

Laborprojekt – Versuch 7

Lernziele

Die Studierenden können die Funktionsweise und den Aufbau eines Halb- und Volladdierers anhand der in der Vorlesung vorgestellten Schaltung in VHDL umsetzen und erklären. Des Weiteren sollen die zuvor erlernten Kenntnisse über Generics angewendet werden. Zusätzlich lernen die Studenten, Entities von generischer Breite mithilfe des generate-Statement aus Komponenten zu erstellen.

Einleitung

Das Herz eines Rechners sind die arithmetischen Schaltungen. Jeder Computer muss die vier Grundrechenarten Addition, Subtraktion, Multiplikation und Division beherrschen. In der Regel werden dabei Multiplikation und Division mithilfe der Addition algorithmisch implementiert. Ähnlich zur schriftlichen Rechnung einer Addition auf dem Papier (im Dezimal- sowie Binärsystem), muss ein Addierwerk die Stellen einer Binärzahl Schritt für Schritt berechnen. Dies geschieht durch die geschickte Verknüpfung logischer Gatter, wodurch sich ein Halbaddierer bauen lässt. Zwei Halbaddierer zusammen ergeben entsprechend einen Volladdierer, welchen Sie in diesem Versuch in VHDL realisieren sollen. Der beim Rechnen entstandene Überhang wird Carry genannt. Dieser wird neben den Summanden in den Volladdierer eingegeben. Der durch die Addition gegebenenfalls neu entstandene Carry kann für n-bit Additionen mit dem Volladdierer der nächsten Stelle verknüpft werden.

Die von Ihnen realisierten Addierwerke sollen ausschließlich NAND-Gatter verwenden. Zunächst soll ein 1-bit Halbaddierer (HA) und ein 1-bit Volladdierer (VA) bestehend aus zwei HA realisiert werden. Mittels Generics und dem generate-Statement lassen sich dann Addierwerke beliebiger Bit-Breite instanziieren (n-bit Volladdierer).

Informationen zu den Halb- und Volladdierern finden sie im Vorlesungsskript und in den Aufzeichnungen der Vorlesung.

Generate

In VHDL (VHSIC Hardware Description Language), die zur Hardwarebeschreibung verwendet wird, dient die generate-Anweisung dazu, wiederholende Strukturen im Code zu erstellen. Sie ermöglicht die parametrisierte Instanziierung von Schaltungsblöcken oder Prozessen basierend auf Bedingungen zur Kompilierzeit. Mit anderen Worten: generate ermöglicht die dynamische Erzeugung von Hardwarekomponenten je nach den spezifizierten Parametern oder Bedingungen während der Kompilierung. Dies trägt zur Flexibilität und Wiederverwendbarkeit von VHDL-Code bei. Die generate-Anweisung wird dazu in Verbindung mit for genutzt (Lst.: 1), um eine zur Kompilierzeit festgelegte Anzahl von Instanzen zu erstellen.

```
1      <Label> : for <Schleifenbedingung> generate
2          <Zu Generierende Anweisung oder Komponente>;
3      end generate <Label>;
```

Listing 1: generate-Anweisung in einer for-Schleife

Die generate-Statements können auch mit if-Anweisungen kombiniert werden, um Sonderfälle der Instanziierung in Abhängigkeit des Schleifenzählers umzusetzen.

```
1      GEN : for i in 0 to <Range> generate
2          <Label> : if i = 0 generate <Statements> end generate;
3          <Label> : if i /= 0 generate <Statements> end generate;
4      end generate;
```

Listing 2: generate-Anweisung mit if-Bedingungen

So kann zum Beispiel unser *my_gen_and* auch mithilfe von generate-Statements und *my_and_gate* geschrieben werden:

```
architecture structure of my_gen_and is
begin
    GEN: for i in G_DATA_WIDTH - 1 downto 0 generate
        and_inst : entity work.my_and_gate(dataflow)
            port map (p_op1(i), p_op2(i), p_res(i));
    end generate;
end structure;
```

Listing 3: *my_gen_and* mit generate-Statement

Addierwerk und Subtrahierwerk – Grundkonzept

In diesem Versuch werden Addition und Subtraktion nicht als zwei getrennte Schaltungen realisiert. Stattdessen wird ein gemeinsames Addierwerk verwendet, das – abhängig von einer Steuervariable – entweder addiert oder subtrahiert.

Addierwerk (n-bit Ripple-Carry-Addierer)

Ein n-bit Addierwerk berechnet die Summe zweier Binärzahlen, indem es n Stück 1-bit Volladdierer hintereinander schaltet. Diese Struktur wird als *Ripple-Carry-Addierer* bezeichnet.

Funktionsweise

- Jeder Volladdierer verarbeitet genau ein Bit der Operanden A und B .
- Zusätzlich erhält jeder Volladdierer einen Übertrag (`carry_in`) aus der vorherigen Stelle.
- Der neu erzeugte Übertrag (`carry_out`) wird an die nächste Stelle weitergegeben.

Start- und Endübertrag

- Der Volladdierer der niederwertigsten Stelle (LSB) erhält einen externen Startübertrag `carry_in`.
- Der Übertrag der höchstwertigen Stelle (MSB) bildet den Ausgang `carry_out`.

Wichtig: Der Übertrag „wandert“ von Bit zu Bit durch die gesamte Schaltung. Dieses Verhalten ist einfach zu implementieren und eignet sich besonders gut für den Einsatz von generate-Anweisungen.

Subtrahierwerk (Subtraktion mit 2er-Komplement)

Ein separates Subtrahierwerk ist nicht notwendig. In digitalen Schaltungen wird eine Subtraktion stattdessen als Addition im 2er-Komplement durchgeführt:

Die Subtraktion zweier Zahlen A und B lässt sich wie folgt umformen:

$$A - B = A + (\neg B) + 1$$

Das bedeutet:

- Alle Bits des Subtrahenden B werden invertiert.
- Zusätzlich wird ein Übertrag von 1 addiert.

Damit kann dieselbe Addierschaltung sowohl für Addition als auch für Subtraktion verwendet werden. Es ändert sich lediglich die Art, wie die Operanden eingespeist werden.

Kombiniertes Addier- und Subtrahierwerk

Zur Umschaltung zwischen Addition und Subtraktion wird ein Steuersignal `sub` verwendet.

<code>sub</code>	Operation	Berechnung
0	Addition	$A + B$
1	Subtraktion	$A + (\neg B) + 1$

Die Umschaltung erfolgt mit zwei einfachen Maßnahmen:

- Jedes Bit von B wird mit `sub` exklusiv-oder-verknüpft: $B'(i) = B(i) \oplus sub$
- Der Startübertrag des Addierwerks wird auf `sub` gesetzt: `carry_in = sub`
- Bei `sub` = 0 bleibt B unverändert und es wird normal addiert.
- Bei `sub` = 1 wird B invertiert und der zusätzliche Übertrag realisiert das 2er-Komplement.

Wichtig: Der eigentliche Addierkern (Volladdierer-Kette) bleibt unverändert.

Aufgaben

Zur Vorbereitung kopieren Sie den gesamten von Moodle heruntergeladenen Ordner in ein eigenes Verzeichnis Ihres Home-Verzeichnisses. Sie können davon ausgehen, dass die Testbenches Zahlen im 2er-Komplement interpretieren.

Aufgabe 1: 1-bit Volladdierer (NAND)

In dieser Aufgabe sollen Sie einen aus NANDs bestehenden 1-bit Volladdierer erstellen und gegen die vorgegebene Testbench testen.

Hinweise zur Umsetzung:

- Erstellen Sie eine Wahrheitstabelle für einen Halbaddierer (HA).
- Minimieren Sie die DKNF des HA und formen Sie diese so um, dass sie nur aus NAND-Verknüpfungen besteht.
- Setzen Sie die Entity `my_half_adder`, basierend auf Ihrer Umformung in der vorgegebenen Datei `my_half_adder.vhdl` und der schematischen Spezifikation der Ports in Abbildung 1 um.

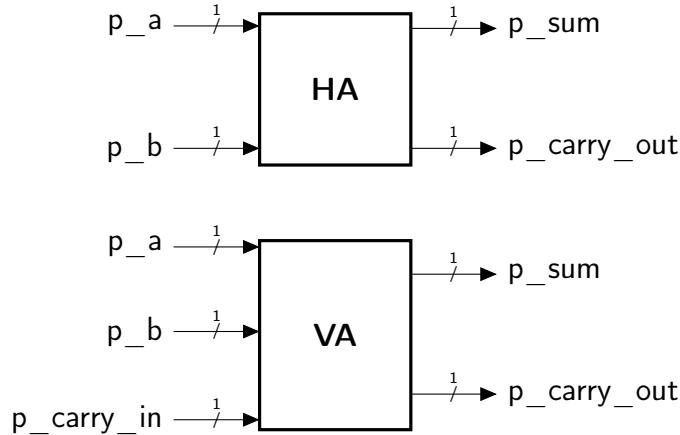


Abbildung 1: Entities my_half_adder und my_full_adder

- Testen Sie Ihren Halbaddierer erfolgreich mit der vorgegebenen Testbench my_half_adder in my_half_adder_tb.vhdl.
- Erstellen Sie nun die Entity my_full_adder, einen 1-bit Volladdierer (VA), in der Datei my_full_adder.vhdl, unter Verwendung Ihres erstellten HA. Verwenden Sie ausschließlich NAND-Verknüpfungen und geben Sie notwendige Umformungen als Kommentar an.
- Testen Sie Ihre Umsetzung gegen die vorgegebene Testbench my_full_adder_tb in der Datei my_full_adder_tb.vhdl erfolgreich.
- Laden Sie Ihre erstellten und angepassten Dateien als Zip-Datei Name_Vorname_V7_A1.zip in Moodle hoch.

Abnahme Standard VHDL, Grundlagen Halb- und Volladdierer

Aufgabe 2: Generischer n-bit Volladdierer

Implementieren Sie nun einen generischen n-bit Volladdierer, unter Verwendung der generate-Anweisung aus Ihrem 1-bit VA aus Aufgabe 1.

- Ergänzen Sie die gegebene Datei my_gen_full_adder.vhdl so, dass Sie mittels Generics und der Funktion generate eine beliebige Anzahl Instanzen der Entity my_full_adder erstellen können. Überprüfen Sie Ihre Umsetzung erfolgreich mit der Testbench my_gen_full_adder_tb.vhdl.
- Erweitern Sie nun Ihre Implementierung so, dass sie unter der Annahme, dass die Operanden im 2er-Komplement angegeben sind, auch subtrahieren können (Randbedingung generate).

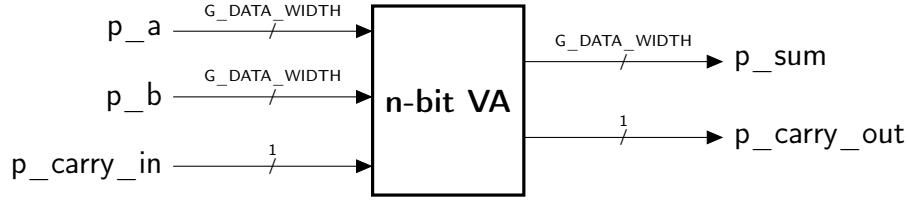


Abbildung 2: Entity my_gen_full_adder

Hinweis: Die Subtraktion ist ausschließlich durch Invertieren des Operanden p_b und Setzen des Startübertrags p_carry_in zu realisieren. Der Addierkern selbst darf nicht verändert werden.

- Überprüfen Sie Ihre Implementierung nun erfolgreich mit der Testbench my_gen_full_adder_tb2.vhdl, welche die Addition und Subtraktion von positiven Zahlen im 2er-Komplement überprüft.
- Laden Sie Ihre erstellten und angepassten Dateien als Zip-Datei Name_Vorname_V7_A2.zip in Moodle hoch.

Abnahme Standard VHDL, Generics, Grundlagen Addierwerk/Subtrahierwerk

Knobelaufgabe Versuch 7:

Ergänzen Sie die Testbench in my_gen_full_adder_tb2.vhdl so, dass Sie die Subtraktion und Addition für alle positiven und negativen Zahlen (erschöpfend) im 2er-Komplement überprüfen. Bedenken Sie dabei mögliche Überläufe und andere Probleme im Vergleich zur Referenzfunktion. Testen Sie mit der angepassten Testbench Ihre Umsetzung erfolgreich und laden Sie die abgeänderte Testbench als Zip-Datei Name_Vorname_V7_K1.zip in Moodle hoch.

Abnahme Standard VHDL, Generics, Überlauf, 2er-Komplement, Addier- & Subtrahierwerk

Abgabeübersicht

Aufgabe	Datei	Inhalt
A1	Name_Vorname_V7_A1.zip	<ul style="list-style-type: none">• my_half_adder.vhdl• my_half_adder_tb.vhdl• my_full_adder.vhdl• my_full_adder_tb.vhdl• ggf. *.ghw (Simulationsergebnisse)
A2	Name_Vorname_V7_A2.zip	<ul style="list-style-type: none">• my_gen_full_adder.vhdl• my_gen_full_adder_tb.vhdl• my_gen_full_adder_tb2.vhdl• ggf. *.ghw (Simulationsergebnisse)
K1	Name_Vorname_V7_K1.zip	<ul style="list-style-type: none">• angepasste my_gen_full_adder_tb2.vhdl (erschöpfende Tests)• ggf. *.ghw (Simulationsergebnisse)

Hinweise

- Alle Dateien müssen syntaktisch korrekt, kompilierbar und simulierbar sein.
- Nutzen Sie klare Kommentare in allen VHDL-Dateien.
- Die ZIP-Dateien müssen exakt nach Vorgabe benannt werden.
- Testen Sie die generischen Addierer für mehrere Bitbreiten (z. B. 1, 4, 8, 16 Bit).
- Für K1: Überläufe, negative Zahlen und 2er-Komplement vollständig abdecken.