Experience of Behavioural Modelling in the Mixed-Signal ASIC Design Process - a Case Study

Peter R. Wilson (Member IEEE)[1], Paul Chapman[†], J. Neil Ross, Andrew D. Brown (Senior Member IEEE),

University of Southampton,
Department of Electronics and Computer Science,
Southampton, United Kingdom,
prw99r@ecs.soton.ac.uk

[†]Avant! Corporation,
Lower Farms Barn, Wasing Lane, Aldermaston,
Berkshire, United Kingdom, RG7 4NG
paul_chapman@avanticorp.com

Abstract

During the last 10 years, the use of computer simulation at IC level has become prevalent in the design process. As the size of designs has increased rapidly, the same simulations have become progressively more difficult to perform, to the extent that the simulations have become less of a design tool and more of a bureaucratic step to be carried out. Behavioural modelling offers a practical route to achieving productive simulation, by allowing what-if analyses, or component variations to improve the design prior to and concurrently with the progression of the design to Silicon implementation. In this paper, the positive use of behavioural modelling, multi-level modelling and mixed-signal simulation is highlighted, based on the authors' experience during a practical mixed-signal ASIC design project. The benefits and drawbacks of such an approach are discussed, with the key aspects identified that led to the project's successful completion.

Keywords-*Behavioural Modeling, MAST, Saber-Verilog, HSPICE, mixed-signal ASIC, Simulation*

1 Introduction

1.1 Design Overview

The ASIC (Application Specific Integrated Circuit) under investigation in this paper is a 10/100 Mbps Switch Controller used in Ethernet Networks [1]. The overall structure of the IC is shown in figure 1.

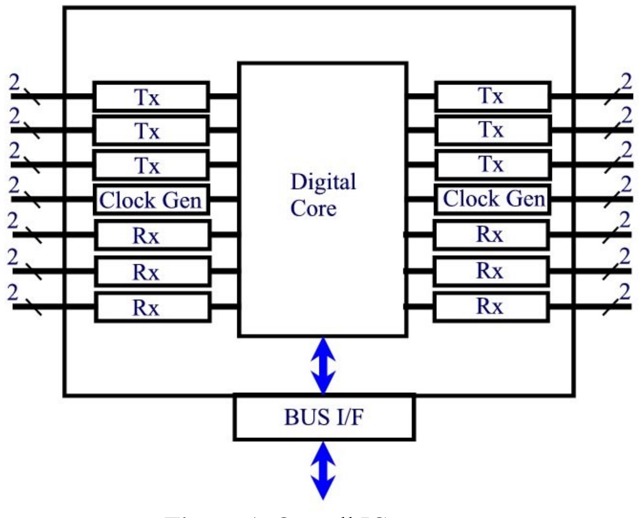

Figure 1: Overall IC structure

The IC consists of a digital core, carrying out all the DSP functions and processing of the network data, and a number of analogue transmit (Tx) and Receive (Rx) channels. In this variant of the IC, there were 6 Tx and Rx ports, with 2 clock generation ports. Each of the analogue interface ports has filters, automatic gain control (AGC), analogue-to-digital conversion (ADC) and digital-to-analogue conversion (DAC) linking the analogue port to the digital core.

Each channel operates using a Decision Feedback Equalizer (DFE), where automatic gain control is used to equalize the amplitude of the received pulse signal, the clock is then extracted by using a clock recovery PLL (Phase Locked Loop), and the data recovered with the correct phase using the properly extracted clock to synchronize the data. The data arrives in the form of MLT3 encoded data of the form shown in figure 2.

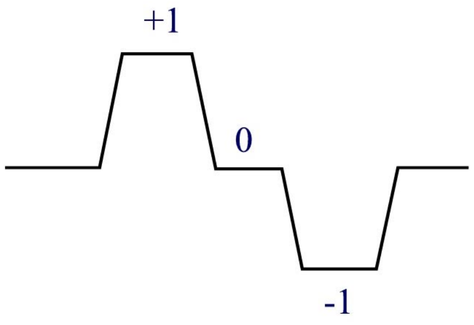

Figure 2: MLT-3 Encoded Waveform

Each channel uses a scaling band-pass filter (BPF) to provide filtering and a companding effect. This is followed by a digitally controlled AGC, where the control signal is 7 bits wide and is generated by the digital controller. The AGC is a current mode multiplying DAC, where the output of the BPF is a gm stage current output. The clock recovery circuit is shown in figure 3, where the oscillator is current controlled (ICO). The frequency and phase of the oscillator is tuned by track up and down pulses from the digital controller and a charge pump.

[1] This work was partly supported by a grant from the Engineering and Physical Sciences Research Council (United Kingdom) and Advanced Power Components (Rochester, United Kingdom).

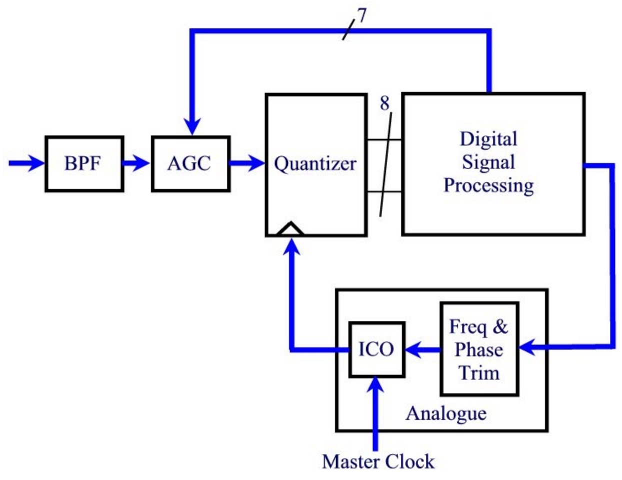

Figure 3: Receive Channel Clock Recovery

1.2 Initial Design Process

As is usual with a large and complex IC, a design team of experts in various aspects of the design process was assembled in the UK. The senior engineer was responsible for the overall design and algorithm, with specialists in analogue cell design and digital processing to implement their sections of the IC. To further complicate the process, the final IC layout and integration was to be carried out by a specialist team in the USA.

During the initial feasibility study, the high level algorithmic design was accomplished using Matlab, the digital processor design with Verilog and the IC simulations using Hspice. While the design of the individual elements proceeded quite well, problems started to occur when the pieces needed to be brought together in an integrated design. It soon became clear that linking the design elements together for the purposes of system checking and analysis using these disparate approaches was impossible practically. This led to a series of fundamental errors with connections, pin names, interface characteristics and behaviour, and repeated design changes without full checking of boundary data transfer. Obviously a new approach was needed.

1.3 Modified Design Process

The decision was taken to take the individual pieces of the design and integrate them in a form that could be designed and analysed properly. This step, which seems obvious to anyone from a behavioural modelling background, was not an easy step to take for the design team. Initial hostility, stemming from a lack of perceived benefit and seemingly extra work, proved an obstacle in producing new high-level models, but as soon as the potential of mixing multiple levels in a single design became apparent, this soon disappeared.

Using high level, Laplace models of the filters and interface circuitry, a full architectural model of the ASIC could be produced. The significant benefit of this step was to allow complete testing of the Verilog code in a representative environment, with real analogue data, for the first time in the design process. As a direct result of this, show-stopping errors in the Verilog code were identified and corrected. Another advantage of this type of simulation early in the design process was the ability to finalise the boundary connections of the device at a much earlier stage of the design, thus allowing the IC Level Designers to being work on the transistor level implementation much earlier than would have been possible otherwise. A summary of the process is shown in figure 4.

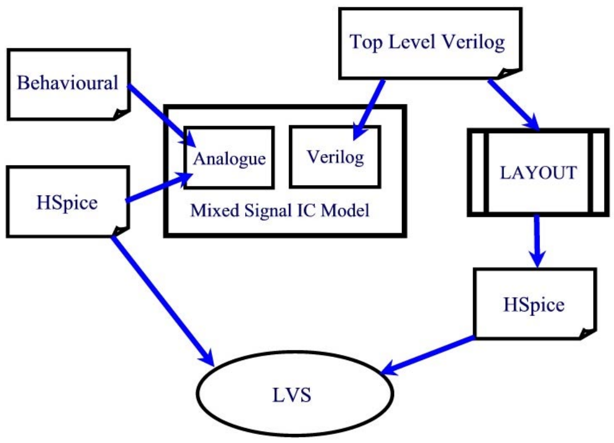

Figure 4: Modified IC Design Process Flow Diagram

Within the design process there were several key steps that needed to be taken to ensure that the design would have minimal errors from its inception. The fundamental starting point was to ensure that the top-level connection points and behaviour of the design was correct. To accomplish this, a behavioural model of the complete IC channel was required that included the Verilog digital model together with a complete behavioural model of the analogue and mixed-signal sections of the channel. This model was used to check that all the connections were correct, polarities of signals were correct and the basic digital core functionality was also correct. With this framework model in place, then each analogue and mixed-signal block could be designed down to the transistor level, and checked in the context of a complete IC simulation, together with the other behavioural blocks and the digital controller.

2 Channel Simulation

2.1 Introduction

Using the structure shown in figure 3, the channel was modelled using behavioural MAST [2] models for the analogue and mixed-signal models and the digital core was modelled using Verilog [3]. The problem at this stage was how to practically simulate these elements together, from the same Concept [4] schematics. The Saber [5] simulator was used in conjunction with the Verilog Simulator in a co-simulation. Using the schematic based approach meant that the same symbols (and hence connection points) were required for the transistor level (IC) schematic models thus implicitly ensuring that the connection points were correct. It was at this point that several connection errors were identified between the Verilog, Matlab and Hspice primitive netlists and corrected. It was also apparent at this stage that several errors in polarity would have not been spotted until much later in the design process unless this integrated approach has been taken.

This section describes the behavioural modelling of the key analogue and mixed signal blocks, and the development of test benches allowing measured and generated cable data to be applied to test the Verilog rigorously.

2.2 Channel Behavioural Modeling

The input filter to the channel consists of a non-linear trans-conductance amplifier stage plus a band-pass filter (100Mbps) or a low-pass filter (10Mbps). The characteristics of ideal and practical trans-conductance (Gm) stages are shown in figure 5.

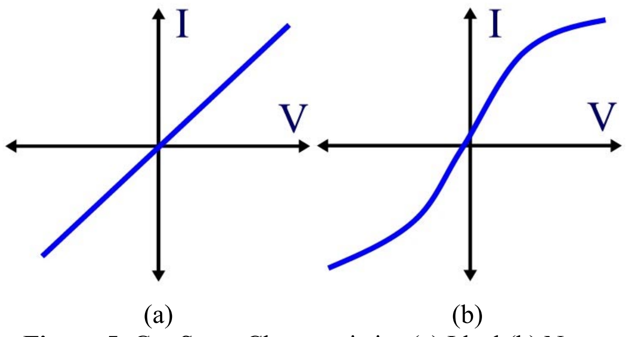

Figure 5: Gm Stage Characteristics (a) Ideal (b) Non-Linear

The basic transconductance function was implemented using the MAST model in figure 6. This was combined with the appropriate filter model for the 100Mbps or 10Mbps option as shown in figure 7.

```
template gm_filter inp inm outp outm = k
#...Declaration of connections
electrical inp,inm,outp,outm
#...Declaration of Parameters
number k = 1m  # gm Gain
{
#...Declare Internal Variables
number xk=0.5m
#...Parameters Section
parameters {
        xk = k/2.0
        }
vccs.1 inp inm outp outm=k=xk
}
```

Figure 6: Gm Stage MAST model

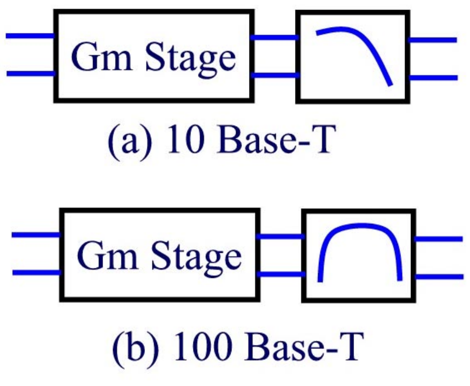

Figure 7: Gm Stage and Filter Model

These models are purely analogue, but with the introduction of a digitally controlled AGC, the mixed-signal aspects of the MAST modelling language become useful. The basic analogue behaviour of the AGC is a differential voltage to current converter, with the gain configured by a 7 bit digital word. The device converts this word to a scaling current, which is multiplied by the current corresponding to the input voltage (with a gain of 1). In nominal operation, the gain from the input to the DFE input (after the AGC) should be unity. There is also a reference dc bias current that can be added to the input. The resulting characteristic function of the AGC DAC is given in (1), with the behavioural model in figure 8.

$$I_{out} = I_{dc} + \frac{(D6:D0)}{127} * I_{in} \qquad (1)$$

```
template a7bdac vbp vbn vb d_6 d_5 d_4 d_3 d_2 \
 d_1 d_0 iop ion=gain, zin, tt
#...Declaration of Connections
state logic_4 d_0,d_1,d_2,d_3,d_4,d_5,d_6
electrical vbp,vbn,vb,iop,ion
#...Declaration of parameters
number gain=4  # Nominal Gain of the AGC
number zin=0.1 # Nominal Input Impedance of the input pins
number tt=0.1n #  Transition time of the output voltage
{

#...Internal Declarations
electrical v6,v5,v4,v3,v2,v1,ivbp,ivbn,ivb
val i ip,in,ioutp,ioutn,igain
#...Look for the next data input change
ide_d2an.6 d_6 v6 v5 \
=model=(voh=64,vol=0,vxh=32,vxl=31,tr=tt,tf=tt)
ide_d2an.5 d_5 v5 v4 \
=model=(voh=32,vol=0,vxh=16,vxl=15,tr=tt,tf=tt)
ide_d2an.4 d_4 v4 v3 \
=model=(voh=16,vol=0,vxh=8,vxl=7,tr=tt,tf=tt)
ide_d2an.3 d_3 v3 v2 \
=model=(voh=8,vol=0,vxh=4,vxl=3,tr=tt,tf=tt)
ide_d2an.2 d_2 v2 v1 \
=model=(voh=4,vol=0,vxh=2,vxl=1,tr=tt,tf=tt)
ide_d2an.1 d_1 v1 v0 \
=model=(voh=2,vol=0,vxh=1,vxl=0.9,tr=tt,tf=tt)
ide_d2an.0 d_0 v0 0 \
=model=(voh=1,vol=0,vxh=0.5,vxl=0.4,tr=tt,tf=tt)

# Analog system assignments
values {
         igain = gain*v(v6)/127
         ip = v(ivbp) - v(ivb)
         in = v(ivbn) - v(ivb)
         ioutp = igain * ip
         ioutn = igain * in
}
# Analog system simultaneous equations
equations {
       i(iop) += ioutp
       i(ion) += ioutn
        }
#...Netlist Section
#...Input Impedances
r.rvbp vbp ibp=zin
r.rvbn vbn ibn=zin
r.rvb vb ib=zin
#...Input Current Monitors
ccvs_4p.vbp ibp 0 ivbp 0=1
ccvs_4p.vbn ibn 0 ivbn 0=1
ccvs_4p.vb ib 0 ivb 0=1
}
```

Figure 8: AGC Model

The AGC gain is controlled by the digital inputs d_6 to d_0. These signals are defined as logic_4 digital pins, which means they have four discrete levels (high, low, don't care and high impedance). They are each fed into ideal digital to analogue converters that provide an output voltage proportional to their weighting as bits (32, 16,8 etc). These analogue voltages are then summed together to provide the analogue gain of the AGC, and this is multiplied by the input current (using equation(1)) to provide the scaled output current (iout).

The output of the AGC is fed into a quantizer (8bits) that also includes a companding function, so the thresholds are set at the voltage levels –1.0,-0.9,-0.5,-0.1,0.1,0.5,0.9,1.0. The quantizer is clocked using a digital signal from the Verilog control block. The MAST model for the quantizer is given in figure 9.

```
template   quantiser   pcas   ioptop   iontop   clk125
cmpout0 cmpout1 cmpout2 cmpout3 \
        cmpout4  cmpout5  cmpout6  cmpout7  =  irange,
zin, vos, tp

#...Declaration of connections
electrical pcas, iontop, ioptop
state                                        logic_4
clk125,cmpout0,cmpout1,cmpout2,cmpout3,cmpout4,cmp
out5,cmpout6,cmpout7

#...Declaration of arguments
number  irange = 1m,   # Input Current range
        zin = 10,      # input Impedance
        vos = 0.8,     # input voltage offset
        tp = 0         # Digital Delay from CLK to
latching output

{
#...Declaration of Analogue intermediate variables
electrical i_inp, i_inm
val i ip,in,idiff

#...Sequential Analogue Equations
values {
        #...Calculate the input current
        idiff = v(i_inp) - v(i_inm)
        }

#...Set Initial Conditions on the outputs
#...Clocked Comparator Equations
when(event_on(clk125)) {
        #...Falling Edge
        if(clk125 == l4_0)  {
                #...Comparator Equations
                if (idiff>=-1.0*irange) {
        schedule_event(time+tp,cmpout0,l4_1)
                }
                else {
        schedule_event(time+tp,cmpout0,l4_0)
                }
                if (idiff>=-0.9*irange) {

schedule_event(time+tp,cmpout1,l4_1)
                 }
                 else {

schedule_event(time+tp,cmpout1,l4_0)
                 }
                if (idiff>=-0.5*irange) {

schedule_event(time+tp,cmpout2,l4_1)
                 }
                 else {

schedule_event(time+tp,cmpout2,l4_0)
                 }
                if (idiff>=-0.1*irange) {

schedule_event(time+tp,cmpout3,l4_1)
                 }
                 else {

schedule_event(time+tp,cmpout3,l4_0)
                 }
                if (idiff>=0.1*irange) {

schedule_event(time+tp,cmpout4,l4_1)
                 }
                 else {

schedule_event(time+tp,cmpout4,l4_0)
                 }
                if (idiff>=0.5*irange) {

schedule_event(time+tp,cmpout5,l4_1)
                 }
                 else {

schedule_event(time+tp,cmpout5,l4_0)
                 }
                if (idiff>=0.9*irange) {

schedule_event(time+tp,cmpout6,l4_1)
                 }
                 else {

schedule_event(time+tp,cmpout6,l4_0)
                 }
                if (idiff>=1.0*irange) {

schedule_event(time+tp,cmpout7,l4_1)
                 }
                 else {

schedule_event(time+tp,cmpout7,l4_0)
                 }

        }
}

#...Input Stage
r.rp ioptop inp=zin
v.vp inp 0=vos
ccvs.1 i(v.vp) i_inp 0=1

r.rn iontop inm=zin
v.vm inm 0=vos
ccvs.2 i(v.vm) i_inm 0=1

#...Ibias Connection
r.bias pcas 0=10meg
}
```

Figure 9: Quantizer Model

The comparator function for each signal level is implemented using the schedule_event() function. This HDL construct schedules a digital event at a specified time (after the clock signal) depending on the appropriate decision criteria having been met.

Using the trans-conductance gain stage, AGC and Quantizer, the analogue data appearing at the receive channel pins is suitably processed for direct connection to the digital core implemented in Verilog.

3 Clock Recovery

With the analogue received waveform decoded into suitable digital words, the Verilog model of the digital core can extract the data, after first recovering the clock signal of the data stream. This is achieved using a classic Phase Locked Loop (PLL), implemented partially in the digital domain (phase comparison) and partially in the analogue domain (ICO - Current Controlled Oscillator). Each MTL3 symbol is decoded using a number of samples, from which the phase of the signal can be measured digitally. Figure 10 shows how depending on the decoded value, the phase can be determined for the symbol.

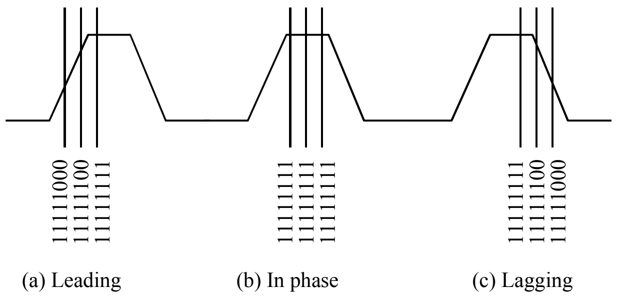

Figure 10: Phase Detection for Clock Recovery

This approach is more subtle than the traditional PLL method of 'brute force' phase detection and the digital core produces a phase and frequency trim command in the form of incremental digital data streams to the ICO input. The control function works on the principle that a phase up command produces a positive current pulse to the ICO input and vice versa for a phase down command. A similar mechanism operates for the frequency trim. The complete model for the receive ICO including the digital control function is given in figure 11.

```
template receive_ico ftracktrim fctrlisol ipllm
i20ux powerdown \
        pctrlisol phtracku phtrackd sicoiph_2
sicoiph_1 sicoiph_0 \
        txclk10_int iico vco_125 = iif , iph,
vref, fref, slope, \
        ftc, ptc, tt, rf, cf

#...Declaration of connections
state logic_4 ftracktrim, fctrlisol, powerdown,
pctrlisol, phtracku, \
        phtrackd, sicoiph_2, sicoiph_1, sicoiph_0,
txclk10_int, vco_125
electrical iico, ipllm, i20ux

#...Declaration of arguments
number  iif = 0.78u    # Internal Frequency
Current Pulse Magnitude
number  iph = 0.26u    # Internal Phase Current
Pulse Magnitude
number         vref = 0       # ICO Reference
Voltage
number         fref = 125meg  # ICO Reference
Frequency
number  slope = 2e12   # ICO F/I Slope
number  ftc=undef             # Frequency Loop
Time Constant
number  ptc = 0 # Phase Loop Time Constant
number  tt = 0.1n      # Current Pulse Transition
Time
number  rf = 15k       # Default Time Constant R
number  cf = 100p      # Default Time Constant C

{

#...Internal Declarations
state nu int_if, int_iphu, int_iphd, ifout=0,
iphout, ioutgain, iadj
state logic_4 vco_int, powerdownselect
```

```
electrical f,p,ia,pm
var i iia
number xftc

#...Parameters Section
parameters {
        if(ftc==undef) {
                xftc = rf*cf
        }
        else {
                xftc = ftc
        }
}

#...Calculation of the Current Adjustment

when(event_on(sicoiph_2) | event_on(sicoiph_1) |
event_on(sicoiph_0)) {
        if (sicoiph_2==l4_0 & sicoiph_1==l4_0 &
sicoiph_0==l4_0) schedule_event(time,iadj,-0.3)
        if (sicoiph_2==l4_0 & sicoiph_1==l4_0 &
sicoiph_0==l4_1) schedule_event(time,iadj,-0.2)
        if (sicoiph_2==l4_0 & sicoiph_1==l4_1 &
sicoiph_0==l4_0) schedule_event(time,iadj,-0.1)
        if (sicoiph_2==l4_0 & sicoiph_1==l4_1 &
sicoiph_0==l4_1) schedule_event(time,iadj,-0.0)
        if (sicoiph_2==l4_1 & sicoiph_1==l4_0 &
sicoiph_0==l4_0) schedule_event(time,iadj,+0.1)
        if (sicoiph_2==l4_1 & sicoiph_1==l4_0 &
sicoiph_0==l4_1) schedule_event(time,iadj,+0.2)
        if (sicoiph_2==l4_1 & sicoiph_1==l4_1 &
sicoiph_0==l4_0) schedule_event(time,iadj,+0.3)
        if (sicoiph_2==l4_1 & sicoiph_1==l4_1 &
sicoiph_0==l4_1) schedule_event(time,iadj,+0.4)
        }

when(dc_init) {
        if (sicoiph_2==l4_0 & sicoiph_1==l4_0 &
sicoiph_0==l4_0) schedule_event(time,iadj,-0.3)
        if (sicoiph_2==l4_0 & sicoiph_1==l4_0 &
sicoiph_0==l4_1) schedule_event(time,iadj,-0.2)
        if (sicoiph_2==l4_0 & sicoiph_1==l4_1 &
sicoiph_0==l4_0) schedule_event(time,iadj,-0.1)
        if (sicoiph_2==l4_0 & sicoiph_1==l4_1 &
sicoiph_0==l4_1) schedule_event(time,iadj,-0.0)
        if (sicoiph_2==l4_1 & sicoiph_1==l4_0 &
sicoiph_0==l4_0) schedule_event(time,iadj,+0.1)
        if (sicoiph_2==l4_1 & sicoiph_1==l4_0 &
sicoiph_0==l4_1) schedule_event(time,iadj,+0.2)
        if (sicoiph_2==l4_1 & sicoiph_1==l4_1 &
sicoiph_0==l4_0) schedule_event(time,iadj,+0.3)
        if (sicoiph_2==l4_1 & sicoiph_1==l4_1 &
sicoiph_0==l4_1) schedule_event(time,iadj,+0.4)
        }

#...Output Selection - choose either ref clk or
vco 125

when(event_on(vco_int) & powerdown==l4_0) {
        schedule_event(time,vco_125,vco_int)
        }

when(event_on(txclk10_int) & powerdown==l4_1) {
        schedule_event(time,vco_125,txclk10_int)
        }

equations {
        i(ia) += iia
        iia : v(ia) = 1.0 + iadj
        }

#...Calculation of the tracking frequency current
offset

or2_l4.f powerdown fctrlisol fen
inv_l4.f fen _fctrlisol
inv_l4.f2 ftracktrim _ftracktrim
ictrl.f _fctrlisol ftracktrim _ftracktrim f
0=vmax=iif,tt=tt
```

```
#...Calculation of the tracking phase current
offset

or2_l4.p powerdown pctrlisol pen
inv_l4.p pen _pctrlisol
ictrl.p _pctrlisol phtracku phtrackd p
0=vmax=iph,tt=tt

#...Filter for Frequency tracking

lpf1.f f 0 ff 0=tc=xftc

#...Multiply Phase Current by Modifier
multiply.1 p ia pm 0= 1

#...Filter the Phase Output
lpf1.p pm 0 pf 0=tc=ptc

#...Add together the total voltage
vadd.1 ff 0 pf 0 total 0

#...Convert Total Calculated Voltage directly into
a current

vccs.1 total 0 iico 0=1

ccvs_4p.1 iico 0 vi 0=1

vco_l4.1 vi 0
vco_int=vref=vref,fref=fref,slope=slope

#...Dummy connections for the bias pins

r.ipllm ipllm 0=1g
r.i20ux i20ux 0=1g

}
```

Figure 11: Complete Mixed Signal Receive ICO

Each of the phase and frequency digital control functions uses simple logic to include power down and enable functions to gate the command signals. These are then converted into current pulses. The frequency command is +/- 0.78uA and the phase command converts to +/- 0.26uA. These current pulse signals are low pass filtered to provide a smooth control signal to the ICO (the phase and frequency control have different filter characteristics. The frequency is much slower to respond than the phase giving the stability of a smooth clock frequency, but with the ability to respond quickly to fast changes in phase). The ICO is based on a reference frequency of 125MHz, and a sensitivity of 2MHz/uA.

The control signals sicoiph_2, sicoiph_1 & sicoiph_0 provide a further fine adjustment for the current levels output from the ICO based on the values in table 1.

sicoiph_2	sicoiph_1	sicoiph_0	I Adjust
0	0	0	-30%
0	0	1	-20%
0	1	0	-10%
0	1	1	0%
1	0	0	+10%
1	0	1	+20%
1	1	0	+30%
1	1	1	+40%

Table 1: ICO Current Adjust Values

4 Conclusions

With the architecture of the design in place, key elements in the design could be designed more precisely with an implementation in mind. Using test circuits, the behavioural model was treated as an electronic specification, to which the transistor implementation was to be designed to. As each element was designed as the transistor level, it could then be tested in the complete ASIC model, quickly and efficiently to ensure that it functioned correctly. The model was also used to help improve the accuracy of the behavioural model used. Using this approach, it was straightforward to test each block in the correct context, in turn, until the complete IC was simulated at both the behavioural and primitive IC level. This resulted in an ability to carry out much more circuit testing, including the input of measured data, and pre-calculated data with injected noise, to test the design under realistic conditions. This aided us in identifying design changes needed prior to the successful first turn of silicon taking place. Without the use of behavioural modeling techniques in this design process, the first turn of Silicon would not have been successful, and in all probability it would have taken several iterations to reach that point.

5 References

[1] IEEE Standard 802.3
[2] Mast reference Manual, Avant! Corporation, http://www.avanticorp.com
[3] Verilog Reference Manual, Cadence Corporation, http://www.cadence.com
[4] Concept, Cadence Corporation, http://www.cadence.com
[5] Saber Simulator Reference Manual, Avant! Corporation, http://www.avanticorp.com