**ISSCC 2011 Tutorial Transcription**
**DPLL-Based Clock and Data Recovery**
**Instructor: John T. Stonick**

---

## 1. Introduction

Ok, Thank you

## 2. Overview

So welcome this morning. I am here to talk to you about clock and data recovery. And really when we talk about clock and data recovery, the focus is really on clock recovery. You sort of get the data for free. You sample in quadrature to your recovered clock and you get the data. So we're really interested in somehow extracting a clock from an incoming data signal, and so we're going to talk first about just what is a CDR briefly for those of you who are coming in you know, without any background at all.

We're going to go through that pretty quickly and then we're going to talk a little bit about analog CDRs because really the DPLL based CDR is derived from an analog-CDR. Basically we're trying to take a good tried and true CDR implementation and somehow make it better, more flexible, easier to use and gain some advantages. And where we're going to do this is, we're going to do two things sort of hand in hand. One is, we're going to look at the architecture sort of at a block level and we're going to develop linear models and we're going to compare the linear model of the DPLL based CDR with the analog CDR and show that there's a correlation there and show the relationship between the various blocks between the digital and analog. And then we're going to use that linear model to do some design work and finally we'll kind of do a quick time step simulation to show how well that analog model works, I mean the linear model works.

## 3. Parallel vs. Serial Links

So first of all, once again, clock and data recovery. What does it mean? What is it for? Well, traditionally, what you had when you looked at links were wide parallel buses and a clock forwarded along with the data, and the buses were slow enough and wide enough and single-ended that you could use that clock that was forwarded to just sample the data at the far end of the link. Now, eventually what ends up happening is, you can only push those single-ended links so fast over a noisy ground plane say across a computer backplane, a motherboard, something like that, before the noise starts to kill you. Additionally it's hard to route once the bus gets wider and wider and wider, which is your only way now to increase the speed of that bus once you get to the single integrity limits of the single-ended signals. Then you start to have the problem that you can't route those signals. You need to keep them close together as you route through say a motherboard or something so you can recover those with a single clock. That type of architecture which was your old, everyone remembers like attaching your hard drive to that big old ribbon cable right. That was your ATA bus that was your big old wide. Each of those little wires was one of these lines right here. Now every computer has a SADA drive in it, Serial ATA. That big wide ribbon cable is the skinny cable. It just has a differential pair in it, and as long as you're going about saving this board space you go all the way and you don't even send a clock over. You just send the data, it comes into your analog front-end and you're going to sample that with the clock that you generate yourself. You don't rely on the other end to send you a clock, but rather you just have to pull it out of the data itself. Then in the end you deserialize that data or widen it back out and from the user's point of view, they can't really tell whether you came across on the serial bus or the parallel bus.

## 4. Clock Recovery

Now, what is clock recovery? We can think of clock recovery as more or less being a PLL. But it's a PLL where I don't get every clock edge. So once again, if I have a clock I've got all these regular edges and I can think of the incoming data from which I need to extract the clock as being a clock where I've got some missing edges. Now if my data is Nyquist, with a 101010 pattern, it does look like a clock, but I can have other data patterns come in where I may not have a clock cycle for 3, 5, or many, many edges depending upon whether the data is coded or not.

And once again, the CDR's job is to recover a sampling clock that is synchronous with that incoming data.

## 5. Clock Recovery Basics

Now, basically if I take the incoming data and make what's called an eye diagram, I basically take each UI coming in and overlay it on top of each other and you've seen maybe scope diagrams or something like that of an eye diagram, and basically what you see is you've got very fuzzy edges and in the middle you've got a clean eye and the goal once again is to sample the data in the centre of that clean eye, based upon information from these very noisy edges. I've got all this noise here and the idea of the CDR, just like a PLL once again, is to take the incoming clock and filter out as much of that noise as possible so you get a nice clean sampling instance in the centre of the eye.

## 6. CDR Types

Now, CDR types, this is not an exhaustive list but it kind of gives you a feel of what we are trying to talk about and how to frame what we're going to talk about. In clock and data recover circuits, traditionally you had an analog CDR and really once again, it just looks like a PLL. It looks like a PLL with a divide ratio of 1 right, because I'm trying to generate a clock that's at the same rate of data coming in, and the only disadvantage you have is that typically that clock in some sense is really crappy, because it's missing edges. That's what an analog CDR was like, it's basically just a PLL, and that PLL is distinguished based upon what kind of phase detector it had. Traditional CDRs either had a linear phase detector or they had a Bang-Bang phase detector. And once again a linear phase detector has some advantages, a Bang-Bang phase detector has some advantages but for a digital architecture we need ones and zeros, we need that Bang-Bang architecture, so that's going to be the starting point for our DPLL based architecture because we can't do anything digitally with the linear information from the linear phase detector.

Now in terms of digital CDRs there's really two broad classes there what are called oversampling CDRs. And this idea is pretty simple. What you do is I take, I oversample the incoming data much higher than the baud rate; say 5 times the baud rate. Then what ends up happening is in any UI ideally I would have say, 5 samples and what I try to do is I try to pick the middle sample for each UI. And the way I do that is by this high oversampling rate I can see where my transitions are because this bit was a 0, that one was a 1 so I know that is where my edge is and if I count over 2 more that should be the centre of my eye. That's basically the idea behind an oversampling CDR.

A really nice idea, very simple, the problem is, once you get the high baud rates you're driving a lot of high speed clocks. Now I have to drive 5 times as many clock lines, high speed clock lines have 5 times as many latches at the front end of my receiver. There's a cost to that kind of architecture. It does have some benefits in terms of its locking time but generally speaking once you get to higher and higher data rates it is very hard to implement that in a reasonable power budget.

Then we went to DPLL based CDRs and once again this is based upon an analog Bang-Bang CDR and the whole idea is to take the analog blocks in this traditional CDR and replace them with a digital equivalent and

the reason that we want to do this, the reason we want to have a DPLL based CDR vs. an analog CDR, is that it's going to save us power and area but if we're going to replace, get rid of our bias voltages, all their bias generator blocks are gone, capacitors, resistors, all these bulky components in a PLL are replaced with a handful of gates. A capacitor is replaced with a digital integrator. This big capacitor tiny integrator so it saves us a lot of time, a lot of area and we're not sensitive to PVT variation because once again, we've got digital things. I don't worry about what corner I'm in as long as I design things so I've got sufficient setup and hold across corners, the performance is exactly the same. Whether I'm fast-fast or slow-slow, doesn't matter because a CDR behaves exactly the same so that's a very nice benefit.

Another thing is testability and observability, because once again, inside my CDR, if things are digital, I could shadow register any signal I want in there, so I can observe anything and it's not like an analog circuit where I have to worry about if I'm going to attach say, I've got a control voltage and I'm going to measure that during operation, well I have to worry about the parasitic that I dump on that node when I try to measure it, I have to worry about if I'm actually dumping noise back onto that node as I try to measure it right, so measuring analog voltages while something is running is challenging and difficult and you can actually break something while trying to measure it, and make sure as you measure it if the measurements affecting what you see, whereas digitally once again, there's no problem there, just a number. Just read a number out of a register.

## 7. Analog PLL-Based CDR Block Diagram

Analog PLL based CDR block diagram, once again, it's just a PLL and I kind of broke out the phase detector from the PLL even though that's really part of the PLL to say that we talked about these two types. There was a Bang-Bang or linear phase detector. I'm going to generate 2 clocks and actually this diagram is a little bit goofy because the clock going up to the phase detector and the clock going down to the sampler of the data should be in quadrature with one another. I should have probably put a little bubble on that sampler to show that those 2 should be in quadrature with one another. But the idea is pretty simple once again, the CDR just kind of looks like a PLL.

## 8. Analog PLL-Based CDR Detail

Since it looks like a PLL, you have a simple model of a PLL right; the phase detector feeds early and late decisions to a charge pump. Charge pump dumps current onto a RC filter and that sets your control voltage for your VCO and that's your loop. And, once again, this is probably pretty familiar to most people because it's just a PLL block diagram. So then down at the bottom we have the small signal model, input phase and now I just say phase detector model right now because we've mentioned two types; a linear and a Bang-Bang and eventually we're going to get to a Bang-Bang model for that phase detector model there but right now we haven't developed that yet.

## 9. Phase Detector Types

Once again, phase detector types. Two main types: linear (Hogge). I apologize, I don't really know how you say his name, but the linear phase detector produces a late and early pulse on every transition, and basically, what it tries to do is eventually in lock those early and late pulses become of the same width. You jump basically the same amount of charge up and down from your charge pump and it's the balance of those charges that causes you to, your CDR to quiet down and settle down. And the nice thing is, the beautiful thing about, is that a linear phase detector once again, the difference in those charge, the amount of charge they dump on there, becomes smaller and smaller as I lock, so the CDR becomes quieter, and quieter, and quieter as I pull in so that's a really good feature.

The problem is that the linear phase detector has a few issues; it's very hard to implement it across PVT

without having some type of static phase error in there, and additionally like the XOR gate that you use in the linear phase detector are inherently asymmetric. We've got problems there that we need to overcome. Once again it's not impossible, people do it often, but there are reasons to use a Bang-Bang phase detector, so once again a Bang-Bang phase detector or an Alexander phase detector produces a late or an early pulse on each transition. I don't know how early or how late I am, I just know that either I'm early or I'm late. And since I don't have any measure as to how early or how late I am, the way that I tell I'm locked, is over a long time average, I should have an equal number of earlies and lates.

That basically in some sense, that charge pump instead of dumping equal amounts of up and down current, dumps either up or down current. But over a time period since I do have a filter there, it averages out to 0.

## 10. Digital View of the Bang-Bang Phase Detector

Let's take a digital view of this Bang-Bang phase detector. The idea is this, if ideally, I would sample right here in the middle. And if I did that that would be perfect right, sampling right where the crossings are. Now if I'm early, if I'm sampling where this red line is, what we observe is I'm sampling right here for data. Here's my say, early data bit or previous data bit, here's my next data bit and I've got the phase sample in the middle.

If I compare my phase sample to my data sample, that tells me whether I'm early or late. Here if I sample at this point in time, I can see that my phase sample agrees with my previous bit. On the following edge I started with a 1, I go to a 0; my phase sample would always be a 1 at that point. My phase sample agrees with my early data sample means that I'm sampling early. If I sample late where the blue line is, you can see that I always agree with the later bit. It's a very simple idea right. If I'm early I agree with the early bit. If I'm late, I agree with the late bit and that's all there is to a Bang-Bang phase detector. Very, very simple, and that's part of the beauty of it. There's very few places where you can create a problem with this type of phase detector.

Then once again there's no transition, there's just no information right. If the previous bit and the next bit are just ones, there's nothing to be learned, and so the output of this Bang-Bang phase detector is not really binary, it's really ternary, because you have an early, a late, or a no information. No transition.

## 11. Bang-Bang (Alexander) Phase Detector (Clock is Early)

Now, for those of you who have read papers on CDRs and things like that on Bang-Bang CDRs you often see this structure drawn as an implementation of the Alexander phase detector. And the basic idea here this is the implementation of what we just talked about pictorially. This is a way to build that circuit and the idea is the following and if you walk through this timing diagram you can see that it's true. I don't really want to take a lot of time to do this but if the rising edge of my clock occurs inside the data bit before the falling edge, if you follow through the logic you'll see that that causes the early gate, early pulse to come out. If you went through and had the other situation where the falling edge occurred inside the data bit first, the late pulse would come out. So once again, it's a way to generate these early and late pulses. So you can work through this and it's pretty simple and it's very straight forward and it's been published a lot.

## 12. Comparison of Bang-Bang Relations

Now the idea then is, a comparison of that Alexander phase detector vs. what we talked about pictorially is that, inside the yellow block I'm just taking basically a data sample and a phase sample. One of those latches samples in the middle of the eye, one sample is at the crossings. Then what I said was, basically I showed a look-up table right, and so the second part of that Alexander phase detector is just the implementation of that look-up table that we described in the slide with the eye diagram. And one of the things that's kind of neat

here is for an analog CDR; I would use that upper structure, because I need to generate those early and late pulses as each bit comes in. But once I switch to a digital PLL, I can actually just sample the data in the phase if I want to I can widen that data and slow it down and deal with it later, I can delay making those phase decisions because once I convert it to digital I can do whatever I want. I have a lot more flexibility there than I have in an analog CDR, and we'll look to see how we use that to our advantage later.

Now, let me just jump back a couple here.

**13. Analog PLL-Based CDR Detail**

Down at the bottom here we said we've got this linear model of - you don't have to jump back your pages because I'm just going to talk for one second. We have this linear model and we need to come up with a phase detector model, and so we need to do then is linearize this Bang-Bang phase detector.

**14. Linearizing the Bang-Bang Phase Detector**

Now what I'm going to do is just sort of introduce how you do it. I'm not going to work through all the math right now because I don't think that's that interesting. The result's more interesting. You want to learn about how we use these things, the mathematical details are in there for you to look at if you're interested, but we're going to kind of skip over those a little bit.

Now the basic idea in linearizing the Bang-Bang phase detector is the following: ideally once again, I would be sampling at this time, right at 0 and at that point in time my phase slicer, I can see that I've got an equal amount of signal above and below that slicer on this falling edge which means that half the time I would say that I'm early, half the time I would say that I'm late.

**15. Linearizng the Bang-Bang Phase Detector**

Now if I have a phase offset say φ, like instead of sampling over there're, I'm sampling over here, so I've got an error in the time that I'm sampling. Now what I notice is that I've got a lot more signal below my slicer than above it, which means that I'm going to say in this case get a 0 out more often than a 1. The point is that in a Bang-Bang phase detector what now when I'm late, it's now like every sample comes out late, right, because the 1's that are above this line. Once again, this is a 1, that's a 0, everything above that line's going to come out as a 1. That 1 agrees with the early bit. That's like a wrong value right. So even though I'm late, some of the time the Bang-Bang phase detector's going to say I'm early. But, it's going to say that I'm late a lot more often than I'm early. And for any phase offset, as I slide this phase offset back and forth what ends up happening is as I move further and further to the right, as I get later and later and later, eventually if I'm sampling way over here, I will always say that I'm late. If I go over far enough I'll always say that I'm late so that basically what I get is sort of like this linear increase in the percentage of time that I say that I'm late, until I hit this ceiling function, and what I need to do to get the gain is first of all, is figure out how to develop an equation for what's the average value coming out of that Bang-Bang phase detector given a phase offset?

**16. Linearizng the Bang-Bang Phase Detector**

Once again, arbitrarily assigning a 1 to a late and a -1 to an early, the mean value is just what we talked about there, it's the probability or the average value coming out of there given a phase offset. As I go further, it gets later and later it gets larger and larger and the gain is just the slope of that mean with respect to the phase. And then, furthermore one other thing that we need to do is remember that for any phase detector and a Bang-Bang phase detector in this case, I only get something out of it when I do have a transition, if there's no transition nothing comes out of it so that lowers the gain. On average random data has a transition density

of .5, that means on average every other opportunity I have, have an edge and so we de-rate our Bang-Bang gain by an extra factor of one half right here. I take one half times the partial of that average value with respect to the phase offset.

Now to compute that, we need to do two things; we need to understand really what is the standard deviation of the jitter coming in and we need to have some type of model, some type of PDF model in order to compute those probabilities. And there are 2 models that we typically would use; a Gaussian, and if I consider the jitter is Gaussian, a very RJ dominated channel, that would be a very clean channel where only RJ is the source of the jitter, or I could assume that the jitter is uniform. And so if it's uniform right, there's no tail at all and that would be if I had a lot of DJ in my channel and didn't have very much RJ. And in actuality, I will be somewhere between those two things. I've got a mix of RJ and DJ so it gives me a nice bound on this Bang-Bang gain.

**17. Uniform PDF (DJ Dominated)**

Now for the uniform case, I've got DJ dominated, that PDF looks like a uniform distribution.

**18. Pr(late φ) for Uniform PDF**

And here I've got a little bit of math that shows how I calculate the value of μ. Once again, how you do that calculation is not that important. In the end, what's important is the result down here that my probability of being late, given some offset that value of μ

that I was talking about is given down here.

**19. Gaussian PDF (RJ Dominated)**

Similarly, for a Gaussian PDF, I would assume that I've got a Gaussian distribution.

**20. PR (late φ) for Gaussian PDF**

I can work through some math, and once again, I can come up with a result down here.

**21. Gain for Bang-Bang Detector**

Now this table shows sort of the progression. First I go the probability of being late given the phase offset for the two cases for uniform and for Gaussian. Then I calculate the value of μ. So 1*(the probability of being late), + -1*(the probability of being early) and the probability of being early is 1-(the probability of being lat)e so you can end up with this equation right here. So 2*(that probability-1) gives me this line right here. Then ultimately I calculate the gain of that Bang-Bang phase detector as one half times that derivative.

Now the important thing here is what I've got now is a linearized gain based upon the standard deviation of the incoming jitter. And I sort of have some bounds on it too. If it's Gaussian, it's really $1/(\sigma*\text{sqrt}(2\pi))$, and if its uniform it's $1/(\sigma*\text{sqrt}(12))$. That gives me a range of gain that I would have expected to be somewhere in that range. If I have very little DJ, it will probably be more towards the Gaussian end and if I have a lot of DJ it will be more towards the uniform end.

**22. Verification for Gaussian Jitter**

To verify that this approximation, because it is a pretty big approximation, a pretty big step, works, what we

did was take a Bang-Bang phase detector, run Gaussian jitter through it, and offset the sampling phase, and sweep the phase through, and so this plot right here, is the value, the average value out of that phase detector for a given phase offset. So this curve here is really that $\mu$ plotted verses $\varphi$. And if I zoom way in right in near 0, the small signal part I've got a slope that I can measure that directly and I measure a slope of 3.98. If I do the calculation 1/(.1*sqrt(2$\pi$)), I get 3.99. You see that linear approximation is actually quite good.

**23. Verification for Uniform Jitter**

Same for uniforms since that was our other case and we just picked a different value of sigma just to make it a little bit more interesting. Once again, measure a slope of 1.89. We calculate 1.92. So once again, we went through this to develop a linear model of that Bang-Bang phase detector so we can use it ultimately in design and we've shown that it is actually a pretty decent approximation.

**24. Bang-Bang Self-Noise Term**

Now, one of the other things that happens though, is Bang-Bang is a hard non-linearity. If I'm going to model it with a linearity with a linear term, I can generate, I'm going to model that really, as a linear term, plus some extra noise and the extra noise is the rounding from the nonlinearity to the linear approximation. For very small phase errors, the Bang-Bang output noise is just the full magnitude of the slice data. I either get a +1 out or a -1 out. And so then, the average power of either a +1 or a -1, is just 1, and then the power is proportional to the probability of a transition. I only get something out of that Bang-Bang phase detector when I do have a transition. The average power, output power of that Bang-Bang phase detector is .5. And I can see how that affects really or how that impacts the system by input referring it, and I input refer it by dividing it by that gain that we just developed.

**25. BBPD Self Noise Input Referred**

Once again, up here I've gone through, just gone through the math, and crunched through directly through the math and what you see is that my ultimate results are down here at the bottom. And it looks pretty horrendous, so in essence my input jitter had an standard deviation of $\sigma$, and once I add in my input referred noise, that gets multiplied by the sqrt (7) for uniform, or sqrt(1 + $\pi$) for a Gaussian.

So (1 + $\pi$) that's about 4, it's like doubling the jitter, and it's even worse for uniform data. So the question is, if I generate that much extra input referred noise, why would I ever do this? It seems like a really stupid idea. And the reason why it's okay is that that noise is incredibly high frequency. This is a small signal model, we're assuming we're near a lock. So near a lock I'm getting earlies and lates one after another. That early and late data looks like it's really way out there near the Nyquist frequency. My CDR band width is going to be probably 2 and a half to three-quarters of magnitude lower than that, so I've generated a lot of extra noise. But fortunately for me I've generated it at very high frequencies that are easily filtered out by the CDR. So this linear approximation makes it look kind of horrible that you're generating a lot of extra noise, but in practice that noise is at a frequency that is easily filtered out. Sometimes you'll see this number thrown around that you're generating all this noise for the Bang-Bang CDR but in the end it's really not very harmful.

**26. BBPD Model Summary**

Once again, the model summary is that Bang-Bang CDR is gain, plus an additive noise term. The Bang-Bang phase detectors is inversely proportional, the gain is inversely proportional to the incoming jitter. So if I have a very jittery input, it lowers my gain. If I have a clean input, I get a higher gain. That's important in the end when you're testing a CDR, if you're worried about stability, you want to put in a very clean signal, putting

the cleanest signal you can, because that will expose any limit cycle behavior, any types of instability. If you're worried about whether you have enough tracking capability then you want to put in your noisiest signal, because that gives you your lowest gain in the system, so that will have, that's where you'll have your hardest time keeping up with the jitter. Typically you want to look at both of those test cases, not just noisy data but you also want to look at very clean data too.

## 27. Analog BB CDR Small Signal Model

Now, we went through all that stuff about the Bang-Bang phase detector, just so that we could replace that phase detector model with this $K_{BB}$ term. But now we really have a completely described small signal model for this analog CDR. That gives us a starting point to develop our DPLL based CDR. But before we do that, I just want to make a couple more comments on the Bang-Bang phase detector, and these deal not with the linear model but really with its practical use and application.

## 28. Practical Issues with Bang-Bang Phase detectors

Practical issues with Bang-Bang phase detectors. First of all, the loop gain is proportional to the transition density. We said that already. We just kind of said yeah, in long term average transition density is about .5. So we multiplied that derivative by .5 and called it done.

Now in practice, over short term, even over long term, transition density can vary quite a bit, even for 8b/10b coded data, I mean, that's the nicest data you will ever get. Nothing behaves better than 8b/10b coded data. And even for 8b/10b coded data, a Nyquist pattern is valid in 8b/10b coded data. Someone could send Nyquist to you all day long. So for Nyquist data once again, the transition density is 1. That's twice as high as we just assumed. And for 8b/10b coded data, you can have a long term, you know, forever data pattern that has a transition density of .3. So the ratio of your 8b/10b coded data you can have, let's say, your best case you've got a transition density that can vary about 3.3 to 1. So that means that your loop gain, you're trying to design, a loop where the loop gain changes 3.3 to 1. So clearly that's problematic. I wouldn't want to try to solve that problem. That's kind of a nasty problem, and later we're going to look at some techniques to reduce our dependence on that transition density, because once again, that's a killer, and if you have uncoded data, man, you're transition density can be all over the map. So for the best data it's bad.

Now the other thing I want to talk about briefly, are offsets in the phase detector. We said that the Bang-Bang phase detector was like a little more robust than a linear phase detector, but the one thing you have to worry about are voltage offsets. We've assumed that we've been slicing perfectly at 0.

## 29. Offsets in Phase Slicer

Now, if I don't slice right at 0, what happens? Here, instead of slicing right here at the zero point, I'm assuming that we have like a 40mV offset in the slicer, which is pretty large but it's just in there, you know, to show what's going on. Now this pink little oval right here, if my phase slicer is anywhere in that region, what's the output of my phase slicer? It's always 0 right, because my phase slicer's completely above all of this right here right? My phase slicer's always 0, my output's always 0 there. What's the impact?

Well on a falling edge this is a 1, and that's a zero. That 0 agrees with my late bit. So on a falling edge, every time, it's going to say late if I'm anywhere in this pink region. On a raising edge, my early bit's 0, my late bit's a 1 so on a raising edge I'm going to say I'm early. So every late edge, every falling edge is late, every raising edge is early. Those cancel each other out. And I can't have a raising edge without a falling edge, so in that region I really have no information, I have no idea if I'm early or late because I can't tell because I'm always going to get canceling early and late decisions. That's clearly a problem. And the question is, you know, how much impact does this really have?

## 30. Compare Average BB Output

We can observe this by once again, going back and now running that same kind of experiment we ran earlier to show that the Bang-Bang phase detector gain was reasonable. And here we put in Gaussian data and we sweep the phase offset, but now we have a voltage offset in our phase slicer, and this legend right here indicates what the phase offset was. Once again, these numbers don't really make a lot of sense but they run from -1 to 1 and my phase offset is really 10%, 20%, 30%, 40% of that phase detector output. And so what I see here is that if, actually I think that this, sorry I'm missing a 0. This should be .01, actually it is good, so that should be it's really 10mV, 20 mV, 30 mV, 40 mV, 50 mV. And what we see is that for the smaller offsets, because I'm still inside the fuzz, I'm never completely lost, so as long as my slicer stays somewhere in this fuzz, I do have some early/late information. I lose some of it because once again, there's a lot of for more of that I'm always below the slicer and my earlies and lates cancel out but there's still some if it that I'm above so I do get some region where I'm actually getting some information.

I get a softening of the linearity, and once again, once I get completely above here say at about 30 you can see that it really starts to flatten out and up to 50 I get a big old dead space in there right. Now this is something to be aware of. And once again from a modeling perspective, what we can do is de-rate our Bang-Bang phase detector gain by a little bit to account for this offset when we're looking at the linear model or we can ignore it while we're doing the linear model and then when you do the time step simulation later you can put that in, see the impact and maybe adjust some loop parameters.

## 31. DPLL Based CDR

So, that's the end of the discussion of the Bang-Bang phase detector. We've been here for a long time to learn about DPLL based CDRs and really it doesn't feel like we've made a lot of progress. But we've been laying a lot of ground work, a lot of foundation. Now, what we want to do is take that analog CDR, the model and the linear model, the structure in the linear model, and replace those analog blocks with digital equivalence, and we want to show the equivalence of these structures and then we want to show the equivalence of the linear models.

## 32. DPLL Structure

What does a DPLL look like based CDR? What I do is, I've got my analog CDR up here, and down at the bottom, I've got my digital DPLL based CDR, so my VCO is replaced by 2 blocks, a digital phase integrator, once again I want to get everything, I guess I should have noted this, everything right from the Bang-Bang output, to the inputs that digital phase converter, are just digital bits, those are all digital signals, just digital words, clocked data. My VCO is replaced by my phase integrator which is just an accumulator, and some type of digital phase converter. My charge pump and loop filter are replaced by a digital filter, and I replaced it in a way that mimics the behavior of that analog filter, so the charge pump and loop filter have a proportional path and an integral path, a path through the R and a path through the C, and we model that down here in a very similar fashion. Now we've got an extra block in this DPLL, which is called a decimator, and we don't have any analog equivalent of that because we can't really hold up the analog signal, so we're going to talk about what that means a little bit later, so that's something that's extra.

And then we've got the same Bang-Bang phase detector. So once again, the idea is, you take those analog circuits and you do, try to make in some sense, almost like a sampled version of that in digital land.

## 33. DPLL CDR Small Signal Model

Now, when I do that, I end up with this for a small signal model. I've got my Bang-Bang phase detector, I'll

have a linear model for my decimator, and once again, we'll get to that later. My charge pump, loop filter replacement is a proportional gain and then an integral gain, and once again $1/(1-z^{-1})$, that's just what a digital integrator, that's how you describe a digital integrator, and then a combination of my phase integrator and digital phase converter are modeled as gain over another integrator, and then because this is a digital system, I want to explicitly model all the delay in the system. So in other words, I'm going to have some delay going into the analog blocks, but then, because I'm doing digital operations, those are clocked operations, and there's going to be real delay in there, so I need to know how many cycles it takes me to get around the loop. And I just put those explicitly into my linear model.

## 34. Loop Gain Comparison

If I compare now, my analog loop gain to my digital loop gain, we observe the following. So charge pump on the R looks like the phug term. The charge pump onto the integrating capacitor, looks like the frug term. My VCO looks like that phase integrator digital phase converter combination, Bang-Bang phase converter is the same, Bang-Bang phase detector is the same in each circuit, and I only have a decimator in the DPLL.

## 35. DPLL Advantages

Once again, to go back and harp on the advantages, we'd get to replace all these analog components with digital circuits. We'd get rid of all the PVT variations, doesn't matter which process corner I'm in, multiplying a digital number by another digital number, always gives me the same answer, there's no PVT variation.

And the other great thing, once again, is reduction of area. That cap in that PLL gets replaced by a little digital integrator, so that's really nice and we've talked about observability and testability.

## 36. DPLL Design

Just reiterating those advantages. Now, in order to do design with this DPLL based architecture, you know, maybe you've done a lot of PLL design in your life, so you look at this analog CDR, and you sort of have a sense for, yeah I know what my control voltages, what my VCO is going to be like, and I know what kind of charge pump currents I'm going to generate, and I have a reasonable idea how to size the R and the C.

You may have a lot of intuition based upon your analog experience. We want to sort of now go through and develop that same type of intuition for the DPLL based designs. We need to figure out, now I've just got a bunch of digital numbers, how do I figure out how to set those, how to evaluate those and come up with numbers that will make sense and make the circuit operate properly. And the best way to do that is to take that architecture, and walk through it and understand how each of those blocks is going to affect my output sampling phase.

## 37. DPLL Structure

Once again, I have this open loop gain and here's just basically the architecture where I've broken the loop, so it's the open loop architecture and the open loop gain up above.

Once again, I've got these values for K and phug and frug. By the way, I used phug as p h u g, that's the phase update gain, and f r u g, is the frequency update gain, that's where those terms come from.

And then I've got some $K_D$ term, and once again, when I look at this in the sample system, the question is, how do I pick values for those that are going to make this system work? And the best way to understand how

to do that, is to look at my sampling clock at the output and walk backwards until I finally understand, hey, when I make a Bang-Bang phase error decision, ultimately it has to have some impact on the sampling clock. I want to understand how each of these blocks affects it as it passes through the system.

## 38. Digital Phase Converter

My digital phase converter, this is going to convert a digital code from that phase integrator, and once again that phase integrator is just an accumulator, into an edge location, with respect to my local reference clock. Once again, this clock and data recovery system, I've got some far end transmitters sending me data, his clock may be a little bit off from mine in frequency, so I've got a local reference clock and I've got edges from that local reference clock and I need to somehow use this digital word to take two edges of a local clock and generate a clock that's in line with the edges of the incoming data. And so really, the parameter that's importance here, is N, the number of bits of that control word, to tell me how many phases I have through a UI. So if I've got an N bit control word, I've got $2^N$ steps, $2^N$ discrete possible edge locations of that reference clock, or of my sampling clock. So there's a quantization error here vs. an analog CDR. For an analog CDR, I could really, theoretically move the edge anywhere, for a digital DPLL based CDR, I've only got these $2^N$ choices.

Once again, we have to make sure that we pick a value of N, that we can live with that quantization noise.

## 39. Phase Interpolator Functional View

In terms of the system view of this, the way to think of this is, if I think of my chip, I'm going to have some rough clock coming in, and I'll have some master PLL there, and that master PLL, it's going to generate the clock for my transmitter and it will also generate my reference clocks for whatever type of phasing interpolator I've got. And once again, that fundamental frequency of that master PLL, may be at a slightly different frequency than the data coming in.

And then I'm going to generate from out of that phase interpolator, my digital phase converter, I'm going to generate a data sampling clock and a phase sampling clock and eventually I'll sample the data in phase, and I'll go through that Bang-Bang phase detector, work out that whole CDR loop. Coming to the phase integrator, and that phase integrator is going to send those N bits over to tell me how to adjust that phase interpolator.

## 40. Phase Interpolator (Rotator)

And the way to think of the phase interpolator, I think of it as being a circle. So a UI is like a circle, because I could continuously rotate the phase forever, I could just wrap this thing around and around and around, and so here what I've drawn is once around the circle is like one UI, and here I think I've drawn 32 little dots there, so I have 32 discrete sampling steps. 32 is a reasonable number for a phase interpolator, you probably don't want to be a whole lot coarser than that, and being finer than that just lowers your quantization error, which a good thing at the cost of some complexity. So, once again, depending on your design, you may want to increase that value, but 32 is probably like the minimum that you want and a reasonable starting point.

Now, the digital phase converter is an analog circuit, digital phase interpolator, phase rotator, so what are the problems I would have in terms of, like analog circuit imperfections and how will they affect my CDR? Because, once again, we're trying to get away from analog circuit imperfections, trying to get away from PVT problems.

Well, things that I worry about, one of the things that I worry about are glitching. This is probably the main

major concern. I've got a digital word controlling this phase interpolator and if I go from say, 0111 whatever, to 10000, that's just a click of 1, but it's caused every bit to change on that bus driving my phase interpolator. That's one of the things you have to be very careful about and look at very carefully in the design of a phase interpolator, that as I make this very tiny change, it causes all the bits to flip that I don't cause a huge glitch.

Another thing that you might be concerned with is, INL and DNL. In the end, it turns out that if we've decided that we can live with the quantization error that we have, the INL and DNL causes very minor variations on that. So you've probably got a reasonable amount of quantization noise in the system and we decided that's okay. My INL is really causing, instead of those phases being perfectly equally spaced on this unit interval, some of them a little bit closer and some of them are a little bit further apart.

But once again, it's a minor, minor problem and really doesn't add a whole lot of disturbance to the system. If you simulate a CDR, and you increase the INL, you have to increase pretty drastically before you really see an impact on your sampling jitter or degradation on your error rate.

## 41. Impact of Reference Clock Edges on PI Output

Now, one of the things that I just want to say in passing, is that when we design a phase interpolator, there's sort of an odd problem, and it's an odd problem in the sense that, typically when you think of clocks, you think of nice crisp edges. And for a phase interpolator, you need those reference clocks to be completely the opposite. Ideally they would be triangular shaped.

So what I've shown here are two situations. In each case I'm generating 32 possible output phases and in one case, my input clocks, my four input clocks given by the cyan, black, magenta, red are sinusoidal, not perfectly triangular but reasonable, and these blue lines right here show all the possible output phase clocks I could make by blending those two phases together.

You can see that if I look at the 0 point, I've got pretty good, fairly uniform spacing. If you look really close, you'll see they get a little closer together here, a tiny bit further apart there, that's INL that we can live with. If on the other hand, I accidentally give edges that are too sharp and try to blend those together, you can see the horrendous performance. So once again, when these edges are very steep, these phases are really close together and when I get up to dealing with flat edges and get on that flat point, I get phases that are really far apart. So in that case, I would get really horrendous INL out of this and that would be problematic.

And so your goal, once again, sort of like your analog design goal, or challenge, in DPLL based design, is one of the major ones is, once again making some type of phase mixer, phase rotator that has not too much glitching when I have bit code jumps. And the other thing is, trying to keep those reference clock edges into the phase interpolator looking fairly triangular or sinusoidal, across PVT.

And really, that's the analog design challenge that you're dealing with mostly in this DPLL based design. The thing about a whole PLL based design, analog CDR design, we've really taken out most of the analog design challenges.

## 42. DPLL Structure

Now, I've spent a lot of time talking here, about this digital phase converter. In the end, we've said ultimately what we want to do is pick a value of N that we could live with the quantization error. We said from an analog point of view, most of the distortions that we want to design with some reasonably sense of care, are not going to effect the loop performance very much.

Now that tells me, this develops the edges, or generates the edge that I'm going to use to sample the clock in the data. Now what drives that is my phase integrator.

## 43. Phase Integrator

The phase integrator is just a digital accumulator, so it's just a digital accumulator, and it is the digital representation of the sampling phase through the digital phase converter. So whatever number is in there, corresponds to some actual sampling time. Now, since our digital phase converter is N bits wide, you might first think, okay, that phase integrator is going to be an N bit accumulator, because I need N bits. But typically, what we'll do is we'll have more bits in our phase integrator, so we're going to have these sub-resolution bits, and what that does is, by varying the number of sub-resolution bits, it allows me to vary that loop gain parameter. So the one loop gain parameter that we had to find out, was $K_{DPC}$. What $K_{DPC}$ is now, when making a change of 1 going into the phase integrator, it's going to move the output phase by a certain amount. It's going to move my sampling phase by a certain amount. And so if my digital-to-phase integrator is, say, 32 clicks around, if my phase integrator matched that, it would take me 32 steps to go around there, or my resolution would be $1/32^{th}$.

If I got some throw away bits in my phase integrator, what that does, it takes me several steps through my phase integrator before I click what my digital phase converter wants. So while that resolution doesn't really exist, I'm not really moving fractions of a step of my digital phase converter, looking backwards or looking into the phase integrator, it sort of seems that way.

We'll go through an example to sort of explain really what I mean. So the example that we're going to focus on is going to have, say, once again, we're going to assume 32 steps in our digital phase converter, N is going to be 5 bits, we're going to have 2 dither bits below that.

## 44. Phase Integrator Example 1

Let's see what happens. Once again, I'm going to think of my phase integrator, I'm going to color the bits blue and red. The blue bits will be tied, say, to my phase interpolator and the red bits, I'm basically throwing away, they're sub-resolution bits.

If on each consecutive clock cycle, for whatever reason, I'm putting a 1 into that phase integrator, I'm trying to walk that phase integrator. What ends up happening is, the phase integrator, starting at a state of 0 goes to 1, 2, 3, and then when it gets to 4, I finally change one of the blue bits. What it took was, it took 4 steps or 4 input steps to cause me to move 1 output step.

That's how I get a gain reduction through these sub-resolution bits. So once again, it acts like I move. I can think of it, it sort of seems, from a linear point of view, that I moved a quarter of a step, each of 4 consecutive steps, and that's the idea, once again. The whole Bang-Bang CDR`s based upon time averaging, so it kind of feels to me, looking at the input of the phase integrator, that I moved a quarter of a digital phase converter step, each of a UI, for 4UI, when in fact I didn't move at all for 3, and I took a step on the $4^{th}$.

## 45. Graphical View

Graphically, again the way to look at this, these black lines are really and these green areas indicate the digital phase converter output values where they are the same, and as I walk through this, it feels like it from a linear point of view, it looks like I'm really walking the phase through these sub-bits steps, even though the output phase doesn't really move. That's just a visual way to think about this as well.

## 46. Phase Integrator Example 2

Second example is, hey, what if I'm dithering back and forth? Plus 1, -1, plus 1, -1, one of the nice things is that the phase integrator is going to eat that. So once again, my sub-resolution bit will dither back and forth, while my upper bits don`t change.

## 47. Graphical View

And that doesn't happen everywhere. If I'm at a boundary, like right here, and I'm dithering back and forth, I will dither my upper bits but for most of the time, and if I have a lot of sub-resolution bits, for a very small fraction of the time, or most of the time, basically, any kind of dithering back and forth, doesn't show up on my output, so that sub-resolution bits help quiet the CDR.

Once again, that was one of the things we said we liked about a linear phase detector based CDR was that as you lock in, you get a much quieter output. Here, these sub-resolution bits help us get to that same point by basically hiding the dithering on the output sampling phase.

## 48. DPLL Structure

So now we've gone through and we understand that the gain, $K_{DPC}$, for a linear model is really how many steps it takes from my phase integrator's point of view, how many steps does it take to go around the UI?

Now we want to look at my digital loop filter and understand how that feeds into that phase integrator.

## 49. Digital Loop Filter

Once again, the purpose of this block is to act like the charge pump loop filter. And so, we want to have a proportional path and an integral path. And we can control all these things are digital, so I've got this value of phug, and this value of frug, that I've got control over. Those are just digital numbers. So the nice thing is, I can dynamically change those, I can make the CDR behave in lots of different ways by controlling those phug and frug values and I may increase phug at start-up to pull in faster, I may set frug to 0 to begin with until I get some semblance of a lock, there's lots of things you can do. I can look at the data and say, hmm, if I'm seeing commas when I'm supposed to, that's good, let me turn down the gain to my CDR, because you have so much control, you can do a lot of cool things.

But like an analog CDR, once again, most of the loop dynamics, the bandwidth and that, are controlled by the proportional path. The purpose of the integral path is to try to compensate for long term frequency drifts, just like in a PLL, the voltage on a capacitor is there to compensate for really, the clock frequency difference between the far end and the near end. And then, sort of like your jitter trajectory's off of that base clock difference, are tracked out by your digital loop.

## 50. Proportional Path (phug)

Once again, I'm going to take my phase error decisions and multiply them by some kind of gain. Typically, 1 is your lowest gain, and you'll use 0, 1, 2 or 4, and turn off my CDR, freeze it, and it will hold state forever, if I set phug and frug to 0. If I set those values to 0, I can freeze my CDR, you can't do that with an analog loop, you've got charge on a capacitor, eventually it's going to leak off. Here I can perfectly freeze my CDR.

Now the proportional path, once again, is going to add in some amount based upon the gain factor of phug,

and once again, that's going to tell me, how much I move my phase in response to the phase error.

So I've set a minimum resolution to my phase integrator, but I can take bigger steps if I want, by increasing the value of phug. And once again, typically, we use values of multiples of 2 because then my multiplication become just shifts digitally.

## 51. Integral Path

The integral path is a lot more interesting. So once again, like in an analog based CDR, like in PLL, the voltage on the capacitor, it's trying to compensate for any type of static frequency offset between the far end and the near end. And so, the thing that happens here is that the DC output or the value in that frequency integrator gets added into my phase integrator on every cycle.

So in other words, if I was adding in, say, I had a 1 in my frequency integrator, I would take 1 step through my phase integrator on each UI, and I'd slowly walk around. So if my incoming clock is walking with respect to my local reference clock, I keep up with that by walking my phase integrator, by having a static value in my phase integrator. Because a DC value in a frequency integrator, once again, gets added on every clock cycle to that phase integrator. That's going to cause the phase to advance, or if it's a negative number, it will cause the phase to recede. And it will track the difference, the frequency difference between those 2 clocks.

So once again, an important thing is that, the frequency integrator, because we need to walk forwards or backwards, is going to be signed, and it's 2's complement arithmetic, and basically you sign extend that, you add that to the phug path and then to the phase integrator.

The only thing is that, we talked about the phase integrator being continuous rotation. Typically you'll want to saturate your frequency integrator because, once again, if I rolled that over, I'd go for my biggest positive number to my biggest negative number since it's signed and that would be unfortunate.

## 52. Frequency Integrator Details

Now, the details. We talked about having these upper bits and dither bits in the phase integrator. We have the same concept in the frequency integrator. Once again, we've used different letters to describe them.

But M are the bits, the upper bits of that frequency integrator that will dump into the phase integrator, on every clock cycle. Just like the upper bits of the phase integrator went to the DPC on every clock cycle. Once again, a 1, coming out of that frequency integrator in those top M bits, is going to move the phase integrator 1 click each clock cycle. So since my phase integrator has $2^{N+Dp}$ phases, I'll move 1 over that on each clock cycle for a value of 1. Since I've got M bits and it's signed, I can move anywhere from $-2^{M-1}$ to $2^{M-1}-1$ basically the plus and minus 2's compliment numbers, anything in that range so I can move 1 or 2 or 3, if I've got 3 bits up there, I could move anywhere from plus 3 to minus 4 on a given clock cycle. And the biggest number that I can move tells me what my maximum frequency tracking capability is.

So in other words, as my clock difference gets larger, between my far end and my near end, I need to take bigger steps in my phase integrator, just to keep up with it. At some point, because this is a digital phase integrator, this digital frequency integrator, there's a biggest number that I can put in there. And that tells me how fast I can move that frequency, or the biggest frequency offset that I compensate for.

## 53. Frequency Integrator Example

Now little concrete example to try to make it a little bit more sense. I know a lot of these concepts are quite

abstract, so from time to time, I try to do a little bit of a numbers example to try to make things a little bit more sense.

If I have a phase integrator that is 8 bits, I've got 256 steps per UI, and if my frequency integrator has a value of 1 in it, in other words, I'm dumping a 1 in on each clock cycle. Let's understand what the impact of that is. So basically, I would be moving my phase integrator 1/256UI every UI, is that a good thing, is that a bad thing? What does that mean?

Well, typically, clock differences are specked in PPM, parts per million difference. Say, for most consumer electronics specs, usually your plus or minus 350 PPM, say as a common bounding range for clock difference, you know, clock uncertainty, we'll say. For networking applications, it's tighter, maybe plus or minus 100 PPM. With this 8 bit phase integrator and a value of 1, coming out of my frequency integrator all the time, what does that mean in terms of PPM, because that's what's interesting to me? Well I'd be moving 1/256UI per UI, if I just do the math, that's 3,906 PPM. That's a big number, right? I just talked about numbers that were in the low hundreds, and this is in the low thousands. This seems about 10 times as much as I want. So for that 8 bit phase integrator, gosh, I don't even want to put a value of 1 in that frequency integrator. Or I don't even want a value of 1 in those top bits, it's too much.

We talked earlier that I'm going to have some dither bits, or some throw away bits at the bottom of that frequency integrator. I'm sorry, some dither bits in the frequency integrator, in the phase integrator, we just threw those away, we just rounded those off. In the frequency integrator, we need to use that sub-resolution, I need to somehow quantify those sub-resolution bits in a way that actually moves the phase integrator, I can't just throw those away, because obviously my step sizes are way too big here. I could fix that by putting a lot more bits of my phase integrator. If my phase integrator had 10 more bits, then I would have, if it was 18 bits, I'd get to divide by an extra thousand and I'd have 3.9PPM, which would be really sweet. But nobody wants to build an 18 bit phase integrator.

## 54. Frequency Integrator Sub-Resolution

To sort of quantify what we're looking for roughly, or a way to think about this, is a 100 PPM offset is something, say, reasonable you might see in a system, so if I want to compensate for that, with my frequency integrator, how do I walk my phase integrator? Once again, in that 8 bit phase integrator, I have a 1/256UI step. That's .0039UI. And if I want to compensate for 100 PPM offset, I have to move .0001UI/UI, so there's an extra 0 there. So if I add a 1 to my phase integrator, every 39UIs, instead of every UI, then I move .0039UI/39UI which averages out to 0.0001/UI per UI, which is what I was targeting. So what I need to do now is somehow, take those sub-resolution bits and use those to occasionally add something to the phase integrator. So most of the time I'm adding 0, but sometimes I want to add something, and the lower the PPM offset is, the less often I want to add it. The higher the PPM offset is, the more often I need to add it in.

So that's the idea, now how do we realize that, how do we build something that does that?

## 55. Sub-Resolution Time Averaging

Well, it's sort of an odd thing, but what you do is sort of sigma delta modulating those lower bits and using the carryout as this occasional bit that you're going to add in. And I'm going to show you the structure right here, it may not make perfect sense, and then we'll go through a little example, we'll go through two examples and hopefully by the end, we'll sort of have a good sense as to how this really works.

Once again, I've got $D_f$ bits of the bottom of my frequency register, and so what I've got is a $D_f$ bit accumulator here, it's $D_f + 1$ bits, I've got $D_f$ plus a carry, and so I've got a state here that's $D_f$ bits, I've dumped $D_f$ bits in it, and I see if I get a carryout. And every time I get a carry, I add an extra 1. So my

frequency integrator is signed, but the sub-resolution ΔΣ is unsigned. But at first, that seems wrong, how could that possibly work? I'm going to take these signed bits from this frequency register, and use them in an unsigned fashion. It doesn't seem like it would work.

**56. Sub-Resolution Example: ¼**

Now, let's see how it actually does work and show that it does work. Once again, I can only add a positive number. I can add a 0 or a 1, my carry is 1 bit, it's either 0 or 1, so I can only add a positive number.

If I want a sub-resolution of a quarter, of a quarter of a phase step. I need 2 sub-resolution bits in my frequency register and then once again, if I have a 1 sitting in the very bottom of my frequency register, and I've got 7 bits in that frequency register, the top 5 bits are all 0, my bottom 2 bits are 0 and 1. That's the smallest value I can have in there.

So now what happens? My ΔΣ state, I'm going to start at state 0, so once again, that's sigma delta is

**57. Sub-Resolution Time Averaging**

this thing right here, so the ΔΣ state is going to be this guy right here, this is my input, and I want to see it come to my frequency register and I want to see what happens with this carry bit, that's I'm trying to see here.

**58. Sub-Resolution Example: ¼**

So my ΔΣ state is 0, $D_f$ is a 1, so on every cycle I'm going to add 1 to that ΔΣ modulator and the red bit there is the carry out. So I start with 0, add 1, carry 0, add 1, add 1, finally on the fourth time through, I get a carry out. So what's happened here? By using this structure, I was able to output a 1, every $4^{th}$ clock cycle. That's the same effectively as putting out a quarter of a bit, every cycle. Once again, it's this time averaging concept, by putting out a 1 every $4^{th}$ clock cycle, I've achieved effectively a quarter bit resolution.

So the phase integrator, we threw those away, we said, it's sort of like doing that, but we just live with the quantization error. Here, we actually do it, we actually get that sub-resolution and we realize it on this time averaging basis which is, once again, what ΔΣ modulators are for. Then, the gain through that frequency path, once again, then is whatever I've got multiplying by that frequency update gain, divided now, by $2^{Df}$. In other words, a 1 in that frequency integrator because I have 2 bits, looks like a quarter.

Great, talking about this is an unsigned process and I gave you an example of the positive value. It's like cheating. Because when I have a negative number in the frequency register and I'm treating this as unsigned where you think you might have trouble.

**59. Sub-Resolution Example: -¼**

Now, let's put the smallest negative 2's complement number into that frequency integrator. So what's your smallest number in 2's complement? All 1s right? And so, in that case, this should represent negative of a quarter. If the other 2 bits were positive a quarter, I would hope that this would represent negative of a quarter. Let's see what happens.

So my upper bits are all 1s. Since this is a signed quantity, that means that my upper bits going to my phase integrator, are a negative 1. So now in every clock cycle, I'm going to be adding a negative 1, from the upper bits of my frequency integrator. Now my lower bits on my frequency integrator are 11. Now I dump that into

my ΔΣ modulator and what comes out? Well now, instead of a 1 coming out, 1 out of 4 times, a 1 comes out, 3 out of 4 times. So I started off with a carry of 0, and then I get 3 carries of 1 in a row. So what I get when I add the output of that ΔΣ modulator back into those upper bits is, I get -1 + 0, that gives me a -1, but then, 3 times in a row I get -1, +1, okay, and they cancel each other out. I get a net 0 going through the phase integrator. So I got it, right? One out of 4 times now I get a -1, 3 out of the 4 times I send a 0.

So by treating this as an unsigned quantity, I'm able to make both positive and negative fractions and they behave as they're supposed to behave.

## 60. Frequency Integrator Summary

Now, quick summary is once again, mathematically we can describe what my maximum PPM tracking capability is, by saying, what's the biggest step out of my phase integrator, divided by the number of bits per UI, and for non-spread-spectrum applications, typically having the capability to track about + or – thousand PPM is far more than adequate. If I'm in a spread-spectrum application, typically the spread limit stay down around -5 thousand PPM, so you would want to be able to track at least + or –5 thousand, say + or – 8 thousand gives you a reasonable buffer to that 5 thousand limit in the specs.

The effective resolution then is, or really how much I move the output of the phase sampling clock with respect to the input to my frequency integrator, while I have to divide by the sub-resolution bits of my frequency integrator and then divide by the number of steps per UI. Because remember, when I put that 1 into my frequency integrator, it takes me many cycles before that 1 spits out. Most of the time, say I have 2 bits, 3 out of 4 times, I'm spitting out a 0.

And once again, those dither bits produce a fraction in the range of $1/2^{Df}$ divided by you know, roughly, N/(N + 1) number.

And I'm also concerned with in a frequency integrator, how fast I can change the frequency when I'm dealing with spread spectrum type applications where the frequency is being moved on purpose.

## 61. DPLL Structure

Now let's go back and sort of think back to what we've learned. We have this digital phase converter that gives me N steps around a UI. And then I sort of make that appear like more steps by going through this phase integrator, then my phase error decisions coming in and go through this digital loop filter. And we showed how that frequency integrator dumps something into that phase integrator on every clock cycle to track clock offsets, frequency offsets, and I get a proportional path where that phase error comes in, gets multiplied by a gain and immediately affects the phase integrator output. So that gives you immediate response to a phase, to some type of phase trajectory whereas my frequency path, once again, I integrate that, I average that, so on average I want to know if I should be adding something to my phase integrator, whereas, the proportional path is like immediately reacts to what's happening.

Now, we have now equations for all the parameters, except for $K_D$, the decimation. I mentioned that this is something that we only do in a DPLL, and now let's talk about that a little bit, and what we're going to find out, is actually the decimation does have some effect on these parameters as well.

## 62. Decimation

Decimation, once again, it is basically, widen the data and slowing it down. So having a 1 bit signal at 5GHz digitally, is the same as having 10 bits wide at 500MHz. It's the same amount of data. So if I take the data and I deserialize it, I widen it, I get a bus of some width, running at a lower clock rate. And once again, that's

nice, because now I can distribute a lower speed clock to do a lot of these operations if I decimate properly.

My phase integrator which is however many bits wide, runs at the slower rate. My frequency integrator runs at the lower rate. I burn less power; it's easier to lay it out to get it to work. If you slow it down enough, you can even place and route it, you can write this CDR in RTL, give it to your place and route guy, and boom, you've got a CDR.

So, that's some of the reasons why we want to do decimation. We want to save that power and want to make it easier to implement.

Now, by decimation once again, it's taking L phase errors where L is some, typically the power of 2, and somehow doing something with them to come out with a wider bus that's L bits wide, say, at a rate of 1/L. But one of the things that we have to understand is, if I slow down the clock rate of the CDR, one of the things that happens is, we talked about how the frequency register dumps something in on every clock cycle. And up to now, we were assuming that was in every UI. But if I only dump something in, every, say, 4UI, I'm dumping it in 1 fourth as often, so on a per UI basis, I'm only dumping in 1 quarter of that value of each UI. So it gives me a further gain reduction on that frequency path, because once again, it's just like those sub-resolution bits, they worked by occasionally dumping something in. And if I slow it down further by only dumping it in every so many UI, by even considering dumping it in every so many UI, it's sort of an identical kind of gain reduction as having the dither bits. I hope that makes sense to people.

Once again, all of this, really comes down to understanding DPLL based CDRs is, gosh, darn it, everything is about time averaging. Things happen, you take sort of a big step and then you just pause and rest for a little bit, and that's really how they work.

## 63. Decimation by Summing

Now, there's two ways to do decimation. One is just by summing, and one is by something called voting, we'll talk about summing first. It's a little bit more straightforward.

Summing is very simple. I'm going to take my L phase error decisions and I'm just going to add them up. And so basically, my phase error coming out of my Bang-Bang phase detector, I have three possibilities, remember? Early, late and no transition. So -1, 0, or 1. So, I'm going to take L of those, let's say, L is 4, and I'm going to add those 4 numbers together, so I get 8. A number with a bigger range, at a lower rate. So, instead of having a -1, 0, 1 it would be a 2 bit bus at the baud rate, I end up with something that could be anywhere from say, -4, to +4 at 1/4th the baud rate. So in some ways, this is like a pre-add for that integrator. But my phase integrator was going to add those numbers together anyway. It was going to add up all those phase errors. I'm just doing some of the work for that phase integrator in advance.

Now, what ends up happening is that phase integrator still has the same width, still looks the same, but it gets to run at a lower clock rate. What I've got is the cost of that is I've done it in two separate stages so I've added some latency into the system. And we'll talk about that a little bit later. Latency is something that we want to avoid. There's a tradeoff there, we can't just keep doing this forever.

The loop gain of a proportional path is unchanged because I was going to add those numbers together in a phase integrator; I'm just adding them in advance. The output range is going to be a bus that's wide enough to carry this bigger number and it's going to run at a lower rate.

Now earlier we talked about one of the problems with the Bang-Bang phase detector is that we're sensitive to transition and density. This type of decimation does nothing to shield me from the transition density, because I'm considering every transition. Every transition either gets added as a -1 or a +1, so I haven't done anything to help myself in terms of sensitivity to transition density. To do something about that, I need to

collapse multiple phase error decisions into a single decision. That's the only way I can reduce my sensitivity. That way, over my L samples, if I don't care how many transitions I had, if I treat L transitions as the same as 1 transition, then I'm no longer sensitive.

## 64. Decimation by Voting

So that's what voting does. I take the L samples, and I just vote over those L samples, did I have more early samples or more late samples? Or an equal number? So if I have more early samples, I vote -1, if I have more late samples, I vote a +1 and if I have an equal number or no transition, I vote 0. So now, the output instead of being this wide thing at this lower clock rate is still just 2 bits with this lower clock rate.

Now I've thrown away some information here. I had information, I may have had L transitions there, and I'm treating it like its 1. But the point is that, whether I have L transitions or 1 transition, I get the same gain for that block or the same movement to that block. So now I'm not sensitive to the transition density. If I vote over a big enough block that I always get at least 1 transition, I'm completely insensitive to transition density. If I make my voting block big enough that it mostly happens, I'm fairly insensitive to transition density. So a lot of times, what you want to do is you want to look at the type of data you're going to be dealing with, and look at what's your longest CID event, consecutive identical digital event and vote in a way that either all the time or almost all the time, you have at least 1 transition over L samples, and if you do that, then you really remove that sensitivity to transition density. That's a huge, huge benefit of voting. I can't stress how important that is.

Now, once again, I said, oh, there's these two techniques, they don't have to be completely divorced, I can slow it down, I want to decimate by a lot for some reason, I could vote over, you know, if I wanted to slow down and decimate by a factor of 8, I could vote over 4, vote over 4, and add those two together if I didn't want to lower the gain so much. Once again, this does lower the gain. It lowers my maximum gain. My minimum gain is still the same because over those L samples, my summer might have spit out just a 1 or a -1, maybe I only had 1 transition, but when I sum my maximum gain, it go up to -L or +L through that block, and now it's down to -1 or +1. So my maximum gain is compressed, but my minimum gain stays about the same.

## 65. Comparison of Decimation

If I want to compare the impact of gain, through that block, because remember in my linear model, I've got this $K_D$, and I want to make, come up with a value for that. And so, what I do is, just run that same old simulation that I talked about a earlier, where I put sweep data through my Bang-Bang phase detector, but now I put it through the combination of the Bang-Bang phase detector and the summing unit, or the voting block, and see what comes out, and see what kind of gain I get out.

And so the blue line is the output of a sum by 4 and the red line is the output of a vote by 4. So the interesting thing is, once again, for the Bang-Bang CDR, I have this factor of 1/2, dependent upon transition density, but if I always get at least 1 transition over L, I could take that factor of a half out. Because I don't really care, as long as I have 1 transition over 4 bits, for this vote by 4, I always get something out of here. So I get to throw that out in terms of the gain.

And what you'll see here and what you'll find out, it kind of makes sense, is that the gain of this vote by 4 is about half, the small signal gain is about half of just summing, and it sort of makes sense, because out of my summer, since on average I have a transition density of about a half, I expect to get over the 4 + or – 2 out. If I'm early, I expect to get 2 earlies, if I'm late, I expect to get 2 lates, now out of the vote block, I would either get just a -1 or +1. So it sort of makes sense that I've got a gain reduction of about a half and if you run the simulation and you look at the curves, you actually plot the gain, you'll see that you actually do realize that.

## 66. Comparison of Summing Versus Voting

And this plot right here is just showing through a time step simulator, what I've done is I've taken the output decimation block and I've just actually averaged 20 consecutive samples together just to remove a little bit of the noise, and it shows you what happens. When I come out of the voting, once again, I am really so much quieter compared to the coming out of just summing those values. So what happened right here was, I ended up with a little period here with a lot of Nyquist data, just in my data stream. Since I was a tad early, I started jamming out bunches of big negative numbers. And then over here, all these big excursions are going to ripple through, and cause through that proportional path, they're going to cause big phase jumps in my output, it's going to cause noise, by voting we quiet it way down, and once again, one of the things we're worried about, one of the things we liked about that linear phase detector was the fact that it was a lot quieter as we got conversions, and here's another way that we can help quiet down the Bang-Bang CDR.

## 67. DPLL Structure

Now we have described every block in this Bang-Bang CDR. And we have values for everything here. My frequency path gets divided by this extra L as we described since I'm only adding it in occasionally and for voting by 4, we have a value of about a half for this $K_D$.

## 68. Latency ($Z^{-N}$)

Now, latency, we haven't talked about this yet, we're going to look at this more in the design example, but a major concern in the DPLL CDR is the latency. And in any loop, in any kind of PLL, anything like that, you worry about latency, because latency affects your stability. It takes you too long to get around the loop, you can't have very much gain because if I take a huge step, and it takes me a long time to realize I've taken a big step, the I take lots of big steps in one direction before I go whoa! whoa! I was going that way, I didn't realize that. That's what latency is, it's when you don't know how you're reacting to what you're doing. That's what latency causes. So it limits the gain you could have in your CDR loop. If you limit the gain, it limits your bandwidth, limits your frequency tracking capability, everything gets limited.

Once again, you want to design this thing to have as low of a latency as possible, and we'll take a look a little bit later in the designing example, how exactly it affects the system.

## 69. Summary

Summary up to this point is, we've developed a small signal model, we've decided, hopefully understand now how all the parameters impact the phase, the output sampling phase, because in the end, that's what we care about.

How does that sampling phase move with respect to phase errors and what can I do in this DPLL architecture to affect that?

## 70. CDR Example

Here's my first designing example and I don't know if we're going to have time to go through both of these, we certainly getting close to running out of time here, but the first design example. We're going to assume that we have a baud rate of 5Gbps, you just have to pick something, and we'll assume that I'm in that position where there's not spread spectrum so I want to achieve roughly + or – a thousand PPM of jitter tracking tolerance. I'd like to have a frequency step that's not too small and not too big, so we'll pick something around 10 PPM, once again, I'm going to have some resolution with my frequency register and whatever

error I have because of that quantization, I'll make up for it in my proportional path.

I want to have something that's fairly small there, otherwise my proportional path will be working too hard to compensate for the miss I make in my frequency register, 10 PPM is well within reasonable range there and you can have much higher values that that, 50, 60 PPM, still are typically fine.

My digital phase converter, I said hey, 5 bits 32 steps per UI was a reasonable value and the phase integrator, we allude at using these 8 bits, you may end up wanting to use more, you may want to use less, but 8 bits again, a reasonable place to start. That would give me 3 dither bits, 3 sub-resolution bits, and so my $K_{DPC}$ if you remember, is $1/2^8$ now, or .0009UI per step. And we're going to need that value when we do the design, pick the other parameters, and we're going to decimate by voting over 4, because we say, about 8b/10b coded data, almost all 8b/10b words, it's very hard to go 4 bits without having a transition. If you look at the code space, almost all the words will have a transition within inside a 4UI words. There are a few words, where that's not true, but typically those are far and few between.

That's a reasonable value, so that I'm not throwing away too much information, but I'm still getting rid of my sensitivity to transition density.

Now, we'll start with an assumption of about 20UI of latency around the loop, so that gives me some delay for the analog and then a handful of clock cycles through the digital. Because, once again, a clock cycle now, is 4UI sometimes I think it might take me 5 pipe stages to get around the loop counting the analog delay.

## 71. Max PPM Tolerance

The maximum PPM tolerance I wanted to hit was about 1000PPM, and my CDR clock cycle is 4UI because I'm voting by 4. So my frequency integrator needs to supply .001UI/UI or .004UI per 4UI clocks cycle. So that's why you need to shove in, while fortunately, hey, my phase integrator step is about .0039UI which is very close to the 0.004 I want to dump in. That means I need to dump in, to get a thousand PPM, I need to dump in a 1 for every 4UI clock cycles. That makes sense, because earlier when I was running at the baud rate at every UI, I came out with about 3900 PPM when I was dumping it in. Now since I am only dumping it in every $4^{th}$ UI, I get about a fourth of that.

I need to put in about a + or –1 per cycle to get + or - a thousand PPM, so the value of M now is 1. So I only have 1 bit, only the top bit is going to go to the phase integrator directly. The rest of the bits are going to go down to my $\Delta\Sigma$ to modulator. And that upper bit is either going to be a 0 or a -1. So I never even send a +1, and the way I get close to a +1, is once again, to put in those $D_f$ bits, be very close to 1. So then I'll dump in a 1 almost every cycle.

## 72. Min PPM Step Size for $D_f$

My minimum PPM step size, I want to have around 10PPM, so that means that I want to move the sampling phase by about .00004UI/cycle, so phase integrator steps .0039, so if I divide those 2 numbers, I see that I should move my phase integrator by 1 step every 97.5 cycles, once again, we don't like those kind of numbers, we like powers of 2, so we'll round that up to 128 and we'll have 7 bits. So I'll have 7 bits in that $D_f$, and that will allow me to move a, as little as $1/128^{th}$ of a UI per UI, or per 4UI. So the total width of the frequency integrator is 8 bits.

## 73. Final Frequency Register Results

I've got 8 bits of my frequency integrator, my maximum PPM offset that I can track isn't quite to a

thousand. If you worked through the math, you get 972, but you know what, hey, that's good enough for government work, as people like to say. That's close enough, it's not going to be like someone going to say, "Oh my gosh, you're at 999 instead of a thousand, you want to get roughly to a thousand. And you can see, I actually have a tiny bit more range than negative side since I only get to the positive side through the dither bits. And my minimum PPM resolution, because instead of dividing by 97.5, I divide by 128, it's a little less than 10, it's 7.6PPM.

Good I'm hitting my design goals and we're seeing how I figure out what values I need.

## 74. Pull-In Range

One of the things we haven't talked about yet is the pull-in range. That's how fast do I move, how fast can the proportional path move in terms of PPM? Because the pull-in range, once again remember, the frequency loop is there to compensate for the PPM offset, but when I start out, it moves very slowly. When I want to grab that signal as it's whizzing by, my fast part of my loop is my proportional path, so I want to understand what kind of PPM offset I can roughly keep up with. And I figure that out, by once again, seeing how fast can I move my phase integrator per UI through the proportional path? It's the same kind of thing we did to the frequency path, but now we're going through the proportional path. We're going to assume that I'm always late, or I'm always early because I'm trying to catch up to something, either it's going behind me or getting ahead of me, so I'm trying to catch up with my proportional path so I'm running really hard.

So, if you go through the math, and I've got a resolution of 1/256, and I only get to move my phase every 4UI, so my proportional path can move roughly 1/1024UI per UI, assuming that the value of phug is 1, so that's 976.6PPM. That's the same as what we had for our frequency register. So that tells me that my phase register is about as strong as my frequency register. Now that assumes, that we have a transition every 4UI and that every one of those is either early or late. As I'm pulling in, I'm behind, I'm always early, I'm always late, this is the maximum that I can sort of track. And so, this is not how you want to run your loop, but it tells you how it could pull-in, how it could lock. And so, if that's too low, then you may want to do something at the beginning like suppose in our design we came up with a pull-in range of only 100PPM, we decide, you know what, that's just not enough. What I could do is, as I bring the loop up, I could increase that value of phug initially, to increase that pull-in range. And then once again, once I get some idea of lock based upon looking at the data, seeing commas in the data, or whatever you want to do, I can lower that down to a nice quiet value.

## 75. DPLL CDR Small Signal Model

If I take the values that we just calculated that we thought we liked, and I shove them into my small signal model, and for analysis I assume that my input jitter is Gaussian with a $\sigma$ of about .03UI, I just need to pick something, and shove all those values and I get an equation for my open loop gain.

## 76. Using the Linear Model

Then I can use my linear model to see how the dynamics of the system behave. Because what we've said so far is, like look, I want to keep up with this, I want to track this kind of frequency, but I don't even know if the loop works. So that's what I'm trying to see here, so my open loop model is basically, you know, just looking at the small signal model and evaluating it. So I can look at the transfer function which is given by the sampling output over the input phase, and I can come up with an equation for jitter tolerance.

This one takes a little bit of thinking, first of all, jitter tolerance for those of you who aren't CDR folks means, how much sinusoidal jitter - for a given frequency of sinusoidal jitter, what's the maximum amplitude of that jitter that I could put to the input of my receiver before I break it, before I get below a bit error rate target?

So, you do a test on your CDR with test equipment, you drive in an input signal and you add sinusoidal jitter. You keep increasing the amplitude until you start making bit errors in your output and that's your tolerance to that amplitude of jitter at that frequency. And they can trace out a curve of amplitude vs. frequency and it shows, really how much jitter you can accommodate at a given frequency.

This is a very important parameter in a CDR. And this equation defines it here, so what you're saying is, really the output, the thing I'm looking for is the amount of input amplitude jitter that gives me a given amount of phase error.

## 77. Transfer Function

If I plot my transfer function for the loop gain that we just derived, I see that, oh my gosh! You know I thought I had it all done, and now I look at it and I'm getting a bunch of peaking in my CDR. And peaking is bad, that means that I'm actually increasing the amount of jitter at those frequencies. So, my linear model tells me that I've got something wrong here, I need to fix it.

## 78. Study Peaking as a Function of Phug and Frug

Things we haven't talked about yet, the thing that's going to control the peaking, control the bandwidth, control all the stuff, are the gains through the loop, gain through the proportional path and the gain through the integral path.

First let's see what happens. On this curve right here, what I've done is, I have kept frug at 1, and I have increased and decreased phug. If I increase it, it gets worse as expected. We've got peaking which means we've got too much gain in the loop, I increase the gain, it got worse, that makes sense, the bandwidth got wider and got more peaking.

Now when I decrease phug, gosh, darn it, the peaking went up again. That doesn't make any sense does it? Why would the peaking go up if lower the gain in the loop? The reason that happened is because now I've lowered the gain of the proportional path to the point where it's interacting with the frequency path.

In other words I need to keep the gain difference, just like in a PLL, if my poles get too close together, I have trouble. The same thing is happening here. If I lower the gain of that phase and I don't lower the gain of the frequency path, I've got the gain, it's too close together, they'll start fighting with each other and I don't want that to happen.

So I can't do anything with the proportional path to help myself, now if I keep phug at 1, and I start lowering the frequency path gain, so I've plotted here .5, .25, and .125, wow, now I'm getting what I want. So the interesting thing is, once again, the proportional path changed my bandwidth, changing the frequency path doesn't really change the bandwidth but it's helped calm the loop down.

## 79. Integral Path Gain Reduction

Integral gain path reduction, once again the integral path gain is $frug/2^{Df}/L$. I can't make a fractional number, fractional multiply in integer math, so my two choices are, either add bits to the bottom of the frequency register, increase $D_f$, or I can increase my decimation rate just through the frequency path. So if I increase the decimation rate, decimate over 16, vote over 16, instead of voting over 4, that reduces the gain by 4, because L was 4, and if I increase L to 16, then that gives me a reduction of 4 in that gain, which is what I'm looking for.

So what that means is that, the output of my frequency register can only change every 16UI, but it still gets

added to the phase integrator every 4UI. So those things are, you need to keep that in mind.

So I haven't changed how often I'm adding it in to the phase integrator, I'm changing how often I can update it. I'm not changing my PPM tracking range at all by doing this, but I'm changing how fast I can slew my frequency integrator.

Now the nice thing is, my frequency integrator was 8 bits at 1.25 GHz, now its 8 bits at 312.5MHz. My digital guy is very happy when I tell him he gets to design the slower integrator instead of the faster integrator.

And I've sort of reduced my minimum PPM resolution, really my minimum rate of tracking down to about 2 PPM.

## 80. Jitter Tolerance with Reduced Integral Path Gain

Now, if I go through and look at the jitter tolerance, with respect to this reduced path gain, what I see is that it looks reasonable, and I'm getting a little bit of dipping right here that I don't like, it looks a little ringy out here, and now I have to understand is, are those real, is that real behavior, this is a model or an artifact of trying to model this highly nonlinear system with this linear model.

## 81. Impact of Latency

And, we'll look at that in the time step simulator later in a minute, I know I'm running late here, if anyone has to leave, please just get up and go, you're not going to hurt my feelings.

The impact of latency, so we've talked about latency with respect to gain, so we've lowered the gain in the loop to bring it in into nice behavior, now if I can't hit my latency target and I start adding more latency to the loop, not because I want to, but because I just can't implement it otherwise, and my latency goes from 20UI to 30UI to 40UI. What you see is you get this big dip and jitter tolerance which corresponds to this huge peaking in my frequency response. But now we sort of understand how to deal with that peaking.

So if I have more latency in the loop, what would I do? I would lower both my proportional and my frequency path gains down together hand in hand, and that will lower the gain of both paths and that will bring that down, but when I lower the phug value to bring that down, what's going to happen to my bandwidth? It's going to be reduced, it's going to pull-in, because we saw that already when I changed the phug value that changes the bandwidth. For a given amount of latency, I'm going to limit the bandwidth of the loop.

## 82. Next Step in Design Process

Just quickly I want to go through, hey, talking about this linear model, and we came up with this design, it looks like it works reasonably well.

## 83. Simple Time Step Simulator

So what you want to do then, is build a time-step simulator. Eventually we want to build something that's very very accurate where model every analog circuit imperfection and you have this massive thing, but when you're first starting out, you start with something very simple, and it looks just like this. Just program this right into C or Matlab or whatever you want to do.

You take in some phase noise, take the sign of that phase noise, there's my Bang-Bang phase error decision, I

vote by 4, vote by 16, go through the phug path, frug path, adds those together, go through this integrator, go through the delay block, add that back in. So I've got a close loop simulation of that linear system that I just designed. But this is nonlinear now, because I've got the actual nonlinearity in here, I've got the voting in there. It's not just a gain factor, it's the actual voting. So now I realize that system with a simple simulation.

**84. Time Step Results**

And the first thing I do is basically I put no PPM offset in and I drive in a noisy signal, I see how well my CDR tracks.

So the input phase is this red signal right here, very noisy, Gaussian noise, jitter with a σ of .03 and the CDR output is in blue. So what I can see is even though my jitter, I'm jittering, say .2UI peak to peak, the output of my CDR is only jittering maybe .05UI peak to peak, something I can live with.

This behavior looks pretty good. I'm pretty happy about that. So it shows me that, hey, my CDR bandwidth isn't too wide, I'm not reacting too much to that incoming jitter.

**85. Time Step Results, Small Signal SJ (.1UIp-p @ 1.5MHz)**

I'm going to talk a little bit about jitter tolerance. So now I put in, on top of that Gaussian noise, I put in sinusoidal jitter. So now my jitter is moving up and down, up and down, on top of the sort of instantaneous Gaussian noise, and the output, if you could just look over here, curves on the right vs. left, basically I've removed the noise from the red signal so you can see the baseline jitter without the Gaussian on it, to see if I'm tracking that jitter. So that red right there, is this choppy red with the Gaussian noise removed so you just see the jitter.

The blue is the output of this noise simulation and I just plotted on top, the sinusoidal jitter to show the tracking. And in the green, is basically the error. Once again, I put in this .1UI at 1.5MHz and I'm tracking it very nicely, no problem keeping up.

**86. Time Step Results, Larger SJ (.1UIp-p @ 1.5MHz)**

I increase the amplitude, because I want to understand how I break my CDR, I need to increase my amplitude until I break it, now I turn it up more, and I run the simulation. And once again, now I'm up to 1UI peak to peak and I can see like, yeah, I'm starting to have a little bit of trouble tracking. You can see the drift in the phase error are up and down but still a very very acceptable range. I'm not consuming very much of UI through the jitter that I'm not tracking.

**87. Time Step Results, Small Signal SJ (.1UIp-p @ 1.5MHz)**

And if I turn it up to 2UI peak to peak, boom, I see that I've broken the CDR. I can't keep up. The phase is going up, I just can't keep up, I'm slewing just as fast as I can in my simulator and I just can't keep up.

So that's what it looks like when I've broken and my error term is just humongous now, this green signal, you can see the sinusoid in it, it's just horrible.

So I know I'm somewhere between 1 and 2UI peak to peak. What you would do eventually is, just like you would do in the lab, is turn the amplitudes up and down until you converge to where you got to, maybe some tolerance limit, like maybe I'll say something like, I'm going to allow myself something like .2UI peak to peak of phase error due to the sinusoidal jitter and you'll adjust the amplitude until you get to that limit.

## 88. Time Step Results, 500 PPM Offset

Last thing that I would do with a phase step simulator to understand if it's well designed, is see how my frequency register converges in, because we've lowered the gain of that thing and I want to make sure that, hey, it behaves nicely. Here I run a simulation, a time step simulation, and I'm just looking at the frequency register value. So I start my frequency register at 0, I put in a 500 PPM offset into the incoming jitter, and I see, does my frequency register, first of all, go to the right value, and it does, it slews right into 500, and that vertical scale I get that by using all those numbers that we calculated. How many PPM per bit in the frequency register, I just get numbers out of there, but I know how to convert those into PPM, because during the process of the design, I had those numbers in mind. I said, oh I designed this frequency register that's got say, 2 PPM per bit or something like that.

So then, hey, if I've got 2 PPM per bit, I've got a value of 250 in there, that's 500 PPM. Look at that frequency register and directly understand the frequency offset between the incoming signal and my local clock. And that's a beautiful thing about a DPLL, once again. For an analog CDR, I can maybe try to look at, measure, somehow sample the value on the capacitor in my PLL and try to convert that into some PPM offset, but in reality, with the DPLL, I can tell you exactly what the clock difference is between my local clock and the far end clock. That's a really cool thing.

Another thing I can do is, I can force a value into that frequency register. I can force 500 PPM into there, I can write that number into there and watch it come back into 0. So I can play all kinds of games say, like in an AT kind of environment. Shove a value in there, make sure my CDR pulls in, lots of cool stuff.

## 89. Spread Spectrum Clocking (SSC)

I have one more example, but I'm so far over time, that I probably shouldn't walk through that. It was a spread spectrum example and I just want to say one thing here.

## 90. SSC Example

I added a couple of parameters, increased tolerance and I'm worried about the slew rate of the frequency register and hopefully I've got enough words on these slides that you can follow through.

I go through sort of a similar kind of design.

## 91. Time Step Results for SSC with 200PPM

And here I'm showing, I just wanted to say what this slide is, so in a spread spectrum system, what you do is you purposely slew the frequency of the clocks, and you do that to reduce the EMI. And this is typically done in cabled application because any common mode noise on your differential signal going out, the return path is the shield of that cable, so that's just a radiator. And so to reduce the EMI and really sort of widen it, you spread it out in frequency so you don't have a big tone, you slew the clock back and forth. So this is showing the frequency register for this example tracking the incoming clock. So the incoming clock is slewing, that's the red signal and the blue signal right here is the frequency register. And because I'm trying to track a moving frequency with something that only has 0 static phase error for a fixed frequency, I'm going to end up with a static frequency error, static phase error, due to the fact that I'm underdetermined. A slewing frequency is a third-order system, I've got a second-order DPLL. So you see that in the phase error. As the frequency's falling, I end up with a negative phase error, I'm always a little bit behind. Then over here, I end up with a positive phase error when I'm behind because in 1 case, it's running away from me negative, in another case it's running away from me positive. So one case I get a negative static phase error, another case I get a positive static phase error.

When you do this type of design, you will see this kind of behavior, it's okay, it has to be there, it makes sense.

## 92. Jitter Tolerance

And finally, the last thing I did was, just do the jitter tolerance of the simulator vs. the linearized model. And so the linearized model remember, had this dip in it, and had a little of oscillation out here, and what we see is that when I run my time step model I don't really see that behavior. Why would I not see that behavior? The reason is jitter tolerance is a large signal event. I'm trying to push that CDR really, really hard. I'm not talking about teeny, teeny, little phase errors that my linearized model is expecting, I'm making big errors. Now the gain of that Bang-Bang phase detector, we used the gain down near 0, but that gain falls off as I increase the workload of the CDR, so my gain is a little bit lower. The other thing that happens in that linearized model is that in some sense my jitter tolerance is largely determined, once again, by the proportional path. The frequency path isn't really helping. It's considered too slow, but in the large signal case, it actually slews fast enough, that is actually helping me track some of the jitter. What I end up with is, little lower gain through my proportional path but more help through my frequency path.

What I end up with is, you know I get rid of this ringy dingy stuff, and this dip, and actually get a little bit more bandwidth out of this CDR. The other thing that I get sort of negatively is, I do get sort of a little less high frequency jitter tolerance than I expect from the linear model, because once again, the linear model is assuming very small things, although I'm taking bigger steps, I have bigger quantization errors, I fall behind, I overstep, I understep, all those kinds of things really aren't totally quantified in that linear model. So I'd expect to see this type of behavior, little bit wider bandwidth, little bit higher bandwidth for my jitter tolerance, a little bit, sort of looks like lower gain through my proportional path, little less peaking, and a little less high frequency jitter tolerance, a little more self noise than I had modeled in the original system.

So I thank you guys, I'm sorry I kept you so long.

## 93. SSC Design Summary

This is just a summary of that.

## 94. Conclusion

You know, that once again, hey, we covered a lot of material here today, and gosh, I tried to shrink it down, every time I shrunk it down, it grew back, you know, this ugly thing, you tried to squish it out, and it keeps coming back.

I couldn't figure out what I could really remove without causing some loss of understanding of the material. So I really needed more than an hour and a half to cover this material and I took it.

So I hope everyone got out of this talk, what they came to get out of it.

You have my email address on the first slide, please feel free to send me questions, or ask me things. I'm going to be at the conference all week, come up to me, do whatever you need to do. Rather than do open questions right now, everyone can leave, anyone who wants to ask me a question, come and ask me a question.

Feel free to email me, bug me at the conference, do what you need to do, I want you guys to understand this material.

Thank you very much.