Technische Hochschule Nürnberg Georg Simon Ohm
Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Studienarbeit

Im Fach Smart Systems Design / Teil B (ESY1/B)

Sinusgenerator mit Lookup Tabelle

Name: Toni Sedlmeier
Matrikelnummer: 3236272

Datum der Ausgabe: 02.06.2021
Datum der Abgabe: 09.06.2021 (13 Uhr)

Prüfer:
Prof Dr. Claus Kuntzsch
: Prof Dr.-Ing. Jürgen Krumm

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Hinweis: Diese Erklärung ist in alle Exemplare der Prüfungsarbeit fest einzubinden. (Keine Spiralbindung)

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Toni Vorname: Sedlmeier Matrikel-Nr.: 3236272

Fakultät: EFI

Studiengang: BEI

Semester: 6

Titel der Prüfungsarbeit: ADC/DAC Nichtlinearitäten

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Nürnberg, 01.06.21 Sedlmeier

Ort, Datum, Unterschrift Studierende/Studierender

Inhaltsverzeichnis

1 Einleitung

FPAGs (Field Programmable Gate Arrays) bieten eine flexible und performante hardwareseitige Realisierung von logischen Funktionalitäten und sind deshalb bei Systemen, die Signale parallel verarbeiten sollen, beliebt.

Der Nachteil von FPGAs ist jedoch die Realisierung von komplexeren mathematischen Funktionen. Während die Architektur von FPGAs mit Logikblöcken auf die Abbildung von boolschen Funktionen ausgelegt ist, können komplexe mathematische Funktionen in diesen Logikblöcken nur unter großem Rechenaufwand abgebildet werden. Komplexere Funktionen sind aber bei der Verarbeitung von Signalen von großer Bedeutung.

Nicht zuletzt bei der Implementierung von Algorithmen des maschinellen Lernens, beispielsweise Neuralen Netzen, die auf hochkomplexen mathematischen Funktionen basieren, will man die parallele Verarbeitungsweise trotzdem nutzen. Eine Lösung mit diesen Funktionen umzugehen, bieten sogenannte „Lookup Tabellen“. Diese ermöglichen eine quantisierte Darstellung beliebiger auch nichtlinearer Funktionen als Wertemenge im Speicher des FPGA. [1]

2 Beschreibung des Themas

Im Rahmen dieser Studienarbeit soll eine Lookup Tabelle auf einem FPGA realisiert werden. Dazu soll der RAM des Lattice ice40 Development Board mit der Wertemenge einer Sinusschwingung befüllt werden. Anschließen sollen diese Werte mit verschiedenen Geschwindigkeiten ausgelesen werden und in ein digitalen PWM Signal umgesetzt werden.

3 Grundlegende Betrachtung der Lookup Tabellen (LUT)

Das grundlegende Konzept der Lookup Tabellen zielt einerseits auf die Implementierung von schwer in Logik realisierbaren Funktionen sowie andererseits auf die Beschleunigung von rechenintensiven boolschen Funktionen. Dazu muss die zu realisierende Funktion zunächst diskret abgetastet werden. Dabei wird eine Wertemenge generiert, welche die Funktion zu bestimmten Zeitpunkten abbildet. Diese Wertemenge wird anschließend in den Speicher des FPGA geschrieben. Hierfür können sowohl RAM als auch ROM genutzt werden.

Für das Beispiel der rechenintensiven nichtlinearen Aktivierungsfunktion Sigmoid sig(x) wurden performancetechnische Aspekte evaluiert. Es wurden Messungen für den Fall einer echten Berechnung der Funktion und für die alternative Abbildung der Funktion in einer Lookup Tabelle gegenübergestellt.

Excitation Function	Resource required in Slices			Timing Required in cycles		
	LUT	COMPUTATION	% Saving	LUT	COMPUTATION	% Saving
Log Sigmoid	281	953	70.5	25	121	79.33
Tan Sigmoid	282	1096	74.27	25	149	83.22

(ii) Log-sigmoid function

$$f(x)=\frac{1}{1+e^{-x}}$$

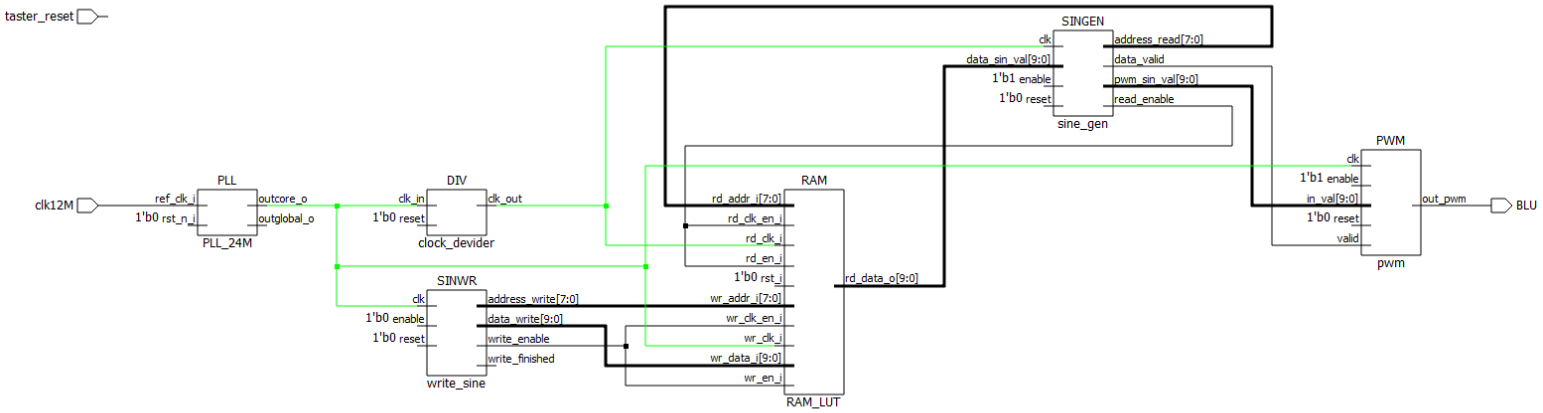
(iii) Tan-sigmoid function

$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

Abbildung 1: Performanceanalyse der LUT[2]

Abbildung 1 zeigt eine Analyse der Performance am Beispiel der Aktivierungen von Neuronen in Neuralen Netzen. Dabei wurden die beiden Aspekte Ressourcenverbrauch und Timing mit und ohne Lookup Tabelle gegeneinander verglichen. Demnach konnte durch die Verwendung einer Lookup Tabelle ca. 70% bei der logarithmischen Sigmoid Funktion bzw. 74% für die etwas komplexere Tangens Sigmoid Funktion eingespart werden. Bei der Geschwindigkeit ist dieser Effekt noch signifikanter ausgefallen. Die Anzahl der Taktzyklen für die Abbildung der entsprechenden Funktionen durch eine Lookup Tabelle gegenüber deren Berechnung konnte um 79% bzw. 83% verringert werden.[2] Aus dieser Evaluierung lässt sich ein weiterer Effekt ableiten. Die deutliche Zunahme der Komplexität einer Funktion beeinflusst die Größe der Lookup Tabelle nur in geringen Maß. Daraus folgt die Unabhängigkeit von Komplexität und Größe einer Lookup Tabelle. Das macht diese Technik bei FPGA Entwicklern so beliebt.

4 Realisierung im FPGA



Die Abbildung zeigt die Visualisierung der Toplevel Entität der logischen Schaltung. Diese wurde durch das Netlist Analizer Tool der Lattice Radiant Software aus dem Verilog Code generiert.
Die Grundarchitektur der Funktionalität beinhaltet die Module SINWR, SINGEN, PWM, RAM, PLL und DIV. Da die Module PLL und DIV intuitiv sind, wird darauf nur kurz eingegangen. Die übrigen Module werden anhand von Verilog Code genauer erklärt. Zusätzlich ist noch ein Taster vorgesehen, der in zukünftigen Versionen als Reset dienen könnte oder eine Möglichkeit zum Verstellen der Frequenz bieten könnte.

Die PLL sorgt für eine stabile Taktversorgung und erzeugt aus dem OnBoard 12 MHz Oszillator einen Takt von 24 MHz. Das Modul DIV realisiert einen parametrierbaren Frequenzteiler, der standartmäßig die 24 MHz aus der PLL auf 1/1024 teilt. Mit diesem Parameter lässt sich die Frequenz des PWM modulierten Signales einstellen.

Die grundlegende Funktionalität wird durch das RAM Interface und die Erzeugung des PWM Signals beschrieben. Dabei schreibt das SINWR Modul Werte in den RAM, während SINGEN Werte von dort liest und anschließend an das PWM Modul weiterleitet.

4.1 Erzeugung der Lookup Tabelle

Für die Erzeugung der Werte für die Lookup Tabelle wurde ein Python Script geschrieben, dass diese Werte berechnet, visualisiert und in eine Datei schreibt. Dabei werden die Werte für eine komplette Sinusschwingung berechnet. Es wäre an der Stelle auch möglich, nur eine halbe Schwingung zu berechnen und in der Lookup Tabelle zu speichern.

```
def main(argv):

    filename = sys.argv[1]

    #N = int(sys.argv[2]) # Stützstellen
    #ampl = float(sys.argv[3])



    # Generate Sine Vals
    N = 256
    t = np.arange(N)
    signal = (np.sin(2*np.pi*t/float(N)) + 1) *511
    print(signal)
    # -1 : 1 --> -1 = 0x0  : 1 : 0x3FF (1023)
    # -1:1  +1 * 511)

    # Show generated Signal
    #plt.plot(signal)
    #plt.show()
```

Der Dateiname kann beim Aufruf des Scripts als Parameter übergeben werden.

Die Anzahl der Stützstellen und Auflösung der Amplitude können durch entsprechende Abänderung des Programms auch verstellt werden. Es wäre auch möglich andere Signalformen oder mathematische Funktionen zu generieren.

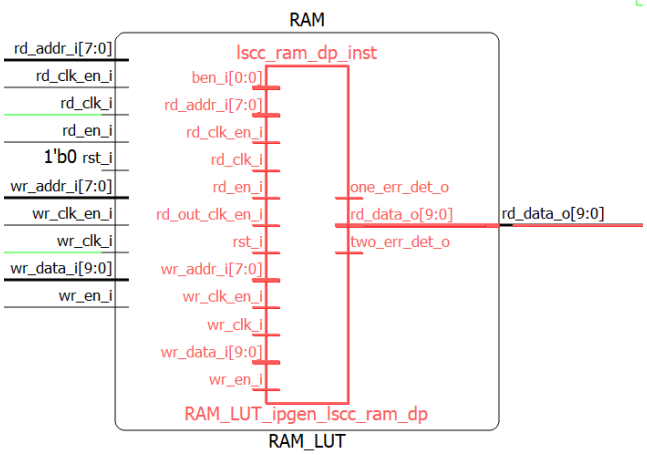
Anschließend kann bei Bedarf das erzeugte Signal noch visualisiert werden.

Zuletzt werden die Werte der Reihe nach in hexadezimaler Darstellung in die Datei geschrieben. Der Ausdruck [2:] bewirkt das Verschwinden des „0x“ vor dem Wert, dass standartmäßig bei der hex() Funktion mit auftaucht.

```
36      f = open("C:\\Users\\toni-\\OneDrive\\Dokumente\\Semester6\\ESY1_B\\Studienarbeit\\{}".format(filename), "w")
37      for i in range(0,N):
38          f.write("{}\n,".format(str(hex(int(signal_bytes[i]))[2:])))
39
40      f.close()
```

4.2 Interface des RAM

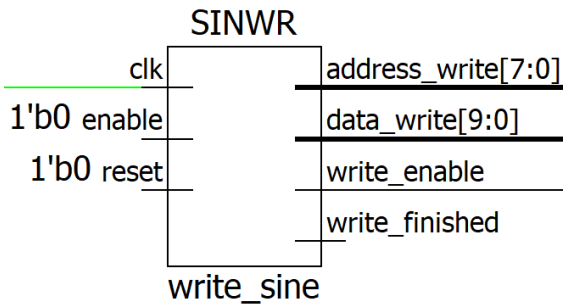
Zunächst wurde ein IP-Core eines RAM innerhalb der Lattice Radiant Software erzeugt.



Der IP-Core bietet die Möglichkeit zwei verschiedene Taktfrequenzen für das Lesen bzw. Schreiben anzulegen. Das ist sehr praktisch, da man grundsätzlich bei LUT mit einer anderen Geschwindigkeit schreibt als man liest. Die Anzahl der Adressen und Breite der Daten ist nach der Erzeugung des Cores leider fest. Für dieses Beispiel wurden 256 Adressen zu je 10 Bit Datenbreite gewählt. Daraus ergibt sich ein Speicherverbrauch von 256 Kbit.

Um nun in den Speicher zu schreiben bzw. von dort zu lesen muss zunächst ein Takt angelegt werden und dieser mit dem entsprechenden enable Signal aktiviert werden. Danach kann eine Adresse angelegt werden. Bei Schreiben muss zusätzlich der gewünschte 10 Bit Wert in das wr_data_i Register geschrieben werden. Durch Aktivieren des wr_en_i Signals wird dieser Wert dann an die entsprechende Adresse des RAM geschrieben. Beim Lesen folgt nach Anlegen der Adresse die Aktivierung des rd_en_i Signals, was dem RAM signalisiert, das jetzt gelesen werden soll. Der Wert der Adresse liegt dann einen Takt später am 10 Bit Register rd_data_o an und kann weiterverarbeitet werden.

Für das Schreiben der Werte in den RAM ist das SINWR Modul zuständig.

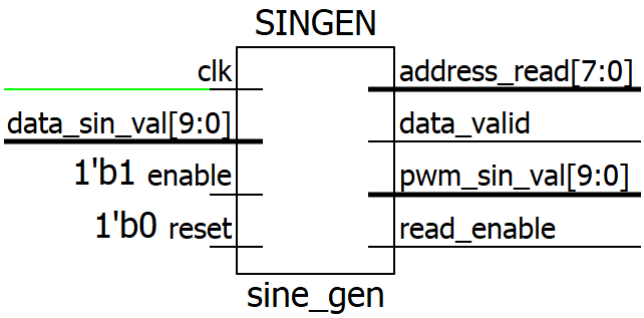


Das Modul bekommt einen Takt und kann durch das enable Signal aktiviert bzw. durch das reset Signal zurückgesetzt werden. Der Ausgang write_enable signalisiert dem RAM, dass geschrieben werden soll. Die Register address_write und data_write enthalten entsprechend deren Namen Informationen für den RAM. Das Signal write_finished teilt den umliegenden Modulen mit, dass der Schreibzyklus beendet ist und ab jetzt gelesen werden kann.

```
initial begin
  write_finished <= 1'b0;
  address_write <= 8'h00;
  data_write <= 10'b0000000000;
  write_enable <= 1'b0;
  i = 0;

  // Fill intern Array with write values :
  // Path needs to be absolute
  $readmemh("C:\\lscc\\radiant\\2.2\\projects\\sinewave_generator\\source\\impl_1\\ram_sine_vals.mem", sine_vals);
 end


// Fill RAM with Sine Values
// Values generated by python script
 always @(posedge clk ) begin
    if(reset) begin // Synchronous Reset
       address_write = 8'h00;
       data_write = 10'b0000000000;
       write_enable = 1'b0;
       write_finished = 1'b0;
   end
   else if(enable) begin
       write_enable = 1'b1;                    // enables RAM write
       data_write = sine_vals[i];              // Write Value to RAM
       address_write = address_write + 1'b1;   // Next address
   end

   if(address_write == 8'hFF) begin
     write_finished = 1'b1;                    // Signals other Moduls that writing is finished now
     write_enable = 1'b0;
   end
   i = i +1;
 end
```

Das Modul SINWR liest die zuvor erzeugte Datei ein und speichert diese in einem lokalen Array sin_vals. Das Modul wird durch die Toplevel Entität aktiviert, indem das enable Signal auf 1 gezogen wird. Dann wird bei jeder positiven Taktflanke das write_enable Signal des RAM aktiviert, was dem RAM mitteilt, dass jetzt geschrieben wird. Dann wird der entsprechende Wert an die erste Adresse geschrieben. Anschließend wird der Adresszähler und der Arrayindex inkrementiert. Das wird so lange ausgeführt, bis der Adresszähler an der letzten Adresse angekommen ist. Das Modul signalisiert den umliegenden Blöcken mit dem Signal write_finished, dass es fertig mit dem Schreiben ist.

Das SINGEN Modul kann Werte aus dem RAM lesen und bei Bedarf vorverarbeiten.



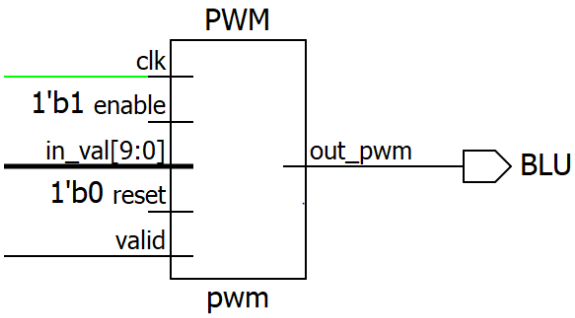
Das Modul erhält einen Takt und kann durch das enable Signal aktiviert bzw. durch das reset Signal zurückgesetzt werden. Das read_enable Signal und address_read Register enthalten die Interaktion mit dem RAM. Das Register data_sin_val enthält den Wert der entsprechenden Adresse und pwm_sin_val liefert diesen Wert weiter an das PWM Modul. Der Ausgang data_valid signalisiert dem PWM Modul, dass jetzt valide Daten anliegen.

```
initial begin
  address_read = 8'h00;
  read_enable = 1'b0;
  data_valid = 1'b0;
end

  // Get Clock and rst Signal from top
  always @(posedge clk ) begin
     if(reset) begin // Synchronous Reset
    address_read = 8'h00;
    pwm_sin_val = 10'b0000000000;
    read_enable = 1'b0;
    data_valid = 1'b0;
  end
    if(enable) begin
      if(address_read) data_valid = 1'b1;       // Signals PWM Module that data is valid now
          read_enable = 1'b1;                   //signals RAM to read value
          pwm_sin_val = data_sin_val;           // Send Value to PWM Module
          address_read = address_read + 1'b1;   //next address
    end
      // Reading RAM finished
      if(address_read == 8'hFF) begin
      read_enable = 1'b0;
      data_valid = 1'b0;
      end

  end
```

Sobald das read_enable Signal auf 1 gelegt wird, liegen einen Takt später valide Daten an. Das wird durch das Signal data_valid dem PWM Modul mitgeteilt. Das Register data_sin_val ist mit dem Leseausgang des RAM verbunden und empfängt den Wert an der entsprechenden Adresse. Anschließend wird dieser Wert an das PWM Modul weitergeleitet und der Adresszähler inkrementiert. Wenn der Adresszähler an der letzten Adresse angekommen ist, fängt er wieder von vorne an, sofern in dieser Zeit das enable Signal nicht deaktiviert wurde.

4.3 Erzeugung des PWM Signals

Um das erzeugte Sinussignal aus der LUT auf einem digitalen Port des FPGA auszugeben, muss es in ein PWM moduliertes Signal umgesetzt werden. Dazu wurde das PWM Modul hinter das SINGEN Modul geschaltet.



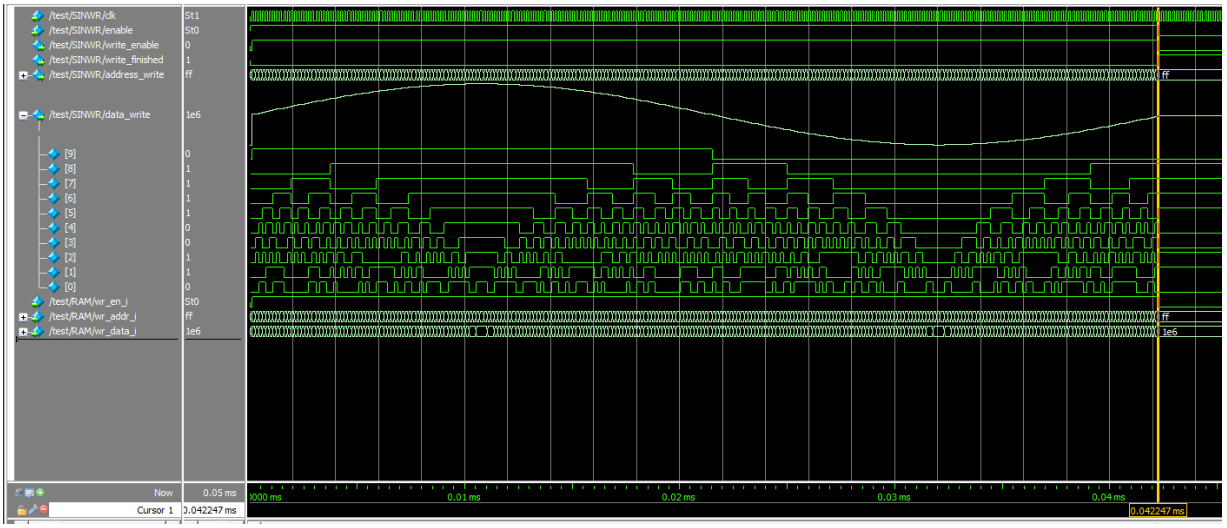
Das Modul bekommt einen Takt und kann durch das enable Signal aktiviert bzw. durch das reset Signal zurückgesetzt werden. Ein zusätzlicher valid Eingang kommt vom SINGEN (Lesen) Modul und signalisiert, dass jetzt valide Werte am Dateneingang anliegen. Der Ausgang out_pwm liefert das digital modulierte Signal in diesem Fall an die OnBoard LED. Man könnte aber auch jeden beliebigen IO Port verwenden. Dadurch konnte zunächst die Funktionalität ohne Oszilloskop verifiziert werden.

```
always @(posedge clk) begin                // Generate Sawtooth
  if(reset) begin // Synchronous Reset
          cnt = 10'b0000000000;
          out_pwm = 1'b0;
      end

  else if(valid) begin // Process digital PWM
          if(cnt >= in_val) out_pwm = 1'b0;
          else out_pwm = 1'b1;

    cnt = cnt + 1'b1;

  end
end
```

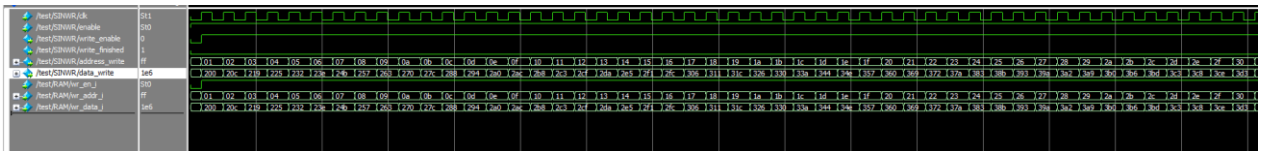
Das Sinussignal wurde dabei mit einem Sägezahn moduliert. Dazu gibt es ein Zählregister, das 10 Bit breit ist und von 0 bis 1023 zählt. Solange der Zählwert unter dem Sinuswert des gelesenen Sinuswerts liegt ist der digitale Ausgang 0. Anderenfalls wird eine 1 ausgegeben.

4.4 Verifikation in der Testbench

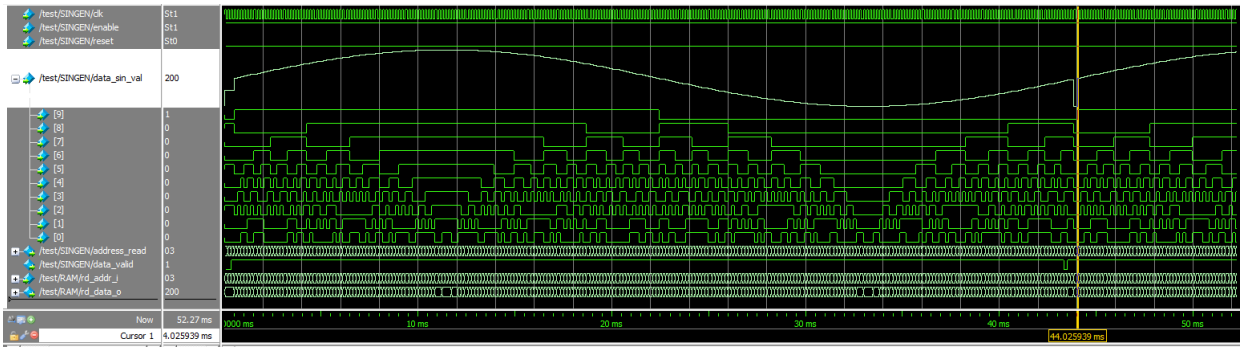
Die Simulation der Testbench in ModelSim zeigt das Verhalten sehr gut.
Hier zunächst die Simulation für das Schreiben der Lookup Tabelle in den RAM:



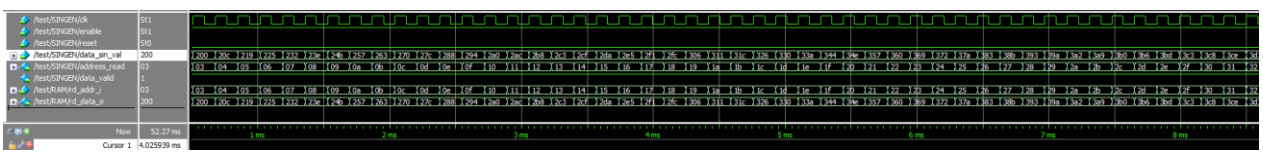
Für das Schreiben ergibt sich eine theoretische Schreibezeit von 2/12MHz * 256 = 0.043 ms
Es ist zu sehen, dass mit jeder steigenden Taktflanke der Adresszähler inkrementiert wird und der entsprechende Wert auch im RAM ankommt:



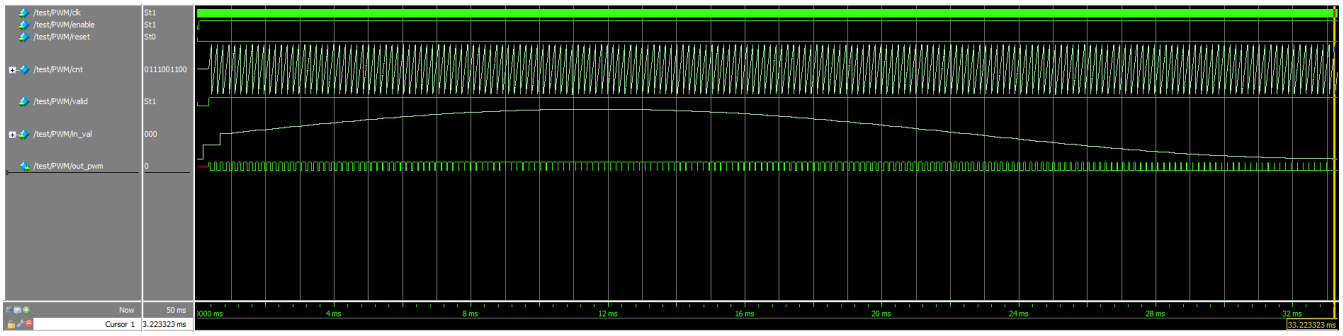
Hier das Lesen des RAM:



Für das Lesen ergibt sich eine Lesezeit von 1024 * 2/12 MHz * 256 = 43,7 ms
Daraus ergibt sich eine Sinusfrequenz von 1/43 ms = 22 Hz.
Es ist erkennbar, dass die richtigen Werte aus dem RAM gelesen werden:

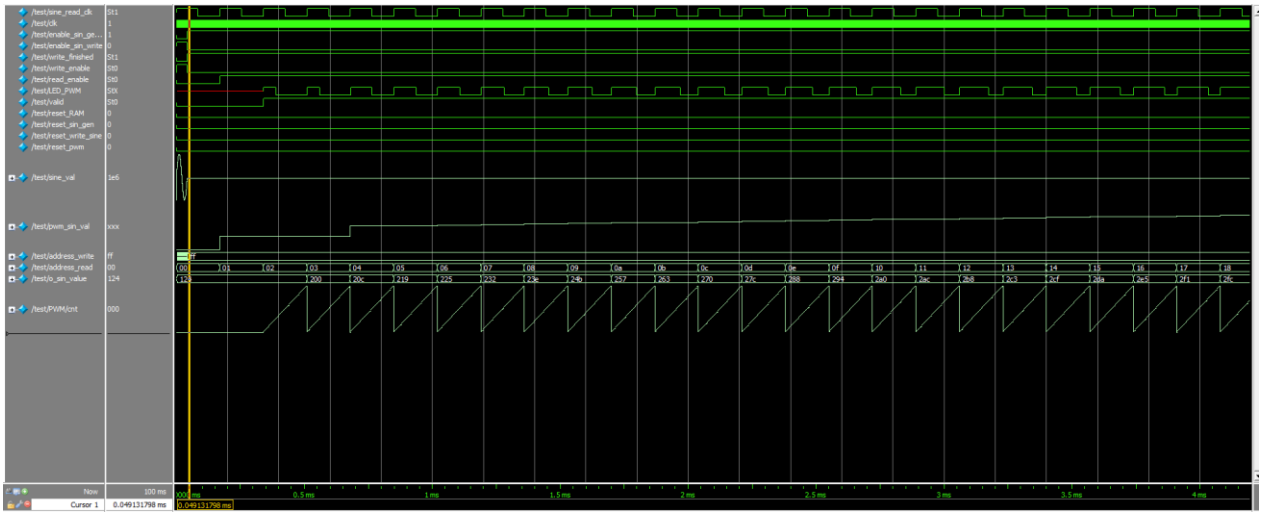


Wenn das Modul SINGEN fertig mit lesen ist, wird das valid Signal auf 1 gesetzt und die Modulation beginnt. Zu sehen ist, wie der Sinus mit einem Sägezahn moduliert wird und dessen entsprechender Ausgang als PWM Signal:



Für die Auflösung der Amplitude ergibt sich ein Wertebereich von 0 bis 1023 und damit 10 Bit. Eine Schwingung wird über die Zeit mit 256 Werten quantisiert. Daher ergibt sich für die Frequenzmodulation eine Auflösung von 8 Bit.

Das Verhalten der kompletten Schaltung ist in der folgenden Waveform dargestellt:



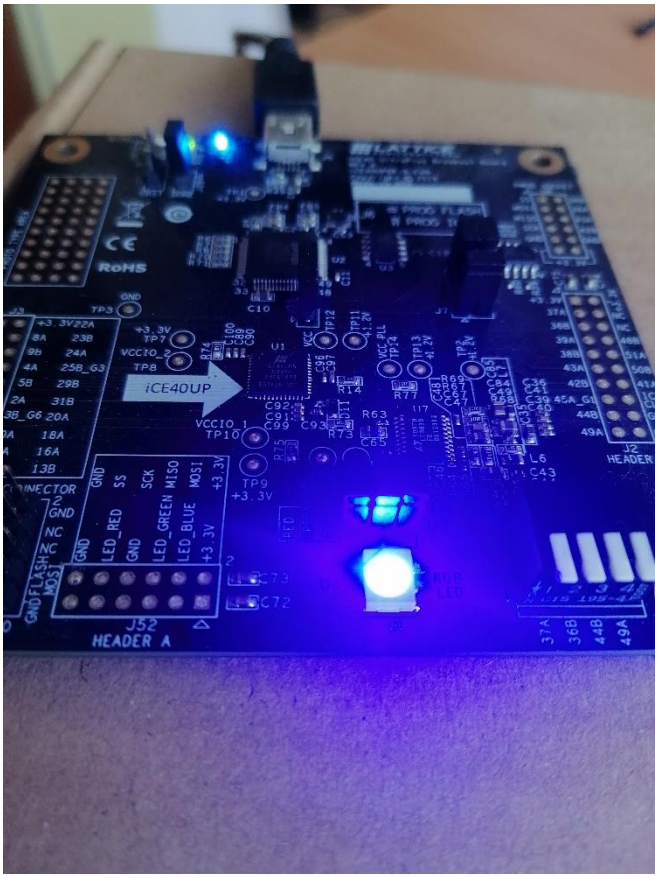
Zu sehen ist, dass bis ca. 0.05ms das System mit dem Befüllen des RAM mit der Lookup Tabelle beschäftigt ist. Nach dieser Zeit geht das write_finished Signal auf high und signalisiert dem Lesemodul, dass jetzt gelesen werden kann. Mit der nächsten steigenden Flanke des sine_read_clk wird das read_enable Signal aktiviert und mit der folgenden steigenden Flanke der erste Wert ausgelesen. Anschließend wird das valid Signal aktiviert und damit fängt auch das PWM Modul an, den digitalen Output zu erzeugen.

4.5 Verifikation auf dem Board

Da kein Oszilloskop zur Verifikation der Hardware vorhanden war, wurde das erzeugte PWM Signal über den Device Constraints Editor auf die OnBoard LED des Development Boards gelegt.

```
1  ldc_set_location -site {25} [get_ports taster_reset]
2  ldc_set_location -site {35} [get_ports clk12M]
3  ldc_set_location -site {39} [get_ports BLU]
4  ldc_set_port -iobuf {IO_TYPE=LVCMOS33 DRIVE=NA PULLMODE=100K} [get_ports taster_reset]
5  ldc_set_port -iobuf {IO_TYPE=LVCMOS33 DRIVE=NA PULLMODE=100K} [get_ports clk12M]
6  ldc_set_port -iobuf {IO_TYPE=LVCMOS33 DRIVE=8 PULLMODE=NA} [get_ports BLU]
7
```

Der Port 39 ist am blauen Eingang der RGB angeschlossen. So konnte immerhin verifiziert werden, ob überhaupt ein Signal an diesem Pin ankommt. So konnten auch verschiedene Frequenzen getestet werden, jedoch nicht quantitativ nachgemessen werden.



5 Fazit und Ausblick

Ziel dieser Studienarbeit war die Entwicklung eines Sinusgenerators. Dieser sollte auf Basis einer im RAM gespeicherten Lookup Tabelle ein digitales PWM Signal erzeugen. Bei der Beschreibung in Verilog wurde auf hohe Flexibilität geachtet. Der Adressbereich und die Datenbreite der Lookup Tabelle sind mit Erzeugung des RAM IP-Core fest. Es können aber durch den Implementierten Workflow mit dem Python Script als LUT-Generator theoretisch beliebige Signalformen und mathematischen Funktionen erzeugt werden. Für eine weiterführende Arbeit an diesem Projekt wäre eine Optimierung der Lookup Tabelle und das genau Nachmessen der erzeugten Signalform denkbar.

Quellenverzeichnis

[1] A Non-linear Approximation of the Sigmoid Function Based FPGA - Xie Zhen-zhen and Zhang Su-yu (2011) - https://link.springer.com/content/pdf/10.1007/978-3-642-25188-7_15.pdf

[2] Neural Network Implementation Using FPGA: Issues and Application - A. Muthuramalingam, S. Himavathi, E. Srinivasan (2008) - https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.307.8864&rep=rep1&type=pdf

[3] https://miscircuitos.com/sinus-wave-generation-with-verilog-using-vivado-for-a-fpga/