

Bericht

VHDL-Miniprojekt IR-Dekoder

Durchführung:	A. Schmocker, T. Lang
Abgabedatum:	22.01.2016
Modul:	BTE5023 Elektronische Systeme
Betreuer:	Thorsten Mähne
Projektseite:	https://github.com/id101010/vhdl-irdecoder

Inhalt

[1 Einführung](#)

[1.1 Projektplanung](#)

[2 Komponenten und Blockschema](#)

[2.1 Aufbau des IR-Signals](#)

[2.2 Aufbau des Gesamtsystems](#)

[2.3 Decoder](#)

[2.3.1 Übersicht](#)

[2.3.2 Implementierung](#)

[2.3.2 Test](#)

[2.4 Seriell Parallel Wandler](#)

[2.4.1 Übersicht](#)

[2.4.2 Implementierung](#)

[2.4.3 Test](#)

[2.5 Clock Divider](#)

[2.5.1 Übersicht](#)

[2.5.2 Implementierung](#)

[2.5.3 Test](#)

[2.6 Outputswitcher](#)

[2.6.1 Übersicht](#)

[2.6.2 Implementierung](#)

[2.6.3 Test](#)

[2.7 hex2seg LCD Driver](#)

[2.7.1 Übersicht](#)

[2.7.2 Implementierung](#)

[2.7.3 Test](#)

[3 Diskussion](#)

[4 Fazit](#)

1 Einführung

Während der 8 Lektionen, die für dieses Projekt vorgesehen sind, soll ein Infrarot (IR)-Dekoder in VHDL entwickelt, auf CPLD implementiert und dokumentiert werden, der das Signal der IR-Fernbedienung RMC-D10 von SUN auswertet und das ermittelte 20-bit breite Datenwort hexadezimal auf der 7-Segment LCD-Anzeige des CPLD-2014-Boards anzeigt. Zur Funktionskontrolle des IR-Empfangsmodules soll die Aktivität des empfangenen Signales über eine LED signalisiert werden. Die Plausibilität des empfangenen Kommandos soll geprüft werden (Timing des IR-Signals für die Kommandos, korrekte Anzahl der Bits pro Kommando, korrekte Kennung der RMC-D10, doppelter Empfang des gleichen Kommandos).

1.1 Projektplanung

Wir haben uns beim Projekt folgende Meilensteine gesetzt:

Datum	Meilenstein
11.12.2015	Grobplan, Konzept fertig
18.12.2015	Modularisierung, Schnittstellendefinition abgeschlossen
01.01.2016	Entwicklung der einzelnen Module und deren Testbenches abgeschlossen
10.01.2016	Erster Test auf Hardware auf Zielhardware durchgeführt
21.01.2016	Toleranzverhalten verbessert und Fehlererkennung erhöht
22.01.2016	Alle Fehler behoben und Dokumentation fertiggestellt

2 Komponenten und Blockschema

2.1 Aufbau des IR-Signals

Die Infrarot Fernbedienung RMC-D10 stammt ursprünglich von einem alten Sun Bildschirm. Die Daten werden mit einem modifizierten NEC Code mit Puls-Distanz Modulation übertragen. Die Datenübertragung beginnt, beim Betätigen einer Taste, mit einem sogenannten Leader von einer Dauer von 2.5 ms (Low aktiv).

Anschliessend werden die 20 Datenbits beginnend mit dem niederwertigsten Bit übertragen (LSB first). Die Dauer beträgt für eine logische "1" 1.3 ms und für eine logische "0" 655 µs (Low aktiv).

Zwischen den einzelnen Bits wechselt das Signal für ca. 574 µs auf High. Solange die Taste gedrückt bleibt, wird die Übertragung der Daten alle 45.2 ms wiederholt. Bei den Tasten 19 und 20 werden die Daten nur dreimal übertragen.

In der folgenden Tabelle ist den Tasten der jeweilige Infrarot-Code in Binärer und hexadezimaler Form zugeordnet. Diese Tabelle wird auch verwendet um das Signal in decodierter Form (Switch Nr) als 2 Dezimalziffern darzustellen.

Switch Nr.	Switch Name	Binary Data	Hex Data
1	Bright Down	00001001110100011111	09d1f
2	Bright Up	00001001110100011110	09d1e
3	Vertical Down	00001001110100001111	09d0f
4	Vertical Up	00001001110100001110	09d0e
5	Height Down	00001001110100101001	09d29
6	Height Up	00001001110100101000	09d28
7	Convergence Down	00001001110100101101	09d2d
8	Convergence Up	00001001110100101100	09d2c
9	Tilt left Down	00001001110100010001	09d11
10	Tilt right Down	00001001110100010000	09d10
11	Contrast Down	00001001110100011001	09d19
12	Contrast Up	00001001110100011000	09d18
13	Horizontal Left	00001001110100001101	09d0d
14	Horizontal Right	00001001110100001100	09d0c
15	With Left	00001001110100100111	09d27
16	With Right	00001001110100100110	09d26
17	Convergence Left	00001001110100101011	09d2b
18	Convergence Right	00001001110100101010	09d2a
19	Size Down	00001001110100111000	09d38
20	Size Up	00001001110100111011	09d37

Fig. 1: Zuweisung von Infrarot-Code zu Tasten-Nummern.

2.2 Aufbau des Gesamtsystems

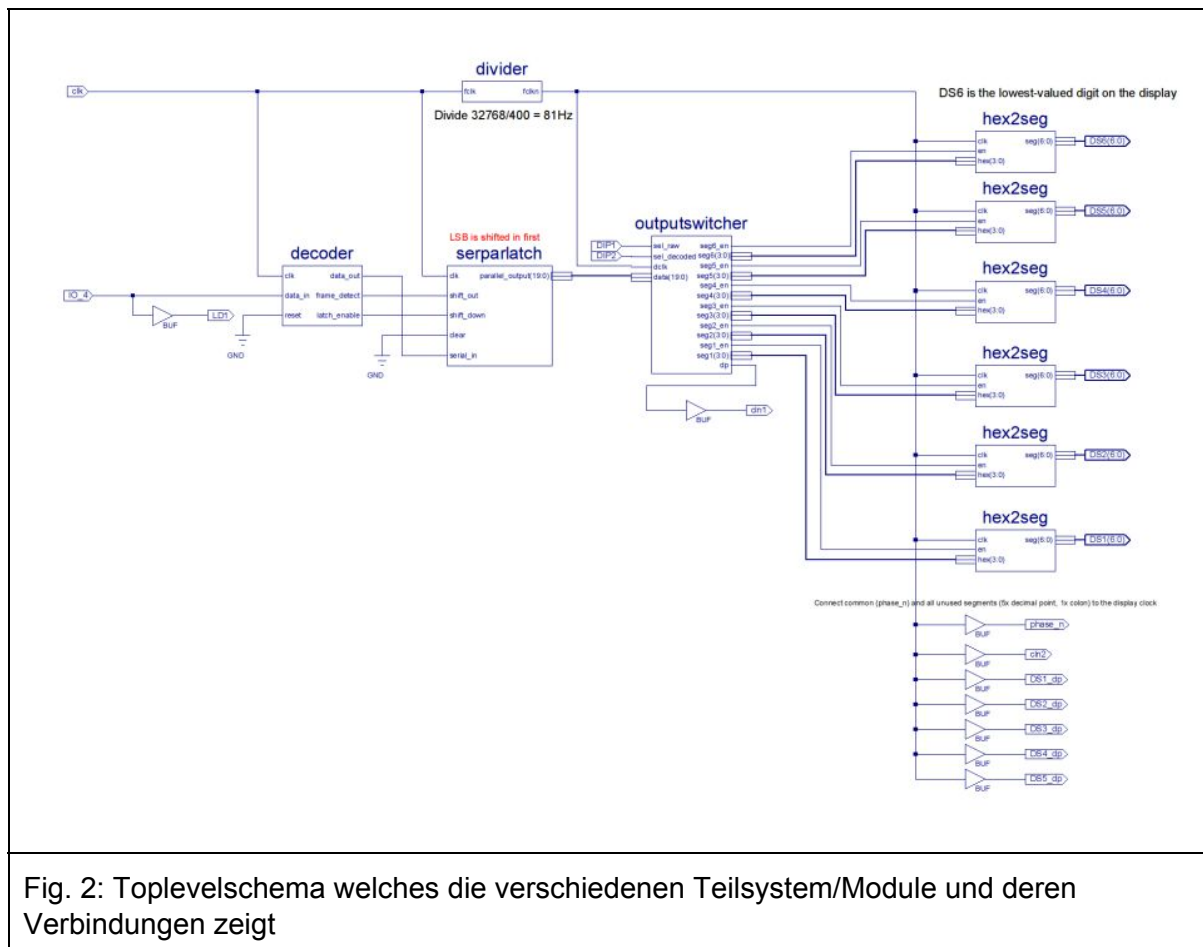


Fig. 2: Toplevelschema welches die verschiedenen Teilsystem/Module und deren Verbindungen zeigt

Das Gesamtsystem ist aus mehreren Teilsystemen aufgebaut. Im System sind 2 verschiedene Clocks vorhanden: Der Systemclock mit 32.768 kHz und der Display Clock der mit einem Clock-Divider auf 81Hz geteilt wurde.

Das Eingangssignal wird vom "Decoder"-Block analysiert und in einen seriellen Bitstrom umgewandelt. Zusätzlich wird das Eingangssignal auf einer LED ausgegeben.

Der serielle Bitstrom wird anschliessend in das Modul "serparbuf" geführt welches eine Seriell-Parallel Wandlung durchführt und den Output solange zurückhält bis der Decoder alle 20-Bits empfangen hat. Am Ausgang des "serparbuf" liegen dann die 20 Datenbit in paralleler Form an, mit dem LSB an der richtigen (=tiefsten) Stelle.

Anschliessend werden die Daten in das Modul "outputswitcher" geführt.

Der Outputswitcher entscheidet auf Grund von Schalter-Inputs wie der Output dargestellt werden soll.

Folgende Ausgabe Modi werden unterstützt (implementiert im Modul outputswitcher):

DIP2	DIP1	Ausgabe
0	0	Display ausgeschaltet
0	1	Anzeige des 20 bittigen Komandos in Hex-Form (5 Ziffern)
1	0	Anzeige des decodierten Codes (Mapping siehe oben), oder "EE" bei Error. Die Darstellung erfolgt dezimal (2 Ziffern)
1	1	Anzeige des decodierten Codes (2 Dezimalziffern), Trennzeichen und die 16 niederwertigsten Bits des empfangen Kommandos in Hex-Form (4-Ziffern).

Zusätzlich zum Umschalten des Ausgabemodus nimmt der outputswitcher auch das Decodieren des Bitstroms zu Tasten-Nummern (1-20) vor.

Der Outputswitcher hat für jedes der 6 Segmente einen Ausgang welcher die anzuzeigende Hexzahl (0-F) repräsentiert und einen Enable Ausgang welcher beschreibt ob das jeweilige Segment eingeschaltet sein soll oder nicht. Zusätzlich steuert der Outputswitcher das Trennzeichen direkt an. Der Ausgang für das Trennzeichen ist bereits eine Wechselspannung, während die anderen Ausgänge noch nicht moduliert sind.

Das Modul hex2seg wandelt anschliessend die logischen Signale des Outputswitchers um in modulierte- bzw Wechselspannungs-Signale. Innerhalb dieses Blocks wird auch bestimmt welche Segmente bei welcher Zahl (am Eingang) aktiviert werden müssen.

2.3 Decoder

2.3.1 Übersicht

 <p>The diagram shows a rectangular block labeled 'decoder' in large blue text. Inside the block, there are six input/output ports arranged in three rows. The top row has 'clk' on the left and 'data_out' on the right. The middle row has 'data_in' on the left and 'frame_detect' on the right. The bottom row has 'reset' on the left and 'latch_enable' on the right. Each port has a short horizontal line extending from the block boundary.</p>	<p>Das Decodermodul nimmt als Eingang das Signal eines IR-Demodulators. Seine Aufgabe ist es das Signal aufzubereiten und zu demodulieren. Das heisst, die Information welche in den Pulszeiten enthalten ist wird in logische Pegel umgewandelt.</p> <p>Zusätzch führt das Modul Error-Checks durch um fehlerhafte Signalerkennungen zu vermeiden. Der Ausgang ist seriell und behinhaltet die demodulierten Bits.</p>
<p>Fig. 3: Schemasympol des Decoders</p>	<p><u>Ein- und Ausgänge:</u></p> <p>clk: Clock Signal data_in: Serieller Dateninput (wird mit clk abgetastet) reset: Asynchroner, High-Aktiver Reset data_out: Serieller output. Zeigt den Pegel des aktuell Empfangen Bits. Ist valid wenn latch_enable = 1 frame_detect: Signal zur Anzeige des Frame-Endes (nach 20 bit). Synchron. latch_enable: Signal zur Anzeige des Bit-Endes. Das Signal data_out kann jetzt in ein Schieberegister eingelesen werden.</p> <p><u>Generische Parameter:</u></p> <p>input_freq: Clock Frequenz in Hz start_time: Dauer des Start-Bit in us one_time: Dauer des 1-Bit in us zero_time: Dauer des 0-Bit in us pause_time: Dauer des Pause-Bit in us tolerance_time: Toleranzzeit +/- us</p>

2.3.2 Implementierung

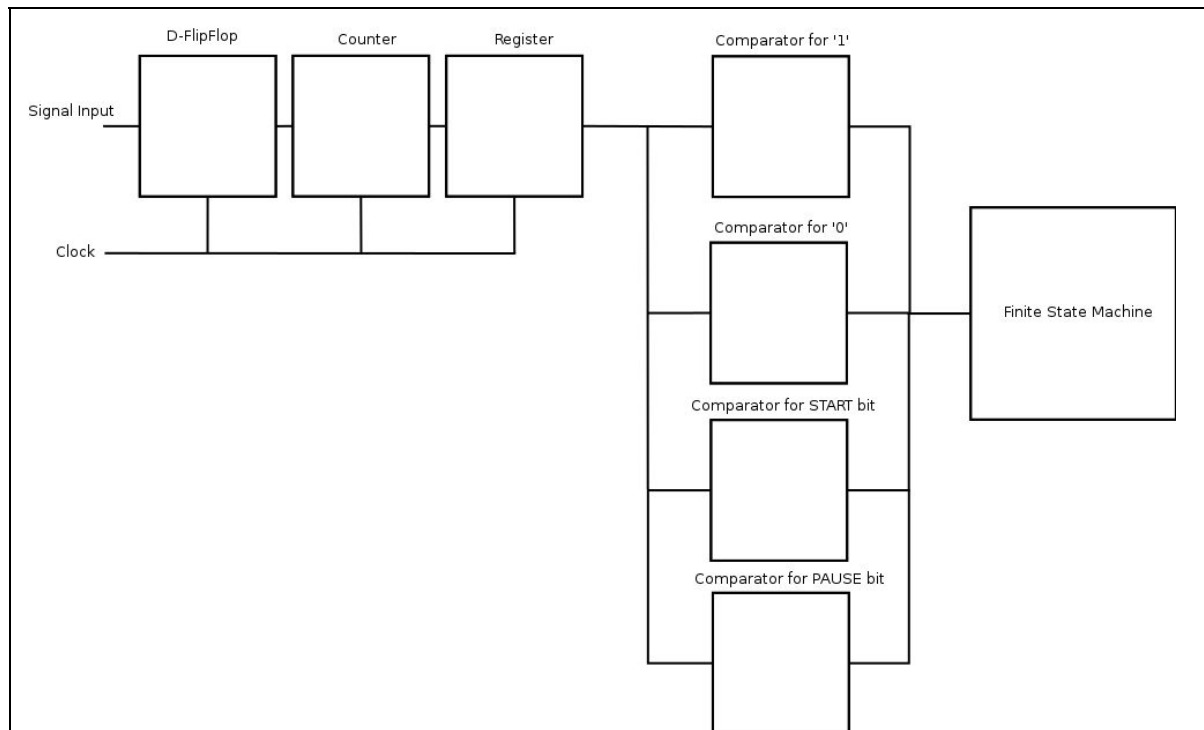


Fig. 4: Blockschema des Decoders

Der Decoder ist nach folgender Grundidee aufgebaut:

Das serielle Eingangssignal wird synchron zum Grundtakt (clock) abgetastet. Der interne Counter beginnt jeweils (bei 0) zu zählen wenn eine positive oder negative Flanke am Eingangssignal erkannt wird. Bei der nachfolgenden Flanke wird dann der Zählerwert in das pulse_time Zwischenregister übernommen und der Counter beginnt die Zeit bis zur nächsten Flanke zu zählen.

Dieser Wert wird anschliessend durch mehrere Komparatoren ausgewertet und es wird entschieden ob das Signal einer **logischen '1'**, einer **logischen '0'**, einem **Startbit** oder einer **Pause** entspricht. Dabei werden auch Toleranzzeiten miteinbezogen um Fehlererkennungen vorzubeugen.

Diese so gefilterten Werte werden anschliessend einer State-Machine übergeben welche dann die eigentliche Decodierung vornimmt. Die State-Machine erzeugt aus den erfassten Zuständen einen Bitstrom welcher dem demodulierten Signal entspricht.

Der Output des Decoders ist wie folgt aufgebaut:

Wird ein Bit erkannt, so wird dessen Pegel an data_out angelegt. Zusätzlich wird latch_enable für 1 Clock-Zyklus aktiviert. Sobald 20 Bits empfangen wurden wird der frame_detect für 1 Clock-Zyklus aktiviert. Der Decoder wird also das LSB zuerst senden, weil es auch zuerst empfangen wurde.

2.3.2 Test

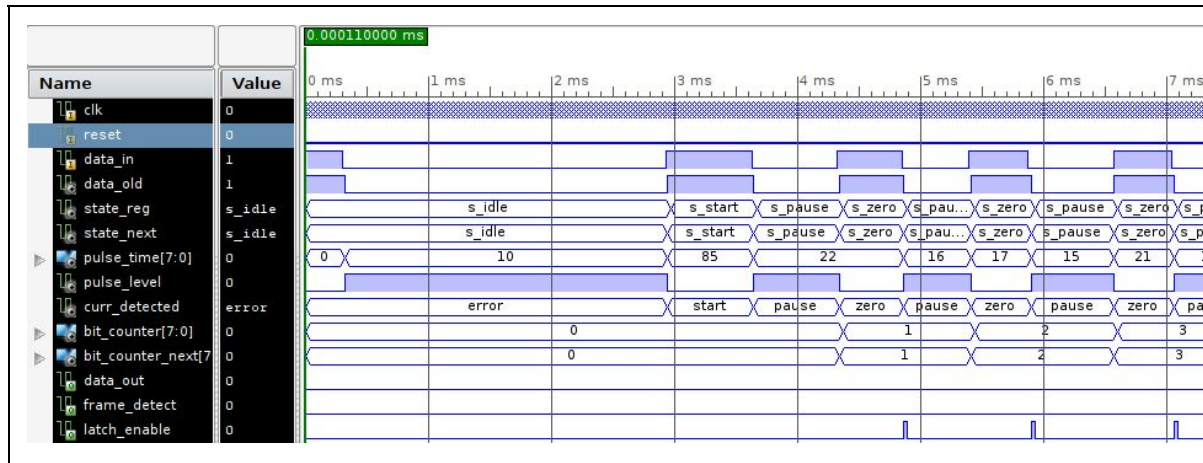


Fig. 5: Testbench des Decoders, zu Beginn einer Übertragung.

In der **pulse_time** Zeile ist anhand der unterschiedlichen Werte sichtbar dass nicht jedes logische '1' genau gleich lang dauert. Dies entsteht dadurch dass die Testbench Zufallswerte mithilfe der Rand-Funktion erzeugt um die Toleranz des Decoders zu prüfen. In der **curr_detected** Zeile ist jeweils das erkannte Bit (0,1, oder Starbit) dargestellt. Der **bit_counter** wird nach empfang eins Bits jeweils inkrementiert und **latch_enable** ist jeweils während nur einer Taktflanke aktiv, um die Gültigkeit des **data_out** Signals zu signalisieren.

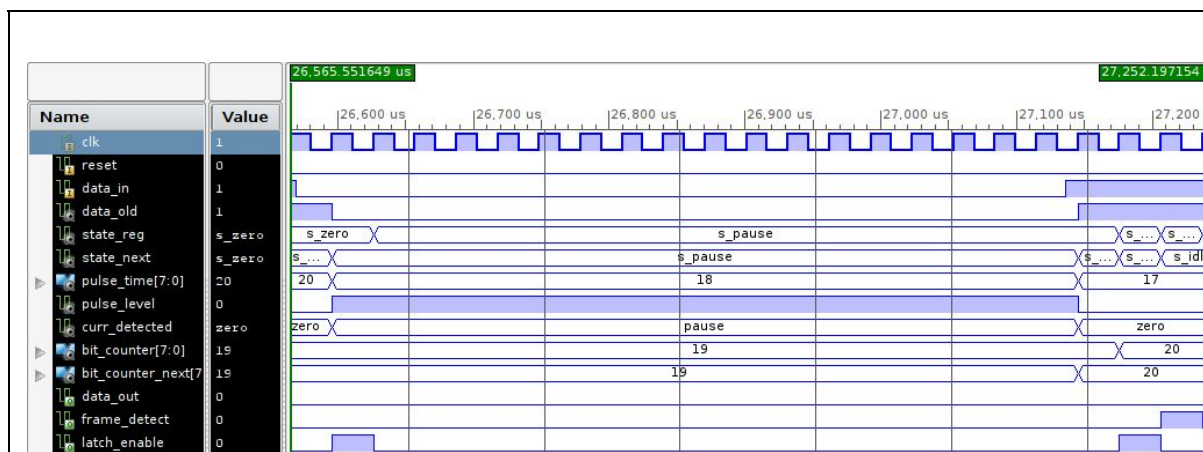
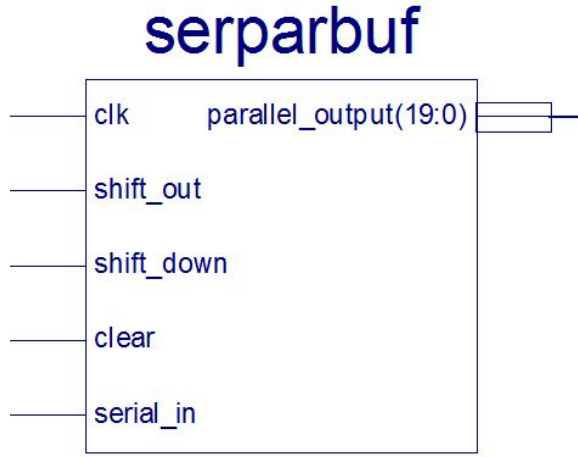


Fig. 6: Testbench des Decoders am Ende der Übertragung bzw beim Empfang des 20. Bits.

Es ist erkennbar dass nach Empfang des 20. Bits zuerst **latch_enable** für eine Taktflanke aktiv ist, und anschliessend **frame_detect** um das Ende der Übertragung zu signalisieren.

2.4 Seriell Parallel Wandler

2.4.1 Übersicht

 <p>The diagram shows a block labeled 'serparbuf'. It has five input ports on the left: 'clk', 'shift_out', 'shift_down', 'clear', and 'serial_in'. It has one output port on the right labeled 'parallel_output(19:0)'.</p>	<p>Das SerParBuf-Modul führt eine Serial-Parallel Wandlung des Eingangssingals (serial_in) durch und buffert zusätzlich den Input.</p> <p>Der ganze Baustein ist synchron zu betreiben, besitzt aber einen aynchronen clear/reset Eingang.</p>
<p>Fig. 7: Schemasympol des SerParBufs</p>	<p><u>Ein- und Ausgänge:</u></p> <p>clk: Clock Signal</p> <p>shift_out: Übernimmt den Status des internen Buffers an den Ausgang, bei nächsten Takt-Zyklus.</p> <p>shift_down: Aktiviert das schieben um 1 Bitstelle (abwärts) und gleichzeitige initialisiern des obersten Bits mit dem Wert von <i>serial_in</i> beim nächsten Takt-Zyklus.</p> <p>clear: Aynchrone, Highaktivier Reset/Clear</p> <p>serial_in: Serieller Daten-Eingang (LSB first)</p> <p>parallel_output: Paralleler Ausgang</p> <p><u>Generische Parameter:</u></p> <p>nbits: Bitbreite des Ausgangs</p>

2.4.2 Implementierung

Das Modul setzt sich aus einem Schieberegister mit seriellem Eingang und parallelem Ausgang und einem Latch zusammen. Der Eingang ist vom Typ `std_logic`, der Ausgang vom Typ `std_logic_vector` mit einer generischen Bitbreite.

Liegt an **shift_down** während einer positiven Flanke auf **clk** ein positiver Pegel an, so wird der Pegel von **serial_in** in das oberste Bit des Schieberegisters übernommen und alle anderen Bits werden um eine Stelle nach unten geschoben.

Die Änderungen am internen Buffer, werden allerdings erst sichtbar wenn während einer positiven Flanke auf **clk** ein positiver Pegel an **shift_out** anliegt. In der Zwischenzeit ist am Ausgang der Wert des Latches sichtbar.

2.4.3 Test

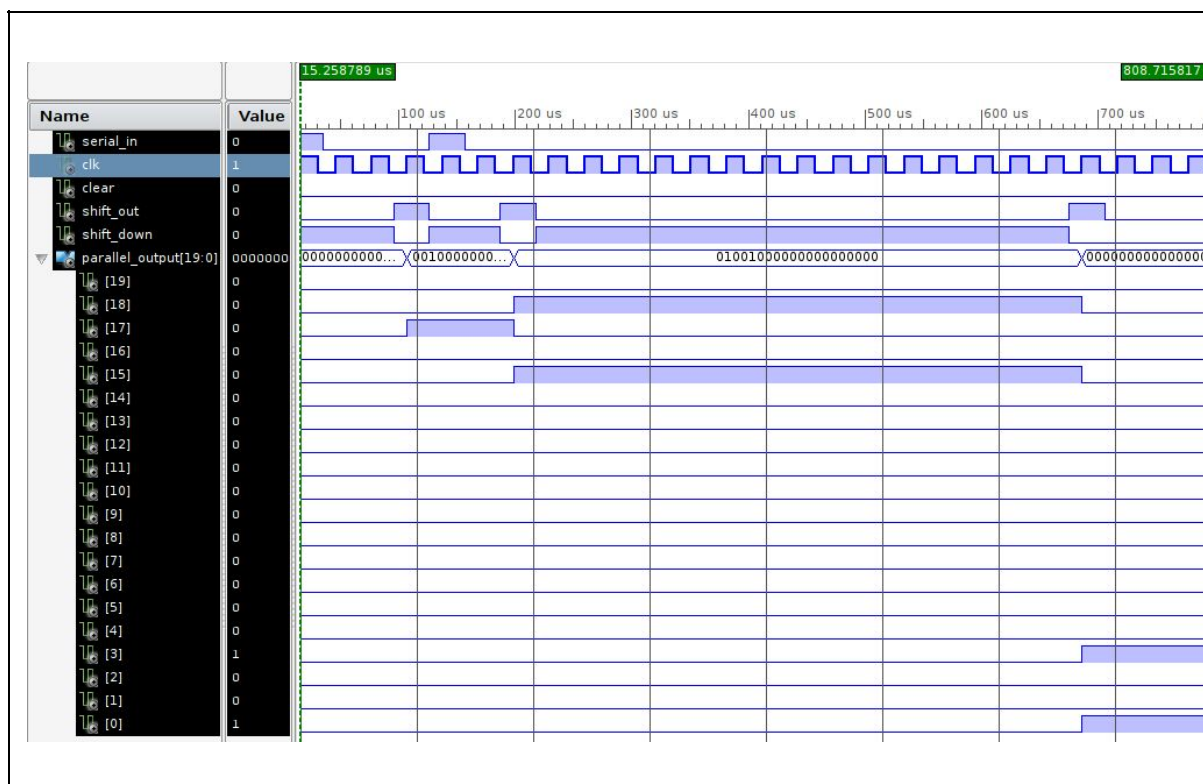


Fig. 8: Testbench des Moduls serparbuf


Zu Beginn werden 3 bits ("100") hinein geschoben, diese werden am Ausgang sichtbar (von Pin 19-17) sobald **shift out** aktiviert wird.

Anschliessend werden nochmals 2 Bits ("10") hineingeschoben und der Ausgang wird aktiviert. Man kann erkennen dass jetzt am Ausgang von Pin 19-15 das richtige Bitmuster anliegt (von oben nach unten "01001").

Abschliessend werden nochmals 15 "0"-Bits hineingeschoben um zu prüfen ob das Bitmuster dann auch in der richtigen Position (Bits 4-0) landet.

2.5 Clock Divider

2.5.1 Übersicht

<div style="text-align: center;">  <p>Divide 32768/400 = 81Hz</p> </div>	<p>Das Divider Modul teilt eine beliebige Eingangsfrequenz durch einen Faktor, welcher durch einen generic-Parameter einstellbar ist.</p> <p>Die Ausgangsfrequenz weist ein Tastverhältnis von 50% (Duty-Cycle) auf.</p>
<p>Fig. 9: Schemasymbol des Clockdividers</p>	<p><u>Ein- und Ausgänge:</u> fclk: Eingangsfrequenz fclkn: Entsprechende Ausgangsfrequenz</p> <p><u>Generische Parameter:</u> n: Teilungsverhältnis, hier 400</p>

2.5.2 Implementierung

Auf jede steigende Flanke von **fclk** wird der Wertebereich eines internen Zählers überprüft. Ist der Wert zwischen der Null und der Hälfte des gewünschten Teilungsfaktors ($n/2$), so wird der Ausgang gelöscht und der Zähler inkrementiert. Ist der Wert zwischen $n/2$ und dem Wert des Teilungsfaktors so wird der Ausgang gesetzt und der Zähler inkrementiert. Wenn der Zähler den Wert des Teilungsfaktors erreicht hat wird er gelöscht und der Ausgang auf 0 gesetzt.

Der Teilungsfaktor kann über einen generischen Parameter eingestellt werden und der interne Zähler ist ein std_logic_vector mit einer breite von 16bit.

2.5.3 Test

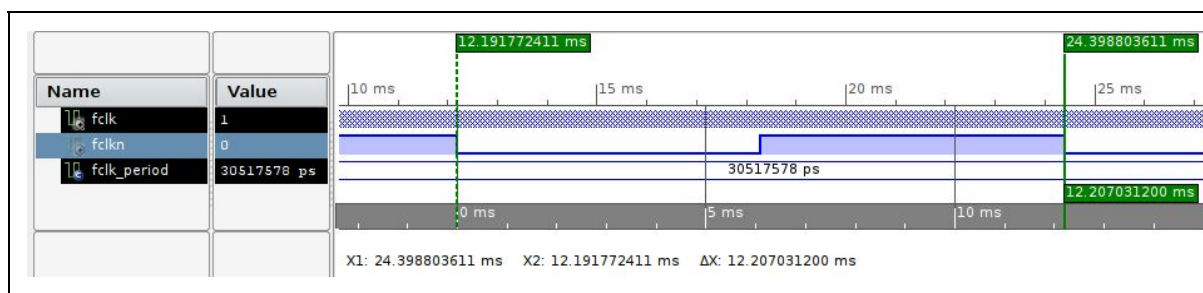


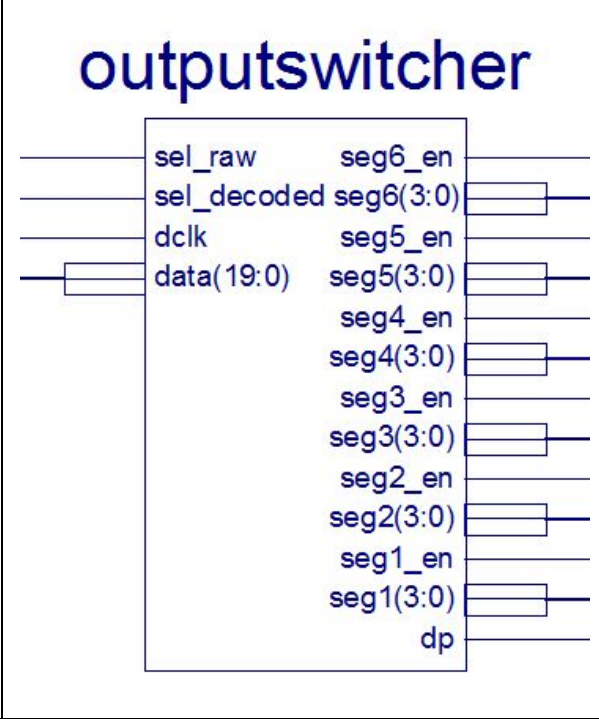
Fig. 10: Testbench des divider-Moduls

Die Periode des Eingangssignals in obiger Testbench beträgt 30517578ps. Das entspricht einer Eingangsfrequenz von 32,768kHz.

Das Ausgangssignal weist eine Periode von 12,207ms auf, was einer Ausgangsfrequenz von 81,9Hz entspricht. Der Teilungsfaktor beträgt in dem Beispiel 400 was dem Quotienten von Eingangs- und Ausgangsfrequenz entspricht.

2.6 Outputswitcher

2.6.1 Übersicht

	<p>Das outputswitcher Modul entscheidet auf Grund der Pegel der beiden Konfigurationsleitungen (sel_raw und sel_decoded) welche Hex-Ziffern auf welchem Segment dargestellt werden soll. Die möglichen Konfigurationen wurden im Kapitel 2.2 beschrieben.</p> <p>Als Eingang wird zusätzlich zu den Konfigurations-Leitungen der Display-Clock und die 20 Datenbit (IR-Code) erwartet.</p> <p>Am Ausgang des Moduls ist für jedes Segment jeweils ein Ausgang mit der anzuzeigenden Hex-Zahl und ein Enable Ausgang vorhanden.</p>
<p>Fig. 11: Schemasympool des Outputswitchers</p>	<p>sel_raw: Aktiviert die Anzeige des 20-bit breiten Empfangenen Bitstromes (4 oder 5 Hexziffern)</p> <p>sel_decoded: Aktiviert die Anzeige der decodierten Tasten-Nr (2 Dezimal-Ziffern)</p> <p>dclk: Display Clock</p> <p>data: 20-bit Daten Eingang (IR-Komando)</p> <p>seg{x}_en: Enable (High Aktiv) für Segment Nr x.</p> <p>seg{x}: Anzuzeigende Hex-Zahl (0-F) für Segment Nr x.</p> <p>dp: Ansteuerungssignal für Trennzeichen (zwischen dezimal/hex darstellung). Moduliert mit Display-Clock (Phase/Gegenphase)</p>

2.6.2 Implementierung

Das Modul wurde rein kombinatorisch implementiert. Alle Ausgänge konnten mithilfe von nebenläufigen Statements zugewiesen werden. Zu Beginn der Modul-Implementierung wird mithilfe einer with-select Anweisung der 20-bit breiter IR-Code in eine Tasten-Nummer (siehe Kapitel 2.1) umgewandelt. Die Ausgänge **seg6**, **seg5**, **seg4** und **seg3** werden jeweils 4-bit (als Hexzahl) der niederwertigsten 16bit des Eingangs zugeordnet. Die entsprechenden Enable Leitungen werden allerdings nur aktiviert, wenn auch **sel_raw** aktiv ist.

Dem Ausgang **seg1** wird die Zehner-Stelle der Tasten-Nr (1-20) zugewiesen. Die Enable Leitung **seg1_en** wird nur aktiviert, wenn auch **sel_decoded** aktiv ist.

Segment Nr 2 ist doppelt belegt. Wenn nur **sel_raw** aktiv ist, aber nicht **sel_decoded**, so werden werden die Bits 16-19 des IR-Codes als Hex-Zahl angezeigt. Wenn allerdings **sel_decoded** aktiv ist, wird die Einer-Stelle der Tasten-Nr(1-20) angezeigt, unabhängig von **sel_raw**. Die Enable Leitung **seg2_en** ist nur aktiv, wenn mindestens **sel_raw** oder **sel_decoded** aktiv ist.

Das Trennzeichen (**dp** Ausgang) wird aktiviert wenn **sel_decoded** und **sel_raw** gleichzeitig aktiv sind. Aktiv heisst in diesem Falle dass der Ausgang **dp** auf das Gegenteil von **dclk** gesetzt wird (=Gegenphase), nicht aktiv bedeutet dass der Ausgang **dp** auf den Wert von **dclk** gesetzt wird (= in Phase).

2.6.3 Test

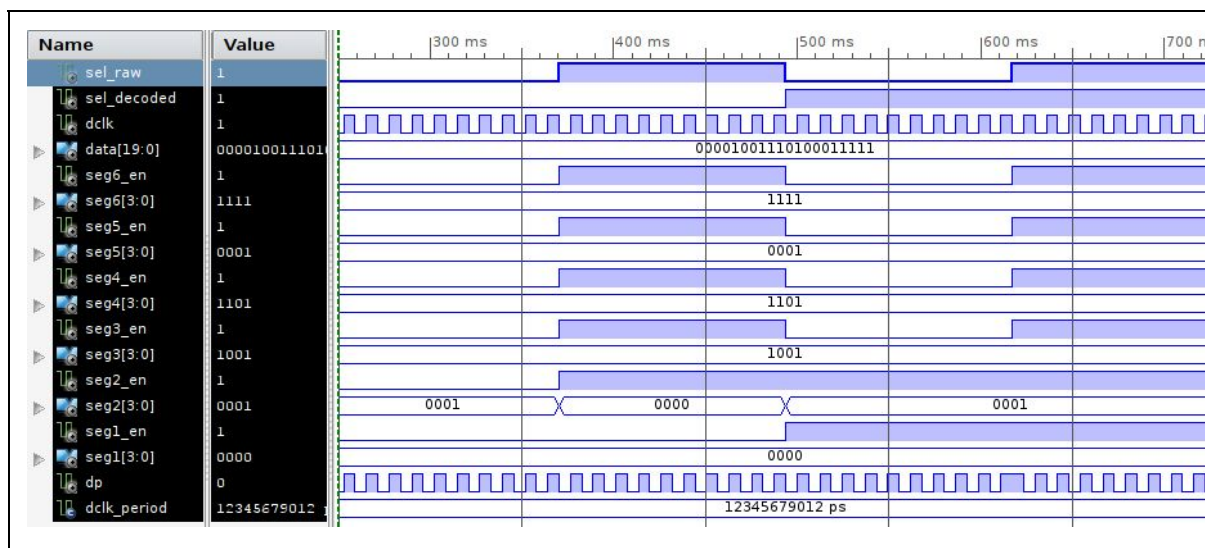
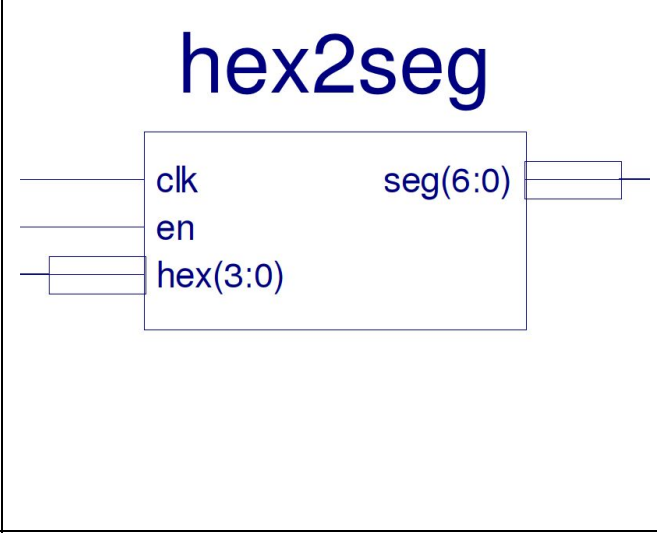


Fig. 12: Testbench des Outputswitcher-Moduls

In der Testbench wird erst ein valides Signal an den Dateneingang **data** angelegt und anschliessend werden alle Kombinationen von **sel_raw** und **sel_decoded** eingestellt. Die Ausgänge **seg_n** und deren Enablebits **seg_n_en** werden gemäss Konzept richtig gesetzt. Man sieht auch dass wenn beide Statusbits aktiv sind, der Ausgang **dp** von In-Phase mit **dclk** zu In-Gegenphase mit **dclk** wechselt. Damit wird das Trennzeichen sichtbar.

2.7 hex2seg LCD Driver

2.7.1 Übersicht

	<p>Das Modul hex2seg steuert eine einzelne 7-Segment Zelle des LCDs an.</p> <p>Das Modul decodiert eine 4bit Zahl (hex) und erzeugt ein Steuersignal für die entsprechenden Segmente (seg). Da das Display nur mit einem Wechselsignal angesteuert werden darf, muss zusätzlich ein Grundtakt (clk) auf das Ausgangsbitmuster aufmoduliert werden.</p> <p>Zusätzlich lässt sich das ganze Modul über ein enable Bit (en) mittels positiver Logik ein oder ausschalten.</p>
<p>Fig. 13: Schemasympol des LCD Drivers</p>	<p><u>Ein- und Ausgänge:</u></p> <p>clk: Display clock, refresh Rate mit welchen die einzelnen Segmente angesteuert werden.</p> <p>en: Ist das enable Bit, wenn gesetzt wird der Clock auf das Bitmuster moduliert.</p> <p>hex: Anzuzeigende HEX-Nummer</p> <p>seg: Output für 7seg Anzeige, moduliert mit Clock.</p>

2.7.2 Implementierung

Das Modul ist rein kombinatorisch aufgebaut. Es verwendet einen 7-Bit breiten Datenvektor und einen ebenfalls 7-Bit breiten Clockvektor. Das **clk**-Eingangssignal wird auf alle 7-Bit des Clockvektors dupliziert. Anschliessend wird für das Eingangsbitmuster am Eingang **hex** das entsprechende Ausgangssignal in den Datenvektor geladen. Dies passiert mittels with-select Anweisung.

Das Ausgangssignal **seg** ist eine XOR Verknüpfung zwischen dem Daten- und dem Clockvektor. Dies passiert aber nur wenn das enable-Bit **en** gesetzt ist. Ansonsten wird nur der Clockvektor auf die Segmente ausgegeben. Aktive Segmente befinden sich also mit dem Clock in Gegenphase und ausgeschaltete Segmente in Phase.

2.7.3 Test

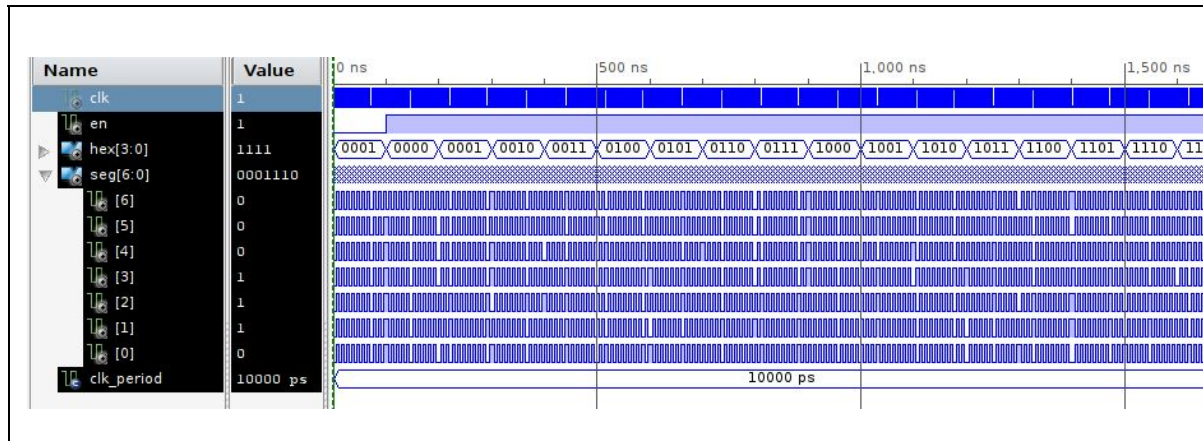


Fig. 14: Testbench des Moduls hex2seg

Man erkennt, dass je nach angelegtem Bitmuster am Eingang **hex** eine andere Kombination von Ausgängen **seg** In-Phase mit **clk** sind. Im Speziellen ist zu erkennen, dass zu keiner Zeit ein nicht-Wechselsignal auf das Display ausgegeben wird. Auch nicht bei nicht gesetztem enable-Bit **en**.

3 Diskussion

Resources Summary

Macrocells Used	Pterms Used	Registers Used	Pins Used	Function Block Inputs Used
160/256 (63%)	498/896 (56%)	92/256 (36%)	55/80 (69%)	280/640 (44%)

Gemäss Resources Summary erkennt man, dass der CPLD nicht stark ausgelastet wird. Dies ist insofern erstaunlich, da wir zusätzlich zur Aufgabenstellung einen Outputswitcher implementiert haben, welcher der Anzeige zusätzliche Modi hinzufügt. Auch haben wir eine eher komplexere Fehlererkennung mit Berechnung von Toleranzwerten und lasten damit den CPLD vergleichsweise wenig aus.

Eine Verbesserungsmöglichkeit wäre dass wir Bitbreiten von `std_logic_vectoren` und andere Konstanten im Code durch generic-Parameter ersetzen. Insbesondere bei die Bitbreiten könnte man auch durch anwenden des 2er-Logarithmus direkt berechnen lassen, was die Leserlichkeit und die einfache Adaptierfähigkeit des Codes erhöht - was bei VHDL ja ein Grundbedürfnis ist.

4 Fazit

Aus dem Projekt ziehen wir durchaus ein positives Fazit. Wir denken, dass wir alle Vorgaben erfüllen konnten und der entstandene Code schlank wie auch performant ist. Durch die Arbeit am Projekt konnten wir unser Wissen über VHDL stark vertiefen und hatten Spass bei der Lösungsfindung. Auch wirkte sich die Verwendung des Versionierungssystemes GIT positiv auf die Produktivität bei der Arbeit aus.

Durch die Modularisierung unserer Lösung könnten wir eine kleine Sammlung an nützlichen Modulen erstellen, welche auch später wiederverwendet werden können.

Termintechnisch konnten wir die Vorgaben welche wir uns gestellt haben sehr gut einhalten und hatten keine Probleme damit die Meilensteine termingerecht zu erreichen.