Modeling the Behavior of an RS-latch in Haskell

Remco Huizenga
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
g.r.huizenga@student.utwente.nl

ABSTRACT
This paper briefly discusses multiple clock domains and the occurrence of metastable behaviour when one tries to synchronize these domains. It takes an existing model of an RS-latch, a the basic memory element used in synchronization, in VHDL and transforms this into a model in Haskell. A Haskell implementation is given that models the normal and metastable behavior of an RS-latch. The paper also shows how this model can be used in the modeling of a D-type latch.

Keywords
Metastability, Hardware Description Language, Haskell, Synchronization, RS-NAND latch, D-type latch.

1. INTRODUCTION
In hardware design one usually has to deal with the synchronization of subsystems operating on different clocks. The synchronizing of multiple clock domains is done by designing a synchronization circuit. These synchronization circuits can get very complex but memory elements are the core of every circuit. When one tries to synchronize two domains with different clocks an effect called metastability can occur. It is very important to be able to simulate this effect accurately since the consequences of metastability can be very grave.

The target language of the model given in this paper is Haskell. The reason Haskell is used is CλaSH. The target language of the model given in this paper is Haskell. A Haskell implementation is given that models the normal and metastable behavior of an RS-latch. The paper also shows how this model can be used in the modeling of a D-type latch.

2. BACKGROUND
In this section the reader is made familiar with the most important aspects and terminology of multiple clock domains, synchronization, metastability and the RS-latch before beginning with the actual model.

2.1 Multiple clock domains
A clock domain is a section of logic where all elements share the same clock. In designs consisting of a multitude of those sections, we talk about multiple clock domains. Most modern digital systems are implemented as SoCs (System on Chips) [8]. There are quite some issues involved in designing these SoCs. This paper will not go into detail about the issues involved in the design of SoCs, this is outside of the scope of this paper. A popular method for dealing with these issues is the deployment of GALS (globally asynchronous, locally synchronous) systems. A GALS system is build up of complex digital blocks that operate synchronously, i.e. have their own clock domain. But communication between the blocks is not synchronized. There are various approaches to designing GALS, but this also falls outside of the scope of this paper. The main difficulty with designing GALS, is designing reliable GALS interfaces to handle metastability. Metastability is an effect that can occur with the synchronization of clock domains, in digital circuits with asynchronous inputs, and it is this effect that we are interested in. The next section will explain in more detail what metastability is and how and where it can occur. One might wonder why we want to simulate this effect. Why are we not designing the system in such a way, with the help of constructions in the development environment or language, that the effect cannot occur? The reason for this is the fact that we can never guarantee that metastability will not occur [3].

2.2 Metastability and the RS-Latch
As stated before, digital circuits with asynchronous inputs are susceptible to metastability. Under certain cir-
cumstances, when the inputs have critical timing combinations, the system may fail due to this effect. The circuits lingers indefinitely between two stable states before it resolves to one or the other stable state, depending on the conditions that caused the failure [6]. The consequences of metastability can be as grave as the system completely crashing [3].

Synchronization between clock domains is usually done with flip-flops. Therefore this paper will concentrate on modeling metastable behavior in an RS-latch with crossed NAND-gates (See Figure 1), a basic memory element used in different kinds of flip-flops. This model can then be used in more complicated structures, for example a D-type latch, as will be described later in this paper. The normal behavior of the RS-latch is shown in the truth table below.

![Figure 1. RS-NAND latch](image)

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>Q'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>last Q</td>
<td>last Q</td>
</tr>
</tbody>
</table>

Table 1. Truth table of an RS-NAND latch

This shows us that, because the latch is active low, the state doesn’t change when the inputs are both high.

3. MODELING THE RS-LATCH

This section will first introduce the equation the model is based upon. According to [9] there exists an exponential relation between the minimal time the bistable operates in metastability \( tr \) and the time interval \( \Delta T(tr) \) in the excitation space within the critical input window. The term bistable refers to an electronic circuit that has 2 stable states at any given time so that it is possible to switch between them. If it then also holds that the setup time and hold time (see the next section) are equal, the bistable is said to be symmetric. The exponential relation is given by:

\[
\Delta T(tr) = \delta e^{tpn/t} e^{-tr/\tau} \tag{1}
\]

In this formula, \( \delta \) is the width of the critical input window, this is the time between transitions of the \( R \) and \( S \) signals where metastability can occur. \( tpn \) is the normal propagation time outside the critical input window, i.e. the time it takes for the output signal to change from one stable state to the other stable state \([4]\). \( \tau \) is a parameter unique to each bistable. As is also stated in [1], the value of \( \tau \) is not usually included in the data sheet but can be determined either experimentally or by electric simulation.

3.1 Normal behavior

This section will give an implementation in Haskell that will model the normal behavior of the RS-latch. What we want from the model is to provide us with the output given certain input values and the time it took to reach that state. This is easily done and follows directly from Table 1.

\[
\begin{align*}
type In & = (\text{Int}, \text{Int}) \\
type Out & = (\text{Int}, \text{Int}) \\
type State & = (\text{Int}, \text{Int}) \\
type Time & = \text{Double} \\
type TPN & = \text{Double} \\
rsLatch :: \text{In} \to \text{TPN} \to \text{State} \to (\text{Time}, \text{Out})
\end{align*}
\]

As can be seen in the code listing above, a few new type synonyms have been introduced to distinguish between input, output and the current state. Type synonyms have also been made for Time, the time it takes for the latch to reach its final, stable output and for \( TPN \), the constant for the normal propagation time. For normal behavior of the latch, the time it takes to reach a stable output is always equal to the normal propagation time. The values of the output signals \( Q \) and \( Q' \) depends only on the input signals \( R \) and \( S \) so we can simply use pattern-matching on the input signals. When both input signals are high, the state of the latch doesn’t change and the output stays the same. This is why the variable \( State \) was introduced. It holds the state of the latch, i.e. the signals \( Q \) and \( Q' \), before the new input arrived. In the tuple \( In \) the first \( \text{Int} \) represents the \( R \) signal, the second the \( S \) signal. In the tuples \( Out \) and \( State \) the first \( \text{Int} \) represents the signal \( Q \) and the second the signal \( Q' \).

3.2 Metastable behavior

As is the case in [1], this paper distinguishes the following four situations causing metastability in the RS-latch:

1. Setup time violation. The setup time is the minimum time the data should be stable before the clock event. One speaks of a setup time violation when the time interval \( q \) between the rise of the \( S \) signal and the rise of the \( R \) signal is less than the setup time, \( T_{setup} \).

2. Hold time violation. The hold time is the minimum time the data should be stable after the clock event. A violation of the hold time occurs when the time interval \( q \) between the rise of the \( R \) signal and the rise of the \( S \) signal is less than the hold time, \( T_{hold} \).

3. S runt pulse. Pulses with a width smaller than \( T_{rump} \) get filtered out by the latch itself and by pulses wider than \( T_{rump} \) the latch operates normally. If the width of the pulse is between those two values it is called a runt pulse and can cause metastability.

4. R runt pulse. This is exactly the same as the previous point but now on the \( R \) input.

3.2.1 Setup and Hold time violation

Because the RS-latch is a symmetric bistable the setup time and hold time are equal. This also means that the equation [1] derives from (1) for the time it takes the latch to reach a stable state after entering metastability caused by violation of the setup time or hold time, is equal in both cases:

\[
tr = tpn + \tau \log \left( \frac{T_{setup}}{g} \right) \tag{2}
\]
The setup time violation means that the $R$ signal is not stable and the time between the rise of the $R$ signal and the last rise of the $S$ signal is less than the setup time $T_{setup}$. The formula for $tr$ gives us the time it takes the latch to reach a stable state when metastability ends naturally. But there are two more ways the metastable state can end and we have to account for those as well. Metastable state can also come to an end if the $R$ or $S$ signal drops. The time it takes then to reach a stable state is the time it takes from entering the metastable state until the signal dropped plus the normal propagation time.

The next thing to realize is the fact that from equation (2) one can see that the time it takes to reach a stable state will take longer the smaller the time $g$. In fact, if $g$ approaches 0, $tr$ gets infinitely long. This is unrealistic and can be avoided in the model by deviating a thousandth of the setup or hold time. This is accounted for in the code by the $if$ -- $then$ -- $else$ statement. This checks if $g$ is less than 0.001 * $T_{setup}$ and if so, makes this the new value. Otherwise just keep it as g.

All this means that a simple Integer for the input and output signals does not suffice anymore. In the case of setup or hold time violation a way to indicate whether output signals does not suffice anymore. In the case of the model now also needs to know the values for $T_{setup}$, $SHTime$ (Setup or Hold time) and $TPN$. Then new model is shown in the code listing below:

```haskell
rslatch :: In -> Drop -> SHtime -> Tau -> TPN -> State -> (Time, Out)
```

### 3.2.2 Runt pulse violation

Modeling metastability caused by runt pulse violation is the most complicated case. The equation [1] deduced from equation 1 to calculate the time it takes the latch to resolve to a stable state after entering metastability caused by a runt pulse is the following:

$$ tr = tp + \tau \log\left( \frac{T_{min} - TW_{fil}}{2|TW - TW_{fil} - T_{min}|} \right) $$

In addition to the parameters introduced in the previous sections, the model now also needs to know the values for $TW$, $TW_{fil}$ and $T_{min}$. To make the code more structured and readable, a new data type has been used to hold all the characteristics of the latch. In code:

```haskell
data Latch = Latch SHTime Tau TWfil TWmin TPN deriving (Show, Eq, Ord)
```

Everything to the right of the constructor `Latch` are just type synonyms for `Double`. The existing data type for `Signal` has been expanded with an option `RP Double` to indicate the occurrence of a runt pulse (RP) followed by the length of the pulse. The following code snippet only shows the code for metastable behavior caused by an S-runt pulse. The code for violation by R-runt pulse is almost the same, apart from the final stable state reached, due to the bistable being symmetric. For completeness the entire code for the finished model can be found in appendix A.

```haskell
rslatch (_, RP tw) d (Latch _ tau twfil twmin tpn) _
```
from either side. The case is checked for by the metastability caused by setup or hold time violation and a different final state is reached. This was also the case for before the signal drops and which one of the signals drops, of the signals can drop and depending on the time passed solve to a stable state. But before this happens either one metastability ends naturally, or

Triing to equation 3.

This.

The.

if

where

a = if (tw <= 0.5005*(twfil+twmin))
then (0.0005*(twfil+twmin))
else (tw - 0.5*(twfil+twmin))
b = if (tw >= 0.4995*(twfil+twmin))
then (0.0005*(twfil+twmin))
else (0.5*(twfil+twmin) - tw)

Dependent on the width of the runt pulse, there are 3 categories Tw can fall into:

1. tw < Twfil. In this case the pulse gets filtered out by the latch itself due to its inertial effect so nothing happens.
2. (Twfil+Twmin) < Tw < Twmin. Even though it causes metastability, the pulse is wide enough to cause the latch to change its state.
3. Twfil < Tw < (Twfil+Twmin). The pulse is still wide enough to cause metastability, but not wide enough to cause the latch to change its state.

These 3 possibilities are reflected by the model in the initial 3 “paths” the function can take. The next thing to take into account is the fact that, as was the case for setup or hold time violation, the time spend in metastability gets infinitely long as Tw approaches the value (Twfil+Twmin) from either side. The where clause in the code deals with this. The if-then-else statement checks if the value for Tw is within a thousandth of (Twfil+Twmin) and takes appropriate actions to ensure finite resolution time. If Tw falls between Twfil and Twmin, tr is calculated according to equation 3. a or b is used depending on the width of the pulse. If neither of the signals R or S drops and metastability ends naturally, tr is the time it takes to resolve to a stable state. But before this happens either one of the signals can drop and depending on the time passed before the signal drops and which one of the signals drops, a different final state is reached. This was also the case for metastability caused by setup or hold time violation and is checked for by the case-af statements in the code.

4. D-TYPE LATCH BASED ON THE RS-NAND LATCH

This section will show how one can construct a more complicated model using the model for the RS-latch developed in this paper. As an example, this paper will take the D-type latch constructed with the RS-NAND latch, see Figure 2. In this picture the two crossed NAND-gates the RS-latch is made up of can be easily recognized.

What follows is the code to describe the behavior of a D-type latch.

dlatch :: In -> Drop -> Latch -> State ->

\[
\begin{align*}
| (tw < twfil) &\quad = (0, \text{Low, High}) \\
| (tw => 0.5*(twfil+twmin)) &\quad = \text{let } tr = tpn + tau \times \log((0.5*(twmin-twfil)) / a) \\
\text{in case d of} &\quad (\text{R } t) \rightarrow (t + tpn, \text{Low, High}) \\
&\quad (S t) \rightarrow (t + tpn, \text{High, Low}) \\
&\quad (N) \rightarrow (tr, \text{High, Low}) \\
| (tw < 0.5*(twfil+twmin)) &\quad = \text{let } tr = tpn + tau \times \log((0.5*(twmin-twfil)) / b) \\
\text{in case d of} &\quad (R t) \rightarrow (t + tpn, \text{Low, High}) \\
&\quad (S t) \rightarrow (t + tpn, \text{High, Low}) \\
&\quad (N) \rightarrow (tr, \text{Low, High}) \\
\end{align*}
\]

\[
\text{where}
\begin{align*}
a &\quad = \text{if } (tw <= 0.5005*(twfil+twmin)) \\
&\quad \quad \text{then } (0.0005*(twfil+twmin)) \\
&\quad \quad \text{else } (tw - 0.5*(twfil+twmin)) \\
b &\quad = \text{if } (tw >= 0.4995*(twfil+twmin)) \\
&\quad \quad \text{then } (0.0005*(twfil+twmin)) \\
&\quad \quad \text{else } (0.5*(twfil+twmin) - tw)
\end{align*}
\]

The arguments given to the dlatch function are exactly the same as for the rslatch function where the Latch argument holds the characteristics of the RS-latch used. The first thing the function does is distinguish between normal behavior and metastable behavior. If either one of the input signals indicates it is metastable or a runt pulse, the second part of the function is calculated. The input signals are given straight to the rslatch function and the values for the time spend in a metastable state and the final state reached are used for the output of the dlatch function. This can be seen in the last two lines of function. The 2 added to the time part of the output of the rslatch function is the propagation time for the signals to reach the R and S inputs of the RS-latch. Here we assume the propagation time of a NOT-gate as well as a NAND-gate is 1 nanosecond. This means that the maximum delay, the D signal propagating through a NOT-gate and NAND-gate, is 2 nanoseconds.

The first part of the dlatch function describes the normal behavior. Again, the rslatch function is called, but this time the R and S input signals are calculated first by the following line of code:

\[
(r, s) = (\text{myNand c (myNot d), myNand c d})
\]

As can be seen from Figure 2, the R input of the RS-latch is the NAND of the negated D input and the C input. The S input is the NAND of the D and C input. This is reflected in the code. Two helper functions, myNand and myNot, have been written and do exactly what their respective names suggest. For brevity, the helper functions are only listed in the appendix. The reason why the build-in functions, like not, could not be used is because the functions have to be able to handle inputs of the data type Signal. Next, the rslatch function is called with the correct inputs for R and S. The last thing we have to account for now, is the fact that we do not add the 2 nanoseconds propagation time if the outputs do not change. This is checked for by checking if the time part of the output of the rslatch function is not 0. Recall that

Figure 2. D-type latch based on the RS-NAND latch
the output state does not change if the \( R \) and \( S \) inputs are both high.

5. CONCLUSION

In this paper the reader was introduced to the effect called metastability that can occur in hardware and the importance to be able to accurately model metastable behavior. In the next few sections an existing model of an RS-NAND latch in VHDL has been taken \( \text{citeacosta:modelinginvhdl} \) and this model has been used to incrementally build a model in Haskell that can accurately model the metastable behavior of an RS-NAND latch. Starting off with a simple model describing the normal behavior of the RS-NAND latch, it was then expanded to account for metastable behavior caused by setup or hold time violation and was finally expanded further to also model metastable behavior caused by \( R \)-runt or \( S \)-runt pulses. The final result can be seen in Appendix A. As was shown in section 4, this model can be used to describe more complex latches like a D-type latch. With this model now ready in Haskell, some more research has to be done to find a way to integrate this in \( C\alpha SH \).

6. REFERENCES

APPENDIX

A. FULL CODE FOR THE MODEL

This appendix holds the complete code for the model developed in this paper including the code for the D-type latch and the helper functions.

```haskell
data Latch = Latch SHtime Tau Twfil Twmin TPN
  deriving (Show, Eq, Ord)
data Signal = High | Low | MS Double | RP Double
data Drop = R Double | S Double | N
type In = (Signal, Signal)
type Out = (Signal, Signal)
type State = (Signal, Signal)
type Time = Double
type TPN = Double
type Tau = Double
type SHtime = Double
type Twfil = Double
type Twmin = Double

myNand :: Signal -> Signal -> Signal
myNand High High = Low
myNand Low Low = High
myNand Low High = High
myNand High Low = High
myNand _ a = a

myNot :: Signal -> Signal
myNot High = Low
myNot Low = High
myNot x = x

dlatch :: In -> Drop -> Latch -> State -> (Time, Out)
dlatch (d, c) drp latch state
  -- Normal behaviour
  | ((d==High || d==Low) && (c==High || c==Low)) = let
  | (t, o) = rslatch (r, s) drp latch state
  | (r, s) = (myNand c (myNot d), myNand c d)
in case t of
  | 0 -> (0, o)
  | otherwise -> (2 + t, o)
  -- Metastable behaviour
  | otherwise
  | let (t, o) = rslatch (d, c) drp latch state
  | in (2 + t, o)

rslatch :: In -> Drop -> Latch -> State -> (Time, Out)
rslatch (Low, Low) _ (Latch _ _ _ _ tpn) _
  = (tpn, (Low, High))
rslatch (High, Low) _ (Latch _ _ _ _ tpn) _
  = (tpn, (High, Low))
rslatch (Low, High) _ (Latch _ _ _ _ tpn) _
  = (tpn, (Low, High))
rslatch (High, High) _ state
  = (0, state)

-- Metastable by setup time violation
rslatch (MS g, _) d (Latch st tau _ _ tpn) _
  = let
  | a = if (g < (0.001*st)) then (0.001*st) else g
  | tr = tpn + tau*log(st/a)
in case d of
  | (R t) -> (t + tpn, (Low, High))
  | (S t) -> (t + tpn, (High, Low))
  | (N) -> (tr, (High, Low))

-- Metastable by hold time violation
rslatch (_, MS g) d (Latch ht tau _ _ tpn) _
  = let
  | a = if (g < (0.001*ht)) then (0.001*ht) else g
  | tr = tpn + tau*log(ht/a)
in case d of
  | (R t) -> (t + tpn, (Low, High))
  | (S t) -> (t + tpn, (High, Low))
  | (N) -> (tr, (High, Low))

-- Metastable by R runt pulse violation
rslatch (RP tw, _) d (Latch _ _ twnf twnm tpn) _
  | (tw < twnf) = (0, (High, Low))
  | (tw >= 0.5*twnm) = let tr = tpn + tau
  | *log((0.5*(twnm-twnf))/a)
in case d of
  | (R t) -> (t + tpn, (Low, High))
  | (S t) -> (t + tpn, (High, Low))
  | (N) -> (tr, (High, Low))

-- Metastable by S runt pulse violation
rslatch (_, RP tw) d (Latch _ _ twnf twnm tpn) _
  | (tw < twnf) = (0, (Low, High))
  | (tw >= 0.5*twnm) = let tr = tpn + tau
  | *log((0.5*(twnm-twnf))/b)
in case d of
  | (R t) -> (t + tpn, (Low, High))
  | (S t) -> (t + tpn, (High, Low))
  | (N) -> (tr, (High, Low))

where
  a = if (tw <= 0.5005*(twnm-twnf)) then 0.0005*(twnm-twnf)
  else (tw - 0.5)*(twnm-twnf)
  b = if (tw > 0.4995*(twnm-twnf)) then 0.0005*(twnm-twnf)
  else (0.5*(twnm-twnf) - tw)

-- Metastable by delay violation
rslatch (_, _, _ RP tw) d (Latch _ _ _ _ tpm) _
  | (tw < tpm) = (0, (Low, High))
  | (tw >= 0.5*tpm) = let tr = tpn + tau
  | *log((0.5*(tpm-tpm))/a)
in case d of
  | (R t) -> (t + tpn, (Low, High))
  | (S t) -> (t + tpn, (High, Low))
  | (N) -> (tr, (High, Low))

where
  a = if (tw <= 0.5005*(tpm-tpm)) then 0.0005*(tpm-tpm)
  else (tw - 0.5)*(tpm-tpm)
  b = if (tw > 0.4995*(tpm-tpm)) then 0.0005*(tpm-tpm)
  else (0.5*(tpm-tpm) - tw)

rslatch _ _ _ _ = error "Invalid input"
```