SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

# Systemarchitektur SS 2021

# Aufgabenblatt 6

Sie können Ihre Lösungen bis **Mittwoch, dem 02.06.2021, um 10:00 Uhr** im CMS abgeben.
Geben Sie auf Ihrer Lösung Ihr Tutorium sowie die Namen und Matrikelnummern aller Gruppenmitglieder an.

## Aufgabe 6.1: Speicherelemente

### RS-Latch

In der Vorlesung haben Sie den RS-Latch als erstes Speicherelement kennengelernt. Untersuchen Sie dessen Funktionsweise für unterschiedliche Eingabeszenarien, zum Beispiel

- einen Puls auf $/R$, oder
- einen Puls auf $/S$ sowie $/R$ gleichzeitig.

Gibt es Eingabeszenarien, die zu instabilen Zuständen führen, das heißt es gibt keinen stabilen Ausgangswert? Begründen Sie Ihre Antwort kurz.

### D-Latch

Ein D-Latch Baustein kann mithilfe eines RS-Latches realisiert werden. Gibt es eine Eingabe (aus $W$ und $D$), sodass das RS-Latch einen instabilen Zustand erreicht? Begründen Sie.

## Aufgabe 6.2: Taktpegel versus Taktflanken

Wir kennen verschiedene Speicherelemente: *taktpegel-gesteuerte* Latches und *taktflanken-gesteuerte* Flip-Flops. Wir haben in der Vorlesung behauptet, dass sich Flip-Flops besser für den Betrieb in Schaltwerken eignen als Latches, da ihr Verhalten einfacher vorhersagbar sei. Diese Aufgabe dient dazu diese Aussage zu validieren. Betrachten Sie den Zähler auf Folie 15 des 9. Foliensatzes, wobei clear $/C$ und load $/L$ jeweils permanent 1 sind.

- Nehmen Sie zunächst vereinfachend an, dass die Propagationsverzögerung $t_{PQD}$ immer konstant sei. Wie verhält sich der Wert des Registers im zeitlichen Verlauf? Skizzieren Sie den Verlauf für unterschiedliche Uhr-Perioden, z.B. für Periode $t_{PQD}$, $2 \cdot t_{PQD}$, $3 \cdot t_{PQD}$, etc.
- Wie verhält sich die Schaltung zeitlich, wenn wir das Register durch ein $n$-Bit Latch ersetzen? Probieren Sie wieder unterschiedliche Uhr-Perioden aus.
- Obige Probleme könnte man durch eine sehr geschickte Wahl der Uhr-Periode noch lösen. Was passiert, wenn die Propagationsverzögerung durch den Schaltkreis, wie in unserem Fall, nicht konstant ist?

*Hinweis: Die Propagationsverzögerung $t_{Pab}$ bezeichnet die Zeit, die vergeht bis ein Signal von Position $a$ in der Schaltung zu Position $b$ propagiert wird. Diese Verzögerung hängt z.B. von technologischen Faktoren sowie dem Layout der Schaltung ab.*

## Aufgabe 6.3: Schaltwerke und endliche Automaten

In dieser Aufgabe möchten wir ein gewisses Muster in einer Eingabesequenz finden. Wir erhalten pro Zyklus ein Eingabebit und geben eine Ausgabe von 1, sobald wir das Muster $101$ in der Sequenz von Eingabebits erkennen.

(a) Zeichnen Sie einen Moore- sowie einen Mealy-Automaten, der dieses Muster erkennt.

(b) Verhalten sich Ihre beiden Automaten identisch oder unterscheiden sie sich? Wie viele Zustände benötigen Sie jeweils?

(c) Wir möchten nun beide Automaten durch je ein Schaltwerk in Hardware realisieren. Das Schaltwerk besitzt eine 1-bit Eingabe $i$ sowie eine 1-bit Ausgabe $o$ und soll das Muster $101$ erkennen. Wie viele Bits an Speicher benötigen Sie? Zeichnen Sie die resultierenden Schaltwerke. Vollziehen Sie das Verhalten der Schaltwerke für die Eingabesequenz $i^0 i^1 i^2 i^3 i^4 i^5 i^6 = 0010101$ nach.

## Aufgabe 6.4: Verilog-Tools

Für die folgenden Aufgaben sowie das Projekt benötigen Sie Werkzeuge zur Verilog-Synthese. Zur Synthese und Simulation von Verilog-Code verwenden wir *Icarus Verilog*[1] und zum Betrachten von generierten Waveforms *gtkwave*[2]. Beide Programme funktionieren prinzipiell unter Linux, Mac OS X, und Windows. Detaillierte Installationsanweisungen finden Sie im CMS unter "Zusatzmaterial".

Zur Synthese eines Top-Level Moduls $M$, dessen Definition inklusive aller Sub-Module sich auf Dateien `file1.v` bis `filen.v` verteilt, rufen Sie in der Kommandozeile

`iverilog -s` $M$ `-o sim file1.v ... filen.v`

auf. Um die Simulation zu starten führen Sie das generierte Binary `sim` aus. Je nach Testbench sehen Sie Ihre Ergebnisse auf der Kommandozeile oder finden eine generierte Waveform-Datei, welche Sie mit `gtkwave` anschauen können.

Eine gute und sehr ausführliche Einführung in Verilog mit vielen Beispielen bietet die Seite Asic-World[3]. Informationen zur Benutzung von Icarus Verilog finden Sie unter `http://iverilog.wikia.com/wiki/Getting_Started` und für GTKWave unter `http://gtkwave.sourceforge.net/gtkwave.pdf`.

Installieren Sie Icarus Verilog sowie GTKWave und machen Sie sich mit deren Verwendung vertraut. Nutzen Sie Ihre Übungen um Probleme bei der Installation der Programme auszuräumen.

## Aufgabe 6.5: Verilog: Zähler

Betrachten Sie den Zähler auf Folie 15 des 9. Foliensatzes, wobei clear $/C$, load $/L$ und $X$ jeweils Eingaben sind und $Y$ die Ausgabe ist. Sei $n = 32$ Bit.

Geben Sie eine Verilog-Beschreibung dieses Schaltwerkes an. Kapseln Sie den Inkrementierer in ein eigenes *Modul*.

Geben Sie ferner einen Testbench an, der die Funktionsweise von $/C$ und $/L$ demonstriert. Verwenden Sie Icarus Verilog zur Simulation.

[1] `http://iverilog.icarus.com`
[2] `http://gtkwave.sourceforge.net`
[3] `http://www.asic-world.com/verilog/veritut.html`

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

# System Architecture SS 2021

# Assignment 6

You may submit your solutions via the CMS until **10:00 a.m. on Wednesday, June 2, 2021**.
Please state on your solutions your tutorial, and the names and matriculation numbers of all team members.

## Problem 6.1: Memory Cells

### SR Latch

In the lecture, we discussed the SR latch as the first memory cell. Examine its behavior for different input scenarios, for example

- a pulse on $/R$, or
- pulses on $/S$ and $/R$ simultaneously.

Are there input scenarios that lead to unstable states, i.e., there is no stable output value? Justify your answer briefly.

### D Latch

A D latch can be implemented using an SR latch. Is there an input (for $W$ and $D$) such that the SR Latch reaches an unstable state? Justify your answer.

## Problem 6.2: Level Triggering vs. Edge Triggering

We have seen different memory cells: *level-triggered* latches and *edge-triggered* flip-flops. In the lecture, we claimed that flip-flops are better suited for sequential circuits than latches because their behavior is easier to predict. The goal of this exercise is to validate that claim. Consider the counter on slide 15 of the 9th slide deck; here, we assume that both clear $/C$ and load $/L$ are permanently 1.

- First, assume for simplicity that the propagation delay $t_{PQD}$ is always constant. How does the value of the register change over time? Sketch the progression for different clock periods, e.g., for the periods $t_{PQD}$, $2 \cdot t_{PQD}$, $3 \cdot t_{PQD}$, etc.
- How does the circuit behave over time if we replace the register with an $n$-bit latch? Again, try different clock periods.
- The problems above could be solved by a very clever choice of the clock period. What happens if the propagation delay through the circuit is, as in our case, not constant?

*Note: The propagation delay $t_{Pab}$ denotes the time that elapses until a signal is propagated from position $a$ in the circuit to position $b$. This delay depends, for example, on technological factors as well as the layout of the circuit.*

## Problem 6.3: Sequential Circuits and Finite State Machines

In this problem, we want to find a certain pattern in an input sequence. We get one input bit per cycle and produce an output of 1 as soon as we detect the pattern $101$ in the sequence of input bits.

(a) Draw both a Moore machine and a Mealy machine that recognize this pattern.

(b) Do your two machines behave identically or do they differ? How many states do you need in each case?

(c) We now want to realize both machines by sequential circuits in hardware. The sequential circuits have a 1-bit input $i$ and a 1-bit output $o$, and they shall recognize the pattern $101$. How many bits of memory do you need? Draw the resulting sequential circuits. Simulate the behavior of the sequential circuits for the input sequence $i^0 i^1 i^2 i^3 i^4 i^5 i^6 = 0010101$.

## Problem 6.4: Verilog Tools

For the following problems, as well as for the first project, you will need Verilog synthesis tools. For the synthesis and simulation of Verilog code, we will use *Icarus Verilog*[4], and for viewing generated waveforms, we will use *gtkwave*[5]. Both programs are available for Linux, macOS, and Windows. Detailed installation instructions can be found in the CMS under "Zusatzmaterial".

To synthesize a top-level module $M$, whose definition including all sub-modules is contained in the files `file1.v` to `filen.v`, run the following command on the command line

```
iverilog -s M -o sim file1.v ...  filen.v
```

To start the simulation run the generated binary `sim`. Depending on the testbench, you will see your results on the command line, or you will find a generated waveform file, which you can view with `gtkwave`.

A good and very detailed introduction to Verilog with many examples is provided by Asic-World[6]. For information on using Icarus Verilog, see `http://iverilog.wikia.com/wiki/Getting_Started` and for GTKWave, see `http://gtkwave.sourceforge.net/gtkwave.pdf`.

Install Icarus Verilog as well as GTKWave and familiarize yourself with their use. Use the tutorials and office hours to troubleshoot installation problems.

## Problem 6.5: Verilog: Counter

Consider the counter on slide 15 of the 9th slide deck, where clear $/C$, load $/L$, and $X$ are inputs, and $Y$ is the output. Let $n = 32$ bits.

Develop a Verilog description of this sequential circuit. Encapsulate the incrementer in a separate *module*.

Further, implement a testbench that demonstrates the operation of $/C$ and $/L$. Use Icarus Verilog for the simulation.

[4] `http://iverilog.icarus.com`
[5] `http://gtkwave.sourceforge.net`
[6] `http://www.asic-world.com/verilog/veritut.html`