FPGA-based Stimulator

Felix Schmitt

Abstract

During deep brain stimulation the used wave-forms, their amplitude, frequency and the location of stimulation can affect the result in fundamental ways. The current stimulator is based on a micro-controller to control several DACs. Due to the serial architecture of µCs this approach is too slow to generate synchronously short wave-forms with high fidelity. This issue can be solved by using the parallel structure of a field programmable array to implement each channel on its own. The system consists of a µC, the FPGA and the following analog hardware to convert the signals of the DACs to currents which stimulate the target. The µCs is interacting with a PC over RS232 and provides the controls to the FPGA. It also loads the wave-forms into the memory of the individual channels.

CONTENTS

I. GENERAL STRUCTURE

Figure 1 shows the top view of the system. It consists of a computer which communicates with the Arduino via RS232. The Arduino encodes the commands of the PC and controls the FPGA. The FPGA implements a configurable number of independent channels. Each channel contains its own memories and a FSM to control the sequential transmission of the stored wavelets to the DACs. The implementation of each component is stated in the following chapters.

The output waveform of each channel consists of 5 parts as shown in figure 2. After a trigger occurred the first wavelet is outputted followed by a neutral inter-interval time. The second wavelet is outputted directly thereafter. After the end of the second wavelet, the channel can not be retriggered during the inter-period time.

II. COMPUTER

At this place, only the format to save waveforms and the function to generate them are described. The available commands are shown in the following chapter in table I. The wavelets are saved in json format because efficient implementations for different micro-controller are available and these data can be directly read and be interpreted on a computer. The json data structure has the following datafields:

Name	Type
filename	string
description	string
length	[length WF1, length WF2]
WF1	vector
WF2	vector

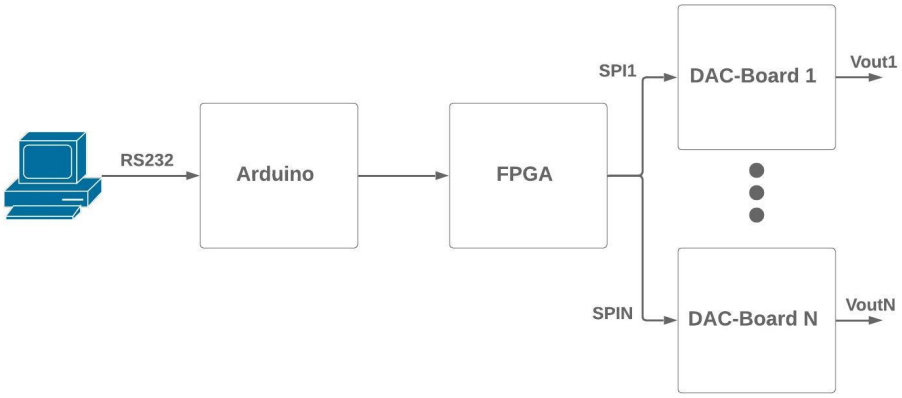


Fig. 1: Top view of system

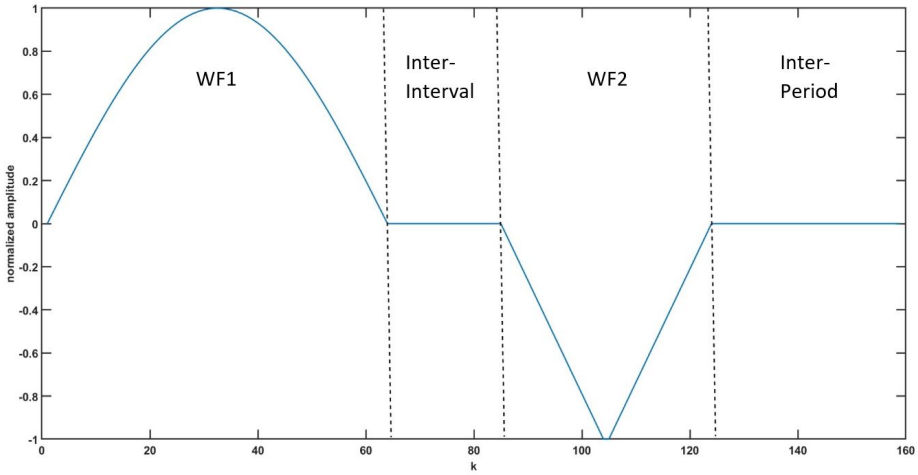


Fig. 2: Waveform

The Matlab function *encodeWave* generates a json file which can be transmitted via RS232. It needs the following parameters:

Name	Type	Description
filename	string	name of the saved file
description	string	description of the file, which can be displayed by the Arduino
WF1	Linevector(double) [-1/1]	Wavelet which will be the first part of the output waveform
WF2	Linevector(double) [-1/1]	Wavelet which will be the third part of the output waveform
wordwidth	integer [0, inf)	Wordwidth of the memory
invert	0/1	Inverts the output to match inversion of amplifier

The function transforms the values of WF1 and WF2 which are within the numeric range of -1 to 1 into the range of datawords with a wordwidth of N bits. The neutral element 0 is placed at $2^{(N-1)} - 1$. Due to implementation issues the function concatenate a neutral element to the second wavelet if their last element isn't already the neutral element. This should ensure that the stimulator outputs its neutral voltage outside of the wavelets.
Neither the Arduino nor the FPGA implements an interpolation strategy. Thus one should create the expected interpolated waveforms in Matlab and store them on the SD-Card to load them during runtime.

III. Microcontroller

The micro-controller is the interface between the computer and the FPGA. It has an SD-card slot which is used to store the waveform configurations permanently. The SD-card can be accessed by with a simple command-line interface which is implemented on the Arduino or can be directly accessed by a computer to exchange the json files. It is formated with a FAT data system. A simple command-line interface is implemented on the Arduino. It supports the following commands:

command	Description
help	Lists available commands
ls	Lists files on SD-Card
rm <file>	removes file <file>
store <file>	stores data transmitted afterward as <file>
reset <ID>	reset channel <ID>, 0:=all
WF <file><ID>	copies wavelets of <file> to channel <ID> on FPGA
trig <ID>	triggers channel <ID>, 0:=all
descrip <file>	outputs description of <file>
amp <ID><value>	sets amplitude of channel <ID> to <value>
iniv <ID><value>	sets inter interval time of channel <ID> to <value>
inper <ID><value>	sets inter period time of channel <ID> to <value>
wave <ID><value>	sets waveaddress of channel <ID> to <value> debugging
ddeb <value>	sets DOUT of Arduino to <value> debugging

TABLE I: Command list of command-line interface
Channel ID [1,2]

The program of the Arduino consists of the main file *Interface* and the Parser.cpp and the other libraries. A list of authors is appended. The Parser is edited to output the parsed datawords of the wavelets at DOUT. DOUT is port D of the Atmel SAM processor which is used on the Arduino Due model line. If the microprocessor is changed one has to adapt this code fragment. because it is controller specific. It's located in Parser.cpp.

A typical initialization of the system after a power-on are the following steps:

1) WF<file><ID>
2) amp <ID><value>
3) iniv <ID><value>
4) inper <ID><value>
5) set up other channels
6) trig <ID>

During the transfer of a file or the set up of a channel, no other command should be used. This could interfere with the data transmission.
The correct transmission patterns of the signals are described in chapter IV during the description of the system components. A good program to test the interface is Docklight Scripting. It's an easy tool to define send sequences which are transmitted via RS232. A test file is available in the git.

IV. HDL-Implementation

The general structure of the stimulator is shown in figure 6. It consists of an array of channels which are connected to the input via a multiplexer. This mux is controlled by the channel address. The corresponding file is Stimulator.vhd. It contains the main entity and the generics which control most of the settings of the stimulator. The following generics are implemented:

Name	Function
Adresswidth	Adresswidth of memories in channels
Wordwidth	Wordwidth of Din and memories
TransmitterWordwidth	Wordwidth of the Interface
MultiplierWordwidth	Wordwith of the multiplier, should be small enough to prevent overflow of output
Clock	frequency of used system clock It's just the clock frequency of the board
SPI_Clock	frequency of the interface output
NWave	Number of Waveforms to store in each channel
MaxDelay	Maximum IV- and IP-delay in periods of systemclock.
NeutralDW	neutral dataword to output during interinterval delay
NChannels	Number of channels in stimulator

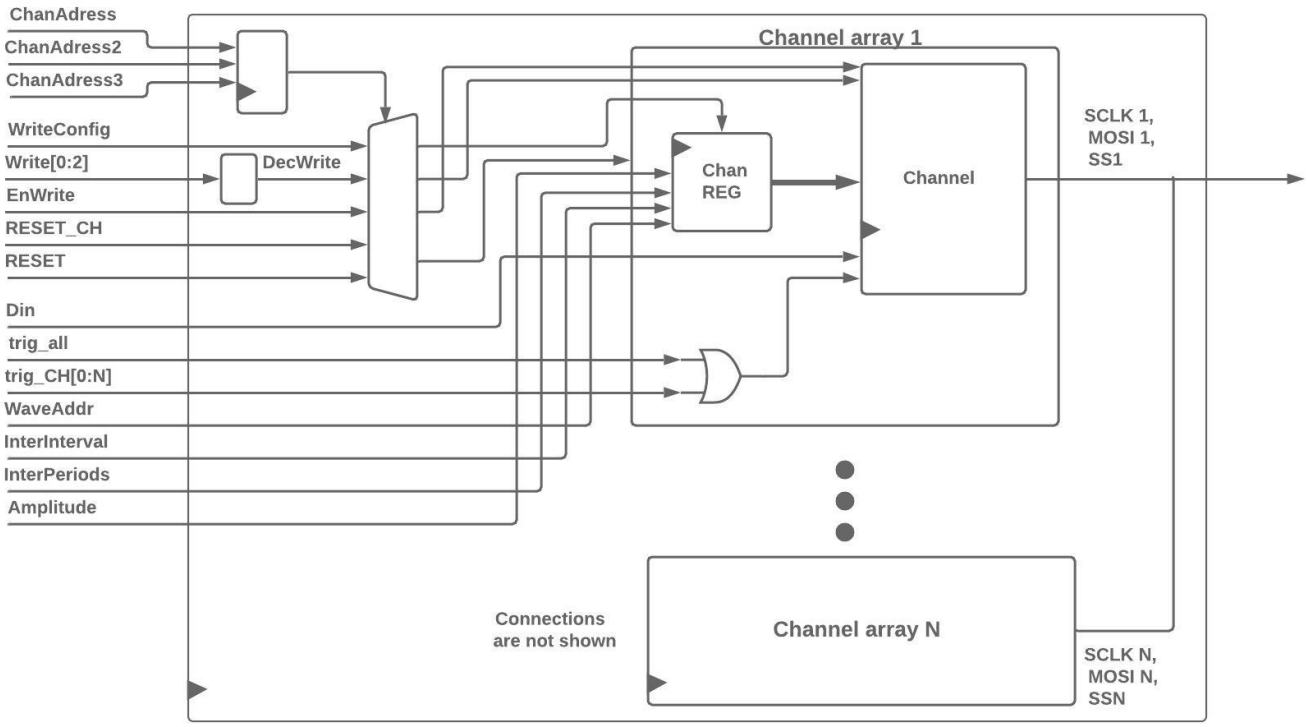


Fig. 3: Block diagram of Stimulator

The signals Write and ChanAdress have each three redundant inputs. These inputs are combined into a single signal. This signal is generated as a majority vote. This should prevent transmission errors in these critical signals for the storing of the wavelets.

A. Channel

The channel forms together with the channel register and a simple OR gate one functional unit. The detailed block diagram of the channel is shown in figure 4. The channel register stores all configuration values. These are Waveadress, InterInterval, InterPeriods and the amplitude. To store these values the right channel address at ChanAdress has to be selected and all of the previously mentioned values must have the right values. One can not store them independent of each other. The values are written to the ChanREG at a rising edge of WriteConfig.
The triggering of the channel can be done without selecting it by either a rising edge of trig all or trig_CH[ID]. Due to the OR gate, a rising edge can just be detected if the other input was reset to 0 after a trigger.
To route the signals WriteConfig, Write, EnWrite and RESET_CH to the channel, it has to be selected with the ChanAdress. The data transfer to the memory of the channel is initiated by the selection of the channel and a high signal on EnWrite. This sets the FSM to the Write mode and resets the dataword counters of both wavelet memories. The target memory is selected by storing the WaveAddr to the Channel register. Each rising edge of Write stores now a new dataword, which

is available at Din, to the target memory. The memory will count the datawords. After the first wavelet is stored, a new waveadress has to be stored in the channel register without leaving the write mode. Afterward the second wavelet can be transmitted.
The possibility to store more than one wave is the best way to switch fast between different interpolations of a waveform during run time. In the current version, this is not possible due to a missing connection from the Arduino to the FPGA, wrong generic values in the VHDL code and a missing implementation in the WF function of the Arduino.
If another DAC is used the Interface can be easily changed to another protocol. The interface has to have the ports TX_Data, TX_Start and TX_Done.
The main component of the channel is the finite state machine. Its state diagram is shown in figure 5 and the corresponding outputs in table II. The FSM is implemented as a Moore machine. This should decrease the possibility of glitches.

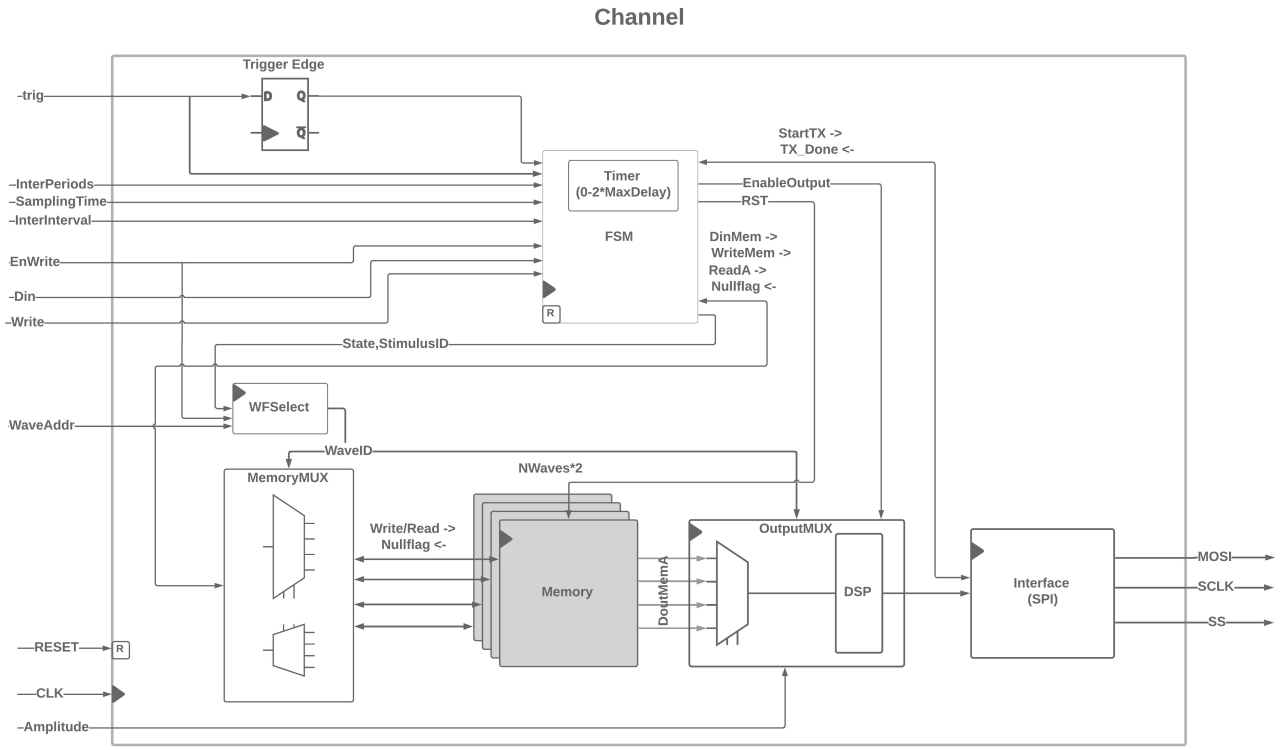
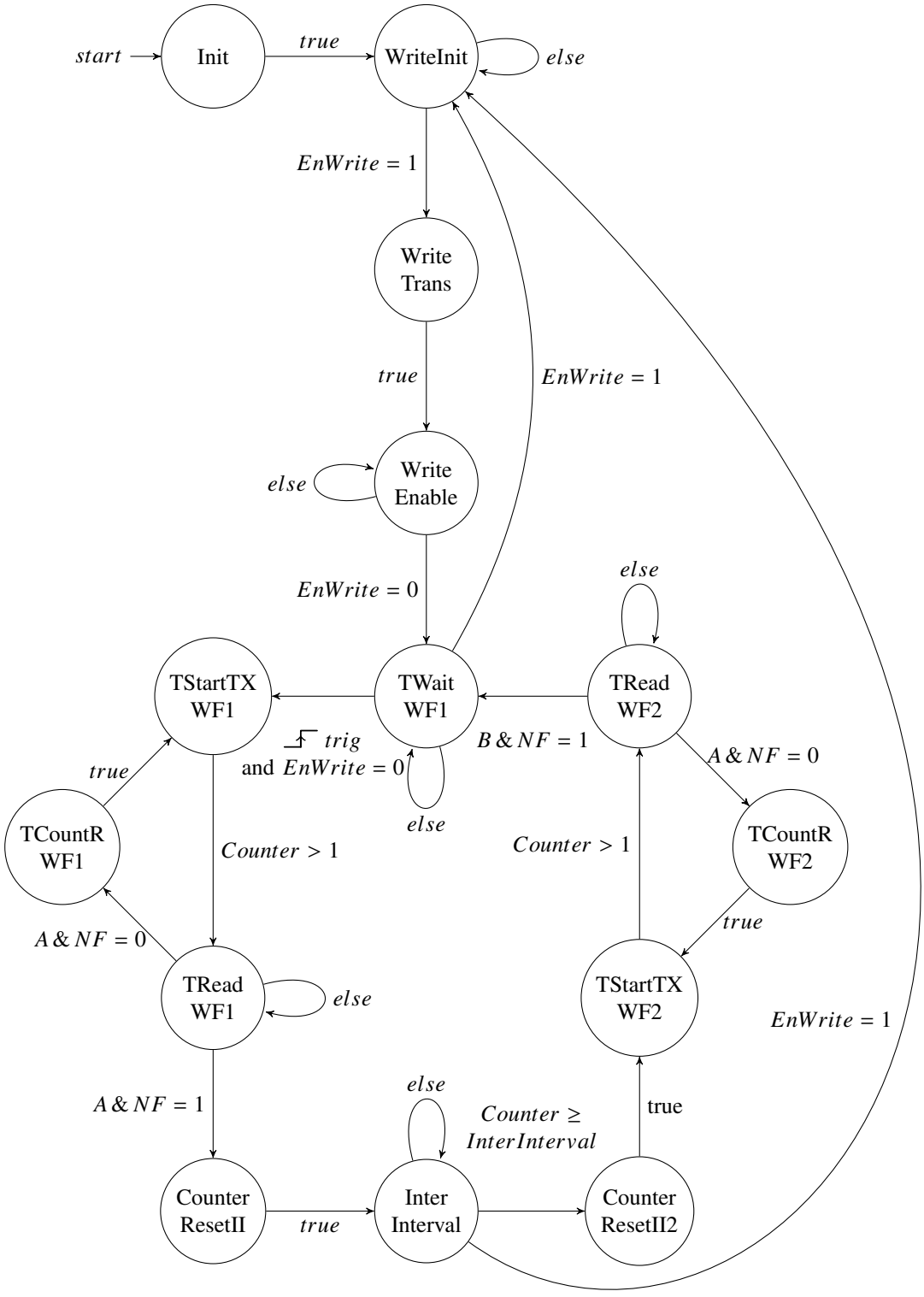


Fig. 4: Block diagram of Channel

States	Outputs							
	Read	RST	counter Rst	Write Mem	Din Mem	StartTx	Enable Output	Stimulus ID
Init	0	1	1	0	00...0	0	0	0
Write memory								
WriteInit	0	1	1	0	00...0	1	0	0
WriteTrans	0	0	0	0	00...0	0	0	0
WriteEnable	0	0	1	Write	Din	1	0	0
Waveform 1								
TWAIT_WF1	1	0	1	0	00...0	0	0	0
TStartTX_WF1	0	0	0	0	00...0	1	1	0
TRead_WF1	1	0	0	0	00...0	1	1	0
TCountR_WF1	0	0	1	0	00...0	0	1	0
CounterR_II	0	0	1	0	00...0	0	0	0
Waveform 2								
InterInt	0	0	0	0	00...0	1	0	1
CounterR_II2	1	0	1	0	00...0	0	0	1
TStartTX_WF2	0	0	0	0	00...0	1	1	1
TRead_WF2	0	0	0	0	00...0	1	1	1
TCountR_WF2	0	0	1	0	00...0	0	1	1

TABLE II: FSM: Controlunit Channel outputs



$A \coloneqq \{TXDone = 1\} \wedge \{Counter \geq SamplingTime\}$
$B \coloneqq \{TXDone = 1\} \wedge \{Counter \geq SamplingTime + Interperiod\}$
$NF \coloneqq Nullflag$
Outputs: see table II

Fig. 5: FSM: Controlunit Channel

V. Schematic

Table III shows the connections between the Arduino and the FPGA. The whole setting requires two power supplies. The FPGA needs a 9V power supply. The Arduino is supplied by the USB-Port of the computer. The DAC evaluation boards need a 5V supply.
In this project, two boards are implemented. The DAC-board which contains two DACs and interfaces the FPGA via the PMOD-header. This board was tested and didn't work. The error wasn't found and this board is replaced by the DAC evaluation board. The second board is a motherboard to host the Arduino, FPGA evaluation board and the two DAC evaluation boards. Due to the missing connectors at the bottom of the FPGA evaluation board, it has to be stacked with the top pointing down and the PMOD headers have to be soldered or connected by cables to the motherboard.

Name	**Microcontroller pin**	**FPGA pin**
WaveAddr	D64	ODH: ck_io6
ChanAddr	D66	ODH: ck_io4
ChanAddr2	D38	IDH: ck_io36
ChanAddr3	D39	IDH: ck_io37
CLK	-	Internal 100 MHz clock: sys_clk_pin
RESET	D68	ODH: ck_io13
RESET_CH	D69	ODH: ck_io12
Write	D63, D36, D37	ODH: ck_io7, IDH: ck_io34, ck_io35
EnWrite	D62	ODH: ck_io5
WriteConfig	D65	ODH: ck_io8
trig_all	D49	ODH: ck_io11
trig_CH	ch0: D51, ch1: D53	ODH: (CH0) ck_io9, (CH1) ck_io10
Din	D25(x), D26, D27, D28, D14, D15, D29, D11	JB: 0(x - Din[0]), 1, 2, 3, 4, 5, 6, 7
MOSI	-	CH0: OAH-ck_a2 CH1: IAH-ck_io22 (A8)
SCLK	-	CH0: OAH-ck_a1 CH1: IAH-ck_io21 (A7)
SS	-	CH0: OAH-ck_a0 CH1: IAH-ck_io20 (A6)
InterInterval	D10(x), D9, D8, D7, D6, D5, D4, D3	JC: 7(x - InterInterval[7]), 6, 5, 4, 3, 2, 1, 0
InterPeriods	D61(x), D60, D59, D58, D57, D56, D55, D54	JD: 7(x - InterPeriods[7]), 6, 5, 4, 3, 2, 1, 0
Amplitude	D21(x), D20, D19, D18	ODH: ck_io3(x - Amplitude[3]), ck_io2, ck_io1, ck_io0

TABLE III: Pinlist
OAH: ChipKit Outer Analog Header, IAH: ChipKit Inner Analog Header
ODH: ChipKit Outer Digital Header, IDH: ChipKit Inner Digital Header

The pins JX0-3 correspond to the pins 1-4 of the PMOD headers and the signals JX4-7 to the pins 7 to 10.

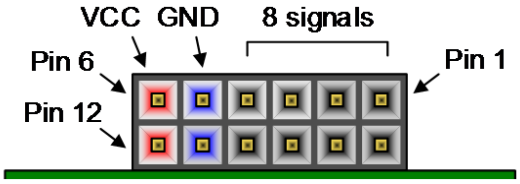


Fig. 6: PMOD connectors [1]

VI. Results

The implemented stimulator is able to output two different waveforms with completely different properties. This is shown in figure 7. Trig all is used as trigger. Both waveforms start at the moment which is hard to see because of the different lengths and the needed timescale. One of the difficulties of transmission can also be seen. In the first half of the sinusoidal waveform a bit error can be seen.

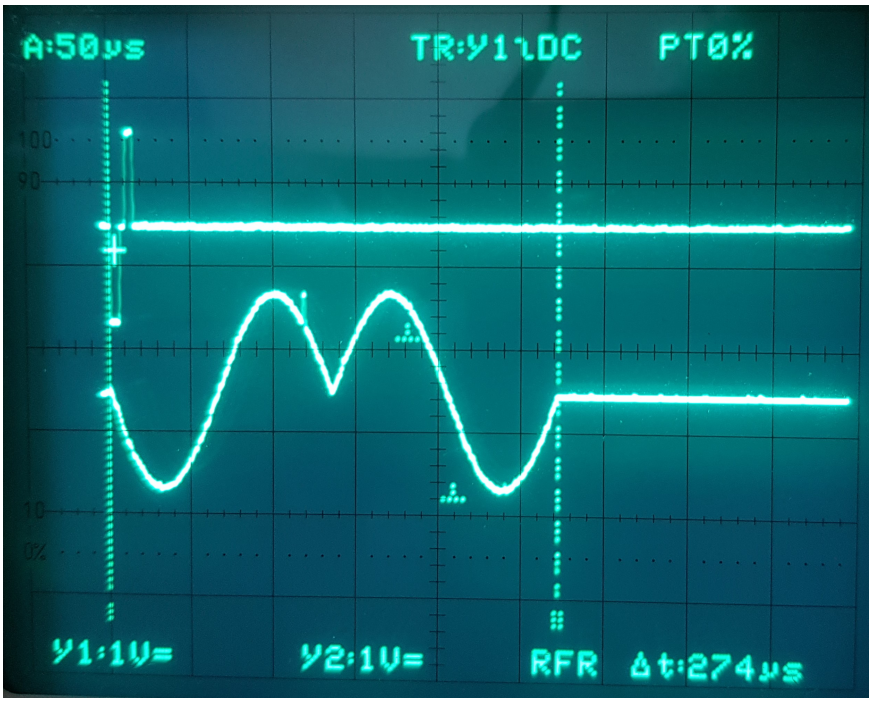


Fig. 7: Two differnt waveforms

The following figures show results that are generated by the following wavelets as shown in figure 8. WF1 and WF2

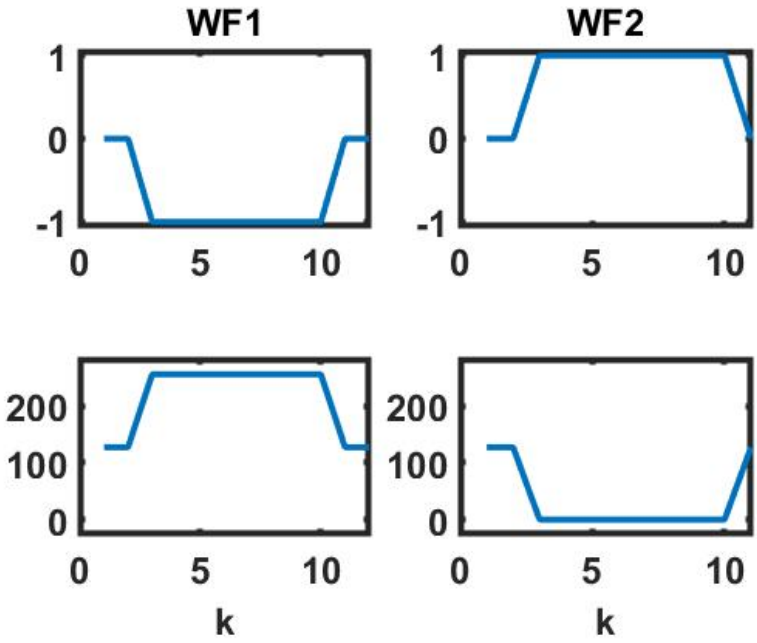


Fig. 8: Matlab generated Waveform

contain 8 times the maximum value or corresponding 8 times the minimum value. and The sample time with the 100 MHz

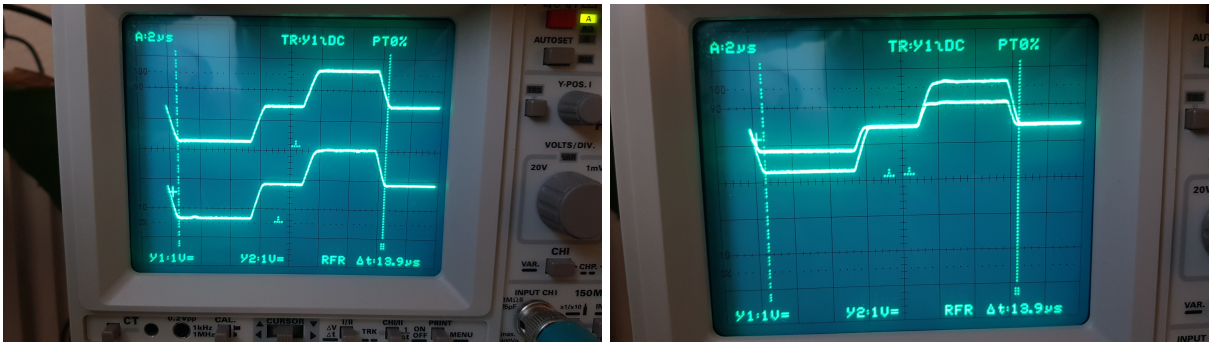


Fig. 9: Same waveform

clock of the FPGA and the SPI Frequency of 25 MHz can be determined by this waveform. Figure 9 shows the output of the system. The output frequencies are in the range of the cut-off frequency of the system. This gives rise to the round edges. The theoretical minimal sample time is:

$$t_S = \frac{16}{25\,\text{MHz}} = 640\,\text{ns}$$

This is the upper limit which isn't reached due to extra operations like changing the chip-select signal and reacting to the TX_DONE signal with the FSM. In the figure, one can see that there is a time difference of around 6 µs between the end of the falling edge of WF1 till the end of the rising edge of WF1. This should correspond to 8 outputted values. This results in a sample time of 750 ns.

The same waveform is also outputted with a different interinterval delay. This is shown in figure 11- Both channels start in the same moment but due to the increased delay, which is measured in periods of the 100 MHz system clock, the second wavelet starts later.

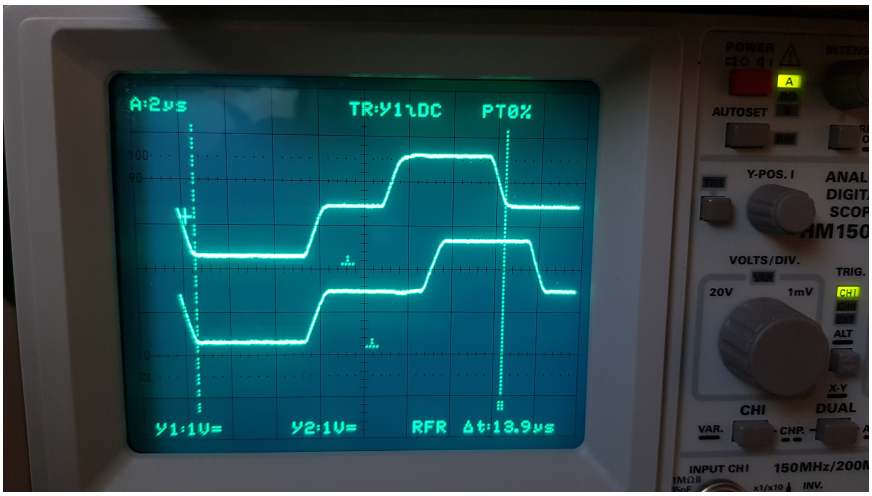


Fig. 10: Different interinterval delay

The last figure **??** shows a corrupted waveform. In one of the channels one wavelet wasn't transmitted correctly. It contains one dataword less than the other. This problem is especially problematic in short waveforms.

VII. Known issues

1) The Output of the DACs isn't set automatically to the neutral dataword after the transmission of the second wavelet. This can be fixed by inserting two extra states into the transmission between TReadWF2 and TWaitWF1. The first one has to disable the output, the second has to start the transmission.

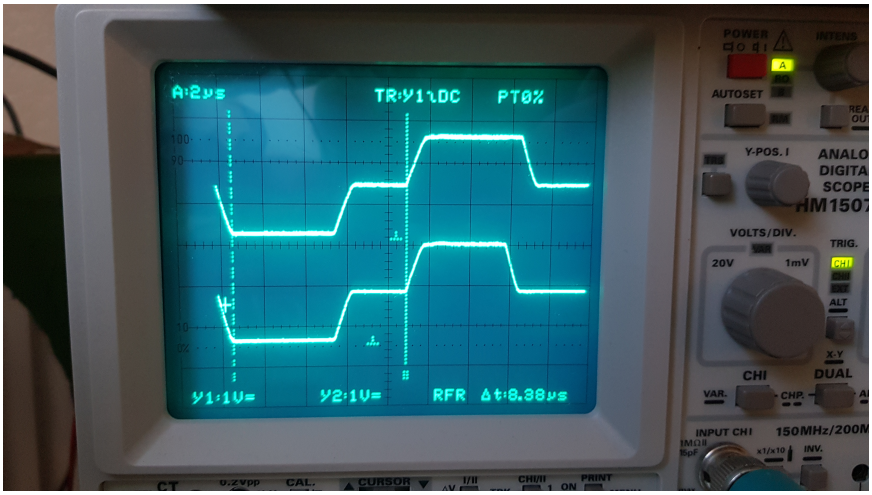


Fig. 11: Transmission error of second wavelet in second channel

2) The transmission between the microcontroller and the FPGA is error-prone. The received data words have occasionally single- or even multibit errors. An error-correcting block code should be used for the combination of Din, Write and Channeladress. This could be a hamming code, which is able to detect two-bit errors and correct single-bit errors. For an eight-bit dataword it would need four extra bits.
3) The transmission of the datafiles to the microconctroller is also error-prone. An error could be easily detected by using a check-sum that is returned to the computer.
4) The implemented simple command-line interface could be implemented on the FPGA. This would reduce the external wiring expense and properly also the need for error-correcting codes.
5) The interinterval- and interperiod-delays are really small. If a greater range is needed a clock divider as input to the delay can be used or a timer with more bits. Just the upper bits have to be connected to the input as compare value.
6) The originally intended DAC-boards are not working. Only one board was tested. I didn't find an error and switched to the evaluation boards. There is a high chance of a damaged IC on my build board. Thus it could be worth building another one and testing it again.
7) No interpolation strategy is implemented. An implementation on the Arduino needs some software changes, but it should be implementable with manageable effort. It's a lot harder on the FPGA. Here the Data source would need to store the last fetched dataword and the current and do a interpolation on both. This would change the FSM in a more fundamental way but would increase the speed during runtime.

VIII. Contributions

The following sources are used and adapted besides the standard libraries of the Arduino framework and the VHDL standard.

1) SPI-Interface VHDL: Lothar Miller, unknown license [2]

2) Simple command line interface: Stefan Kremser, MIT-license [3]

3) JSON-Parser: Daniel Eichhorn, MIT-license [4]

4) SD-FAT: Bill Greiman, MIT-license [5]

References

[1] DIGILENT. (2014). Pmod connector, [Online]. Available: https://www.jeremyjordan.me/support-vector-machines/ (visited on 11/24/2019).
[2] L. Miller. (2009). Einfacher spi-master, [Online]. Available: http://www.lothar-miller.de/s9y/archives/50-Einfacher-SPI-Master-Mode-0.html.
[3] S. Kremser. (2019). Simple command line interface, [Online]. Available: https://github.com/spacehuhn/SimpleCLI.

[4] D. Eichhorn. (2015). Json-streaming-parser, [Online]. Available: https://github.com/squix78/json-streaming-parser.
[5] B. Greimann. (2017). Arduino sdfat library, [Online]. Available: https://github.com/greiman/SdFat.
[6] R. Gray. (2013). Arduino due pinout, [Online]. Available: http://www.robgray.com/temp/Due-pinout-A4.png (visited on 11/24/2019).

APPENDIX

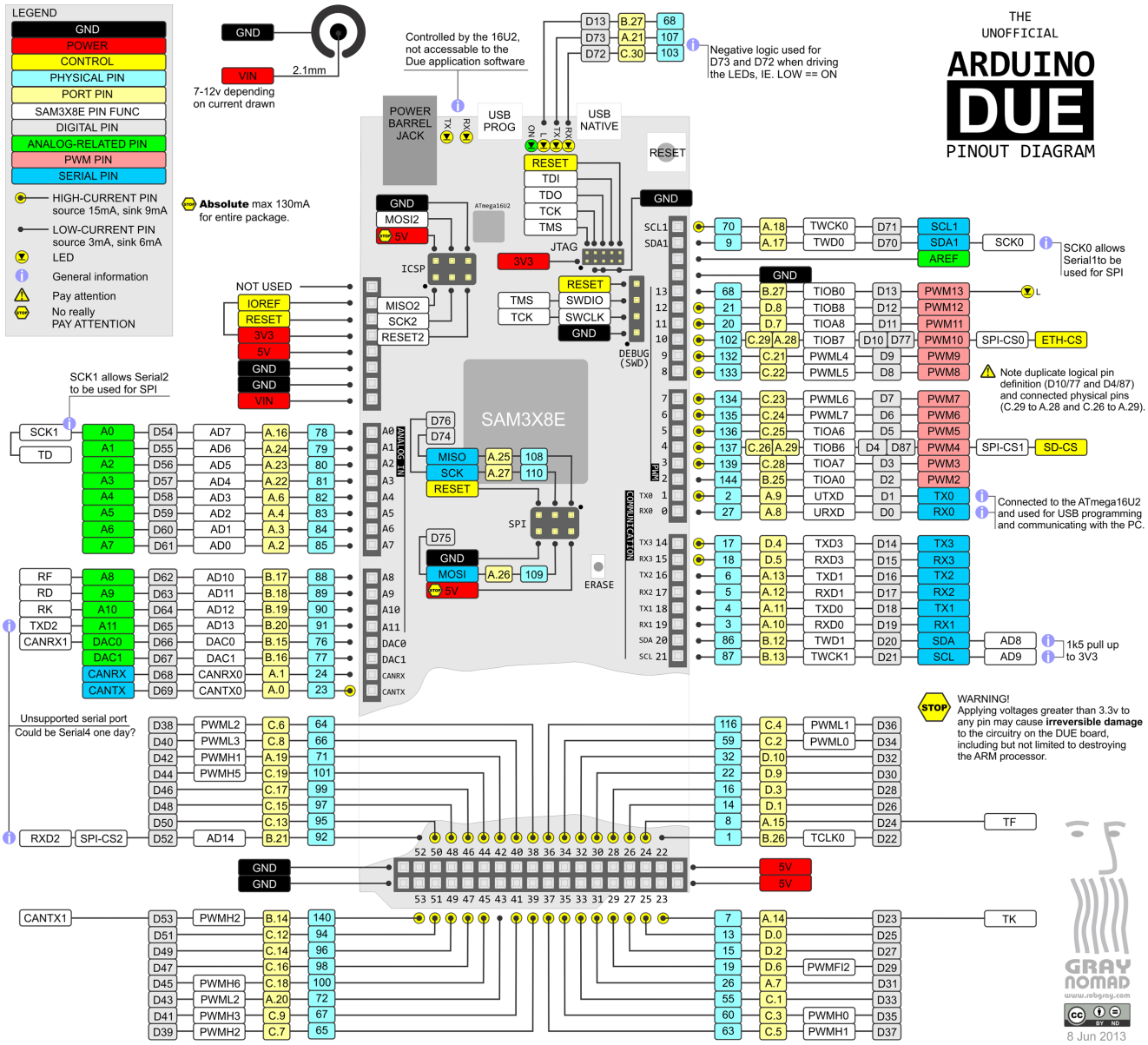


Fig. 12: Arduino Due pinout [6]