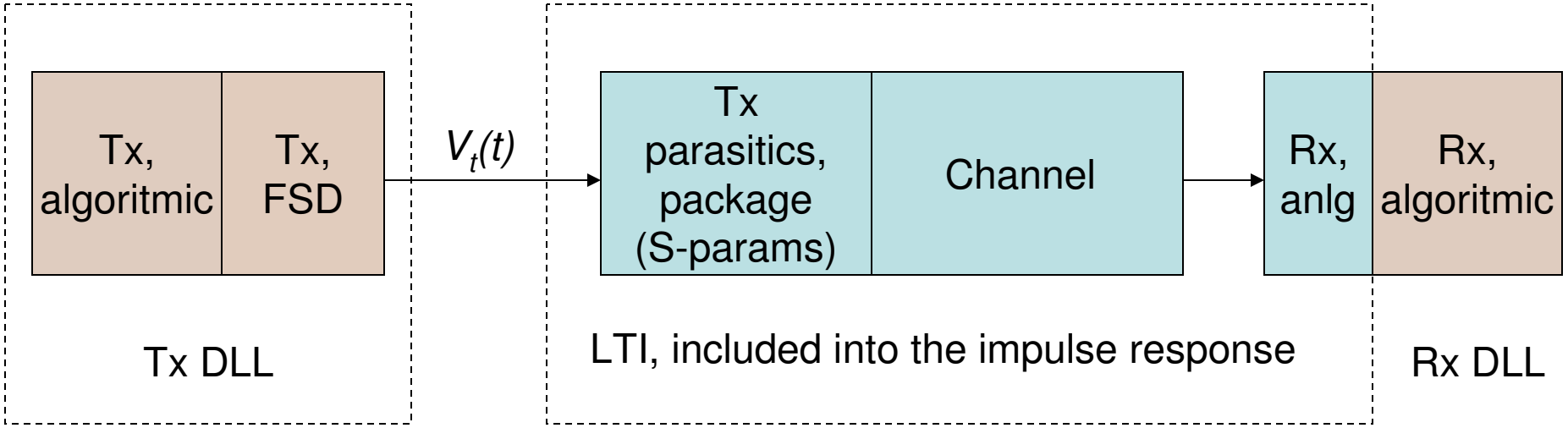
Current partitioning between EDA platform and DLLs

AMI standard assumes LTI of the front end buffers and the channel.
From here it follows that the analog part can be completely characterized by the impulse or step response having a meaning of the transfer function.



Partitioning needed to account for non-LTI Tx buffer

Can we implement non-linear full strength driver model and still use channel's impulse response?

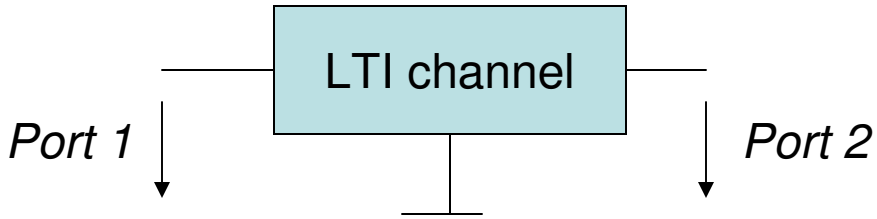


Yes, but then we need two, not one response functions to characterize the "channel".

Partitioning should be made as shown above

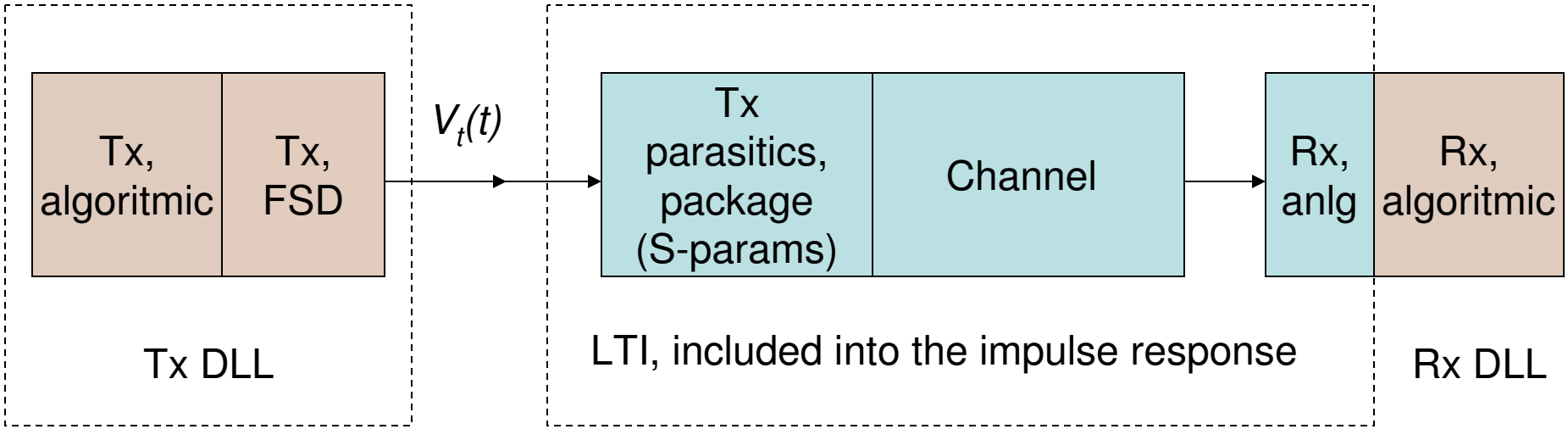
BTW, why do we need two responses, not one?

Any non-differential channel is a 2-port

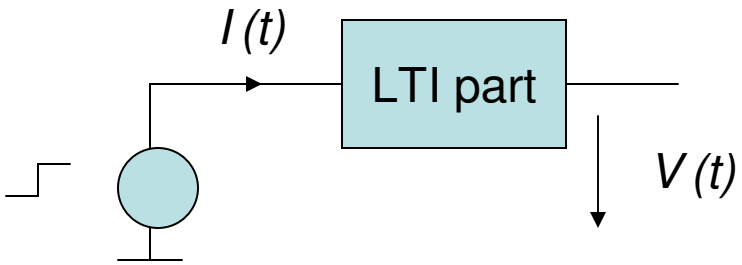


- As such, it generally requires 2x2 matrix of S, Y or Z parameters to describe, that assumes 4 different characteristics.
- Due to reciprocity (symmetry of the matrix) the number of unique functions reduces to 3.
- Finally, since the far end of the LTI model should not be connected to anything (Rx analog end is assumed LTI and included), we get rid of one more function.
- The remaining two independent functions could be of different type/dimension. Most convenient selections are input admittance and transfer function.
- Why we had just one transfer response? Because when Tx was assumed LTI, nothing was connected to the channel electrically at front end, and hence the # of functions was reduced to 1.

What does Tx DLL need to know about the channel to produce the correct voltage $V_{tx}(t)$?

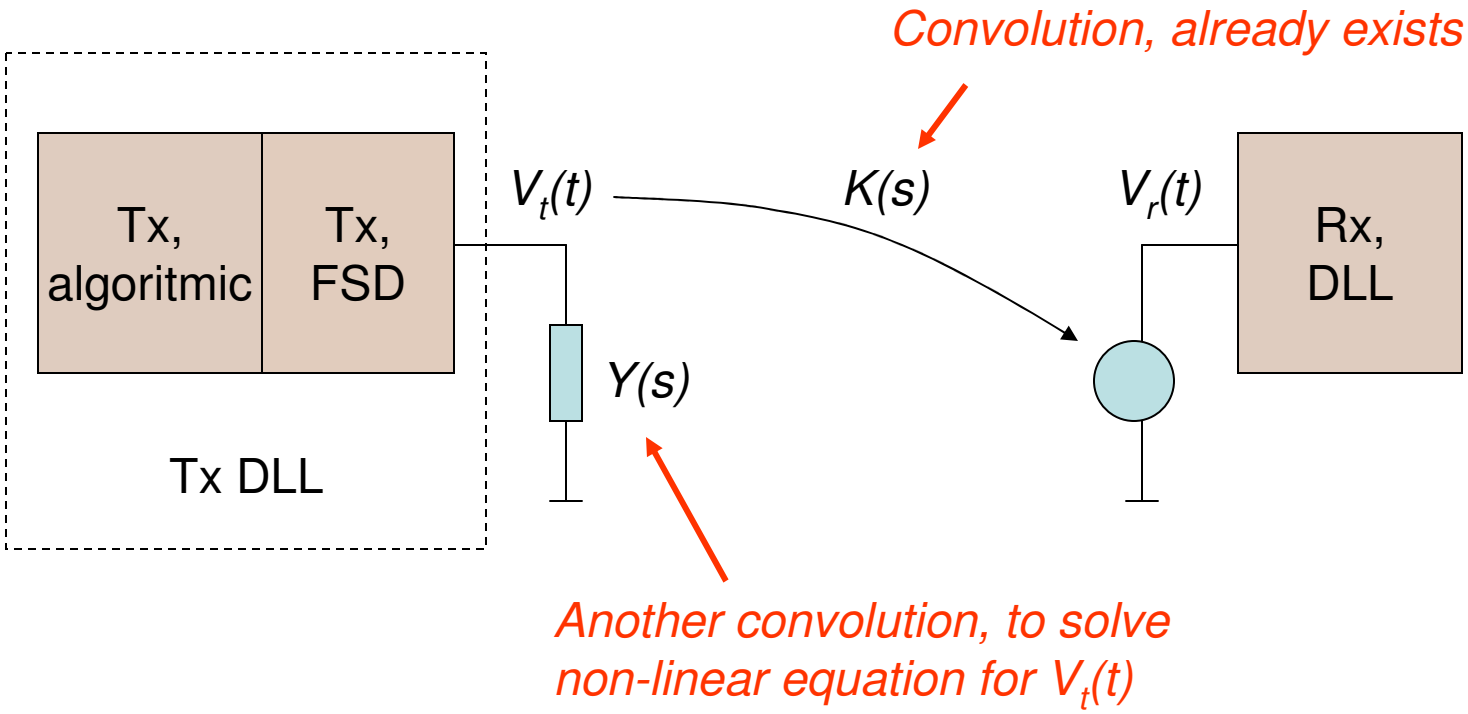


It only needs to know input impedance or admittance of the LTI part. Both responses are measured simultaneously:



By applying the step voltage, EDA platform measures not only the output voltage but the input current of the LTI part. From here, both transfer and admittance impulse responses are created

What should be the structure then (on GetWave stage)?



Convolution with *K(s)* is something we already have, convolution with *Y(s)* is new.
Is it a big complication? How to solve for the voltage $V_t(t)$?

How to solve for the voltage $V_t(t)$?

A simplest way is based on convolution. Current that goes into the conductance:

$$i = y(t) * v(t) = y_0 v_0 + \sum_{i=1}^{N} y_i v_i \quad (1) \quad \text{where} \quad y_i = y(t_0 - ih)$$

Here, t_0 is an observation moment, and v_0 is the only unknown voltage.
The second relation comes from e.g. instant IV curve of the Tx itself:

$$i = F(v_0, t_0) \quad (2)$$

Hence, at every step one has to solve the equation:

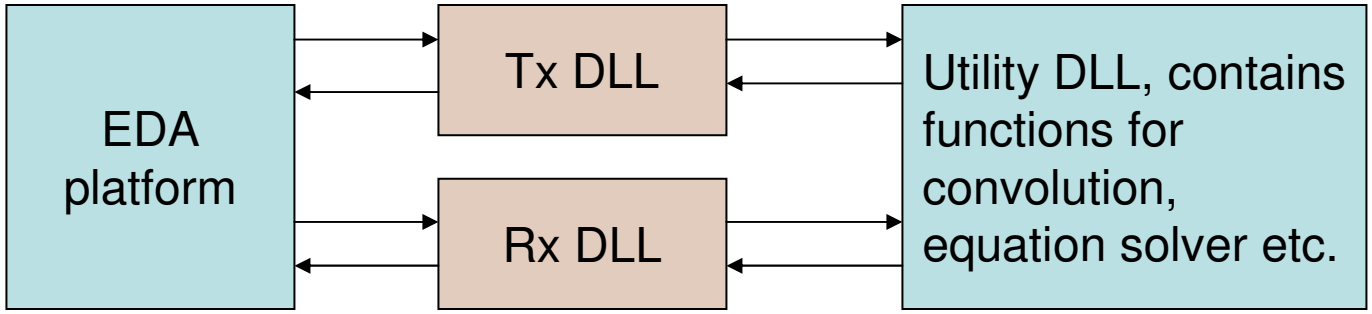
$$y_0 v_0 + \sum_{i=1}^{N} y_i v_i - F(v_0, t_0) = 0 \quad (3)$$

Solution should be governed by Tx DLL since it must produce the entire portion of the voltage waveform, not just one point. Tx DLL can solve equation (3) on its own. *However, EDA platform can take most of the burden to solve the equation*. It can even use more efficient 'recursive convolution' algorithm.

How? (Let see an optional "side" proposal)

Side proposal:
How EDA platform can help to simplify Tx/Rx DLL?

Can we imagine a shared standard utility DLL that contains the set of typical functions needed by Tx/Rx DLL?
The interface to Utility functions should be standardized, the content made available for all IC and EDA vendors.



For example, to solve equation (3), in Tx DLL they call the Utility function:
V = Util->Solve4TxVoltage(t0, fGetTxCurrent).
The second argument is a pointer to the function in Tx DLL that finds Tx current for a given voltage, as defined in (2).
The header info:

```
typedef double (*pGetTxCurrent) (double ); // use for Itx = F(Vtx), as in (2)
double Solve4TxVoltage(double t0, pGetTxCurrent);
void LoadAdmittanceResponse(double *resp, double step);
```

Therefore, the responsibility of Tx Dll is only to provide the function that performs (2).
Util instance will take care of the admittance response and store preceding values of the voltage, to compute the rest in equation (3).

What about “proof of concept” and performance?

We in Mentor have a software that has done very similar things for years. This is a superfast simulator that assumes nonlinearity only in the transmitter.

This simulator implements same type of partitioning (Transmitter is a separate DLL) and interface function (I=F(V)), we see no issues with accuracy and convergence.

2-3 calls per step to the function in Tx DLL that estimates transmitter’s current does not make a difference.

It is an extra convolution itself that affects the performance more. But not more than other types of convolution that we use with AMI models.

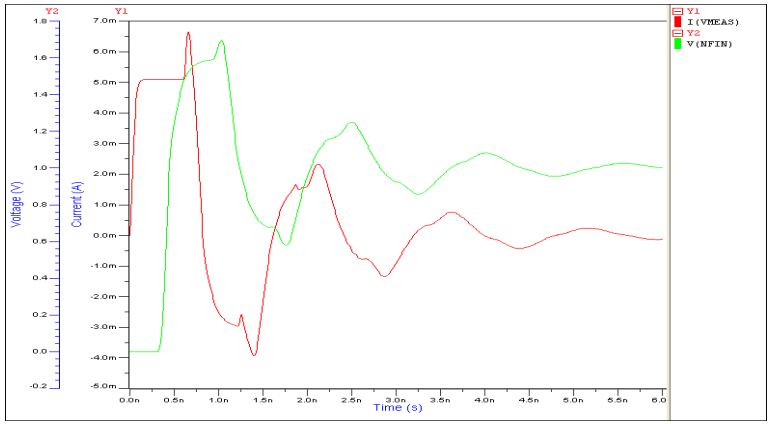
What are complications with input admittance?

-Tx Dll should have an additional functionality: it must be able to estimate the current for a given output voltage. It should either solve the equation on its own, or call the Utility DLL.
- Tx Dll Init() should optionally have one more parameter to handle admittance response. It will store and then use it or communicate with Utility DLL.
- Partition should be done differently when measuring impulse responses.
- EDA platform should measure two responses, not one.
- It is doubtful that Tx equalization could be combined with channel's transfer function to make a single convolution, because non-linear Tx output buffer is in between. Hence, Tx equalization should be made a separate convolution in Tx GetWave(). Utility DLL can provide a convolution function to help coding the Tx model.

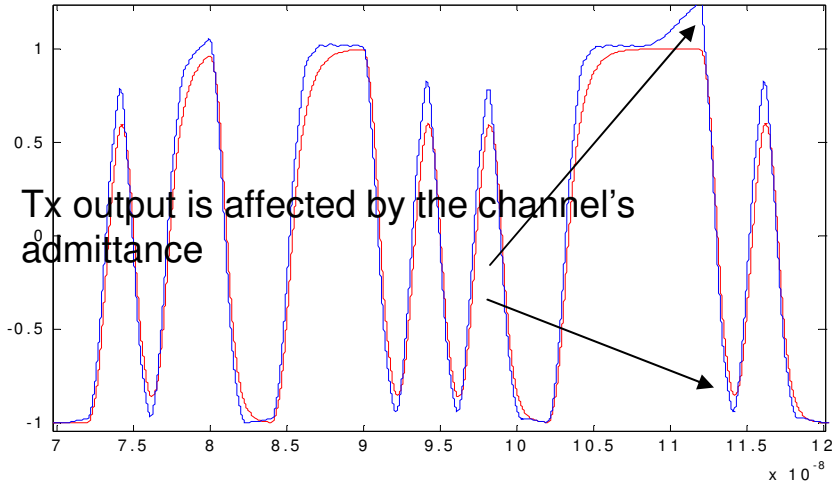
Implementation example

```
double cIbisBuffer::GetOutputVoltage(double time)
{
	/*
	form the sum of Yi * Vi from past voltages stored in CB,
	take admittance response samples in reverse order
	*/
	long i;
	double Isum = 0.0;
	for(i=0; i<lAdmitResponseLength-1; i++)
	{
		xCirBuf->Shift();
		Isum += xCirBuf->Read() *
			dAdmitResponse[lAdmitResponseLength-1-i];
	}

	double V0 = xCirBuf->Read(); // last value, use as initial guess
	xCirBuf->Shift();


	// Now, solve non-linear equation
	double v, dv, F, dF;
	double eps = 1e-6; // small increment;
	double y0 = dAdmitResponse[0]; // first sample, a factor at
unknown voltage
	bool bConverged = false;
	long maxiter = 50;
	v = V0;
	for(i=0; i<maxiter; i++)
	{
		F = GetCurrent(time, v) + y0 * v + Isum;
		dF = (GetCurrent(time, v+eps)+y0*(v+eps)+Isum-F) / eps;
			// derivative estim, optionally returned by fun
		if(fabs(dF)<1e-10)
		{
			bConverged = false;
			break;
		}
		dv = F / dF;
		dv = 0.2 * atan(dv/0.2); // damping
		v -= dv;
		if(fabs(dv)<1e-8)
		{
			bConverged = true;
			break;
		}
	}

	v = bConverged ? v : V0;
	xCirBuf->Write(v);
	return v;
}
```

(solution of non-linear equation and convolution with admittance inside Tx DLL)



Admittance and transfer step responses



Tx output is affected by the channel's admittance

Does differential output make a difference to the proposal?

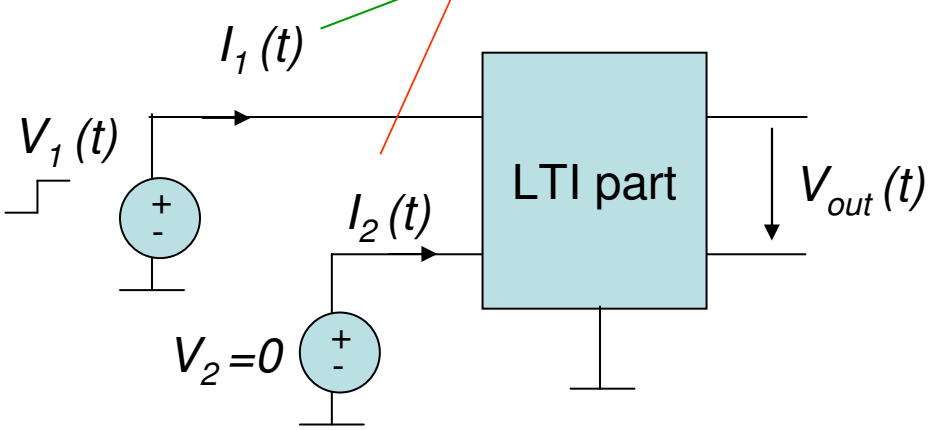
It does not make formal difference, however # of convolutions could be more. Let's consider 3 cases:

1. FSD is not sensitive to the common mode admittance of the channel. Then, the algorithm remains exactly the same. The currents and voltages we operate in Tx DLL become differential, so is the input admittance and the transfer function (have 2 convolutions total)
2. FSD is sensitive to the common mode admittance. Then, the admittance we measure for an LTI part should be an input admittance to the differential and common signal (i.e. two responses and therefore, two convolutions). The function in Tx DLL that estimates FSD's output current should provide two current values for a given time and two input voltages. We should solve a system of two equations, not one as before. Channel's transfer function still propagates only the difference signal, and therefore remains scalar. Hence, we need 3 convolutions total.
3. If we also want to consider mode conversions because of channel's asymmetry, then we need 3 convolutions for admittance and 2 for transfer function. This is well beyond current capabilities, we'll consider that separately (5 convolutions total).

Differential output without mode conversion

1. Measurement

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} \\ y_{12} & y_{11} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$



By applying the step voltage, EDA platform measures 3 responses that give the content of 2x2 admittance matrix and transfer response for the difference signal.

2. Convert admittance into MM (to allow 2 convolutions):

$$\begin{bmatrix} I_d \\ I_c \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(y_{11} - y_{12}) & 0 \\ 0 & 2(y_{11} + y_{12}) \end{bmatrix} \begin{bmatrix} V_d \\ V_c \end{bmatrix} = \begin{bmatrix} y_{dd} & 0 \\ 0 & y_{cc} \end{bmatrix} \begin{bmatrix} V_d \\ V_c \end{bmatrix}$$

Differential output without mode conversion

3. Only MM voltages are kept in circular buffer. Convolution sum (1) becomes:

$$\begin{bmatrix} I_{d0} \\ I_{c0} \end{bmatrix} = \begin{bmatrix} y_{dd0} & 0 \\ 0 & y_{cc0} \end{bmatrix} \begin{bmatrix} V_{d0} \\ V_{c0} \end{bmatrix} + \begin{bmatrix} I_{sumD} \\ I_{sumC} \end{bmatrix} \qquad (4)$$

4. Equations are formed with (4) converted into STD mode:

$$\begin{bmatrix} y_{11\_0} & y_{12\_0} \\ y_{12\_0} & y_{11\_0} \end{bmatrix} \begin{bmatrix} V_{10} \\ V_{20} \end{bmatrix} + \begin{bmatrix} 1 & 0.5 \\ -1 & 0.5 \end{bmatrix} \begin{bmatrix} I_{sumD} \\ I_{sumC} \end{bmatrix} - \begin{bmatrix} F_1(V_{10}, V_{20}, t) \\ F_2(V_{10}, V_{20}, t) \end{bmatrix} = 0 \qquad (5)$$

Equation (5) is solved for V_{10}, V_{20}.

5. Result is converted into MM and stored in circular buffer:

$$\begin{bmatrix} V_{d0} \\ V_{c0} \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} V_{10} \\ V_{20} \end{bmatrix} \qquad (6)$$

6. Differential voltage V_{d0} also participates in channel transfer response convolution

responses/convolutions in different modes

Case #	Analog channel type	# of analog responses to measure	# of analog convolutions to perform	# of n/l equations to solve
1	No admittance considered, only difference signal (current approach)	1 (transfer)	1	0
2	Single ended, consider admittance	2=1(trn)+1(adm)	2	1
3	Differential, not sensitive to common mode admittance	2=1(trn)+1(adm)	2	1
4	Differential, with common mode admittance but no conversion from C to D mode	3=1(trn)+2(adm)	3	2
5	Differential, most general case	5=2(trn)+3(adm)	6=2(trn)+4(adm)	2

Summary

- The proposal allows to greatly improve the accuracy of AMI-based approach, making it similar to Spice simulation
- The proposal does not incur considerable performance degradation compared to the existing standard
- The changes in Tx DLL are moderate (and remain optional). The use of Utility DLL is an option to help in development
- The “proof of concept and performance” exists by way of the well tested product plus Tx DLL example.

All above was all about supporting non-LTI behavior of the Tx's output buffer.

Below, we consider some potential future enhancements, to understand what it would take to implement them…

What if we want to consider common to differential mode conversion?

The next step when analyzing differential channels could be to account for common mode conversion, too. It is an important issue affecting BER and eye diagram. What extra would be needed then?

What extra is for Tx/Rx DLLs?

- *since Y-parameter matrix in (4) becomes full, 4 convolutions are required*
- *Tx DLL still solves for two unknown voltages and should estimate two currents as a response on two given voltages, as in (5), (6).*

Rx will not be affected, its input remains scalar.

Algorithmic parts in Tx/Rx DLL will not be affected, they still deal with differential responses only.

What extra must EDA platform do?

- *Compared to previous case, input admittance matrix becomes asymmetric, hence we need to measure 3 not 2 admittance responses. Plus, differential and common mode will have different transfer responses, hence 5 responses total.*

What if we want to consider non-linearity of Rx analog input, too?

This is a serious complication.
- Most importantly, we will not be able to run Tx/Rx DLLs independently on many bits, as we do now, because then we should solve the equations in Tx and Rx DLLs *simultaneously* at every point.
- Solution should be run on the point-by point basis, EDA platform should govern the solution for the analog part, and combine 2x2 channel stamp with momentary stamps from Tx and Rx for non-linear parts
- Convolution in algorithmic part of Tx/Rx is still possible but is less efficient because every time it will update just one call
- The channel should provide 3 responses for a single channel (non-differential) and 10 responses in differential case.
- The algorithm becomes very much as in general SPICE simulators, with addition of algorithmic processing in Tx/Rx DLLs.
- Performance is considerably affected, however it remains for 1-2 orders faster than in general simulators, because of avoiding much of the overhead costs.

Thanks