COMP22111: Processor Microarchitecture

Experimental Board Peripherals

Figure 1 illustrates the layout of the Spartan 6 laboratory board, identifying the various peripherals available.

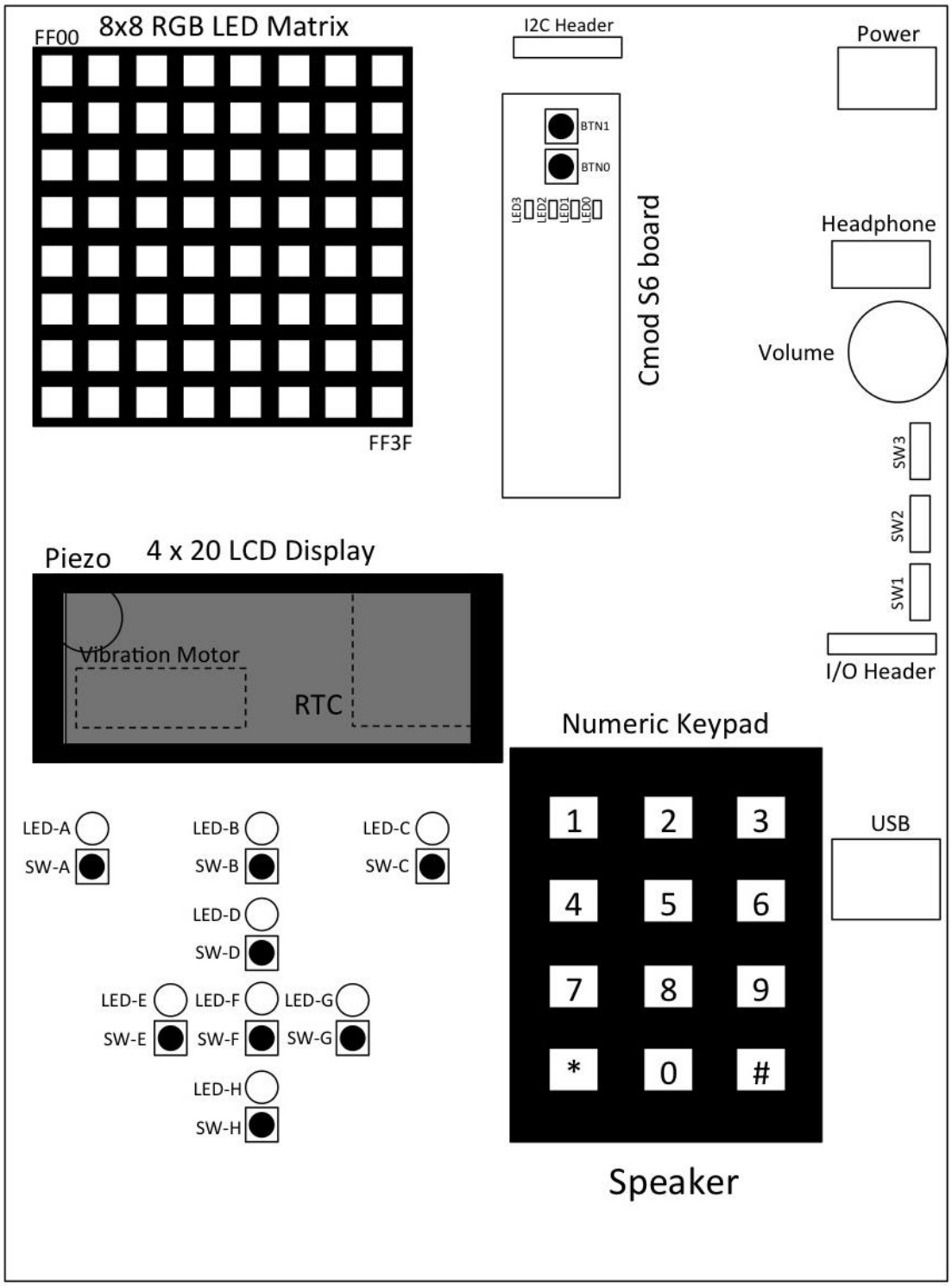

Figure 1: Layout of the Spartan 6 laboratory board

Peripherals on the board are memory mapped to addresses FF00 onwards, as listed in Figure 2. In some cases, use of the peripherals is relatively straightforward and simply involves writing to, or reading from memory locations (for example reading the status of buttons or switching on LEDs). However, in some cases the process is more complex (such as the use of the buzzer or real time clock), and may require the use of multiple addresses to perform (what appear to be) simple actions.

Memory Address	Peripheral
FF00-FF3F	8x8 RGB LED Matrix
FF40 – FF8F	4x20 LCD Display
FF90	Spartan 6 LEDs
FF91	Spartan 6 Buttons
FF92	Piezo Buzzer
FF93	Buzzer Busy
FF94	Numerical Keypad
FF95	Switches SW-A to SW-H
FF96	Vibration Motor
FF97	SW LEDs
FF98	RTC Seconds
FF99	RTC Minutes
FF9A	RTC Hours
FF9B	RTC Day of Week
FF9C	RTC Date
FF9D	RTC Month
FF9E	RTC Year
FF9F	RTC Control
FFA0	ADC Data
FFA1	ADC Control
FFA2	DAC Data
FFA3	DAC Control
FFA4	Free running counter (1MHz)
FFA5-FFFF	Reserved

Figure 2: Memory map for peripherals on the experimental board

LEDs & Switches

The memory addresses for the LEDs and switches on the board are configured as illustrated in Figure 3.

There is a numeric 12-key numeric keypad and 8 PCB mounted switches labelled SW-A to SW-H. In addition, there are an additional 2 switches mounted on the Spartan 6 PCB labelled BTN 0 and BTN1. To check if a key has been pressed you simply read from the memory address associated with the key and check the appropriate bit has been set to '1'. Note: no hardware debouncing is applied to the output of these switches.

Above each SW switch there are red LEDs labelled LED-A to LED-H, along with 4 LEDs labelled LD0 – LD3 on the Spartan 6 PCB. To turn an LED on a '1' must be written to the appropriate bit at the memory location associated with the LED.

Peripheral	Spartan 6 LEDs	Spartan 6 Buttons	Numeric Keypad	SW Switches	SW LEDs
Address	FF90	FF91	FF94	FF95	FF97
Read/Write	Write	Read	Read	Read	Write
Bit	Signal				
15:12	Not Used	Not Used	Not Used	Not Used	Not Used
11			'#'		
10			'*'		
9			'9'		
8			'8'		
7			'7'	SW-H	LED-H
6			'6'	SW-G	LED-G
5			'5'	SW-F	LED-F
4			'4'	SW-E	LED-E
3	LD3		'3'	SW-D	LED-D
2	LD2		'2'	SW-C	LED-C
1	LD1	BTN1	'1'	SW-B	LED-B
0	LD0	BTN0	'0'	SW-A	LED-A

Figure 3: Configuration of Memory addresses for the LEDs and Switches

Vibration Motor

There is a vibration motor is mounted underneath the LCD display. This is mapped to bit 0 of memory address &FF96.

Piezo Buzzer

The Buzzer, mounted below the LCD display, is a sound emitting device that is connected via a single wire. In order to be able to emit any meaningful sound, the buzzer must be turned on/off at an appropriate frequency to produce a sound wave in the human audible range. There are two modes to drive the buzzer: *Direct Control Mode* and *Programmable Mode*.

Direct Control Mode

In direct control mode the programmer generates the required pulse purely under software control, by writing to the buzzer address to turn the buzzer on and off at an appropriate rate in order to hear a sound. To activate Direct Control Mode the MSB (bit 15) in the buzzer address (&FF92) must be set to 0. Then the buzzer can be controlled by modulating the LSB (bit 0) of the same address. The period of the pulse applied will determine the frequency of the emitted sound.

Programmable Mode

In programmable mode a buzzer driver generates the pulses for some predefined frequencies that correspond to specific musical notes, as listed in Figure 4. To activate Programmable Mode, bit 15 of the buzzer memory address (&FF92) must be set to 1. Then the note and duration can be set by writing values to the appropriate bits at address &FF92 - see Figure 5.

In Programmable Mode bit 0 of address &FF93, Buzzer_busy, is used to indicate when the circuit is busy generating a pulse; when busy the Buzzer_busy bit is high. Therefore, before writing any data to address &FF92 one must check if the buzzer is no longer busy before writing the next note, otherwise the requested note will not be played. A suggested flow of operation is shown in Figure 6.

The maximum error for the note duration held by the buzzer is 1/10 seconds. The maximum error for the note frequency is 0.37Hz (on the default note C4).

Code	Note
0	C
1	C#
2	D
3	D#
4	E
5	F
6	F#
7	G
8	G#
9	A
10	A#
11	B

Figure 4: Code and corresponding musical note

Buzzer Address &FF92

Bit Number	Signals	Function
15	Mode	Controls the operation of the buzzer. Mode = 1 uses the buzzer in programmable mode and configures the buzzer circuit using the values specified for Note, Octave and Duration. If Mode = 0 then the buzzer uses direct control mode where the user can directly control the buzzer using bit 0 of Buzzer input.
14:12		Not used.
11:8	Duration	A value from 0 to 15 that determines the duration of the note in 1/10th of a second intervals. Only used in programmable mode (Mode = 1).
7:4	Octave	A value representing the octave. The higher the value, the higher the octave and thus the pitch of the notes. An increase by 1 octave is equivalent to an increase by a factor of 2 of the note frequency. The buzzer is able to emit in frequencies from the 4th to the 9th octave. Values from 1-3 will default to the 4th octave. An octave value of 0 or values higher than 9 will result in the buzzer "playing" a silent pause for the specified note duration. Only used in programmable mode.
3:0	Note	A value from 0 to 11 representing the 12 semitones in an octave. See Figure 4 for corresponding note. Values outside this range default to 0 (note C). Only used in programmable mode (Mode = 1). If the buzzer is set to direct control mode (Mode = 0) then bit 0 is used to control the buzzer, in which case bits 3:1 are ignored.

Buzzer Address &FF93

Bit Number	Signals	Function
15:1		Not used
0	Buzzer_Busy	A value of 1 signifies that the buzzer is busy generating a note in programmable mode.

Figure 5: Buzzer control bit map

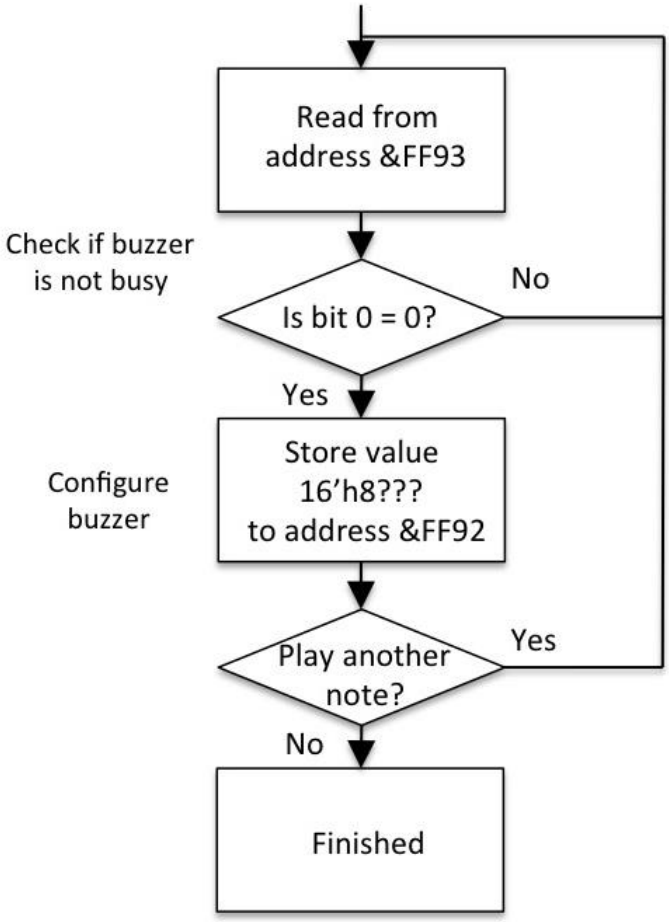

Figure 6: Using buzzer programmable mode

8 x 8 LED RGB Matrix

An 8x8 RGB LED matrix is used. It has a total of 64 cells, each one consisting of a red, a green and a blue Light Emitting Diode (LED). Each cell is mapped to a memory location as shown in Figure 7.

		Column Pixel			
		1	2	…	8
Row Pixel	1	FF01	FF01	…	FF07
	2	FF08	FF09	…	FF0F
	…	…	…	…	…
	8	FF38	FF39	…	FF3F

Figure 7: 8 x 8 LED RGB LED cells map to memory address

For each LED cell a total of 8 bits are utilized to define the colour and intensity, from the 3 colours in each pixel. Figure 8 shows which bits at the address for each pixel are used to control each colour. As the human eye is less sensitive to blue, only 2 bits are used to represent the blue, while 3 bits are used for green and 3 bits for red. The driver uses Pulsed Width Modulation to turn on/off each colour at a high frequency for the appropriate time proportion, in order to vary the intensity.

Note: using this implementation it is not possible to directly control each individual colour LED of each cell as the direct signals connected from the Matrix to the FPGA are not exposed to the programmer. Instead the underlying matrix driver translates the given colour bit values for each cell and modulates each individual LED colour accordingly.

Bits	15:8	7:5	4:2	1:0
Colour	Not used	Red	Green	Blue

Figure 8: 8 x 8 LED RGB LEDs map to colours

LCD Display

A 4 line, 20 characters per line LCD display is available. Each character position on the display is mapped to one memory location, starting from address &FF40 for the first line leftmost character up to &FF8F for the 4th line rightmost character. At each memory location 8 bits are used (bits 7:0) to allow 256 different characters to be displayed. Displaying a character is as simple as storing a code in the memory location associated with the position where the character is to be displayed. When bit 7 is set to 0 the remaining lower 7 bits are interpreted by the LCD controller according to the ASCII character standard. When bit 7 is set to 1 the LCD controller's ROM is programmed to display some custom predefined characters.

Note: the programmer is the one responsible for initializing the display when loading a program. This can be achieved by zeroing out the LCD memory mapped locations from any previous values.

I2C Devices

There is an I2C bus which is accessed through a main I2C controller residing in the memory subsystem. The RTC Clock, the DAC and the ADC all share the bus and therefore the CPU can only communicate with one of these devices at a given time. The user can practically control which device has access to the bus by setting the relevant enable flags or by updating the data in the writable memory mapped locations for the desired device. However, it is suggested that only one I2C device operation is enabled at a time and that only after ensuring that any previous I2C operation has finished, otherwise there is a possibility that results are not the ones expected. Flow diagrams are provided for the various I2C devices to illustrate a suggested mode of operation, although these are not the only possible ways of accessing the devices. In the current implementation the I2C bus runs at 1/3 of the CPU speed and can support transfer speeds up to approximately 333 Kbits/sec.

Real Time Clock (RTC)

The Real Time Clock is an independent device that keeps track of time even if the board's power is off. It has its own rechargeable battery, its own independent oscillator to count time and its own control logic and registers to store the current time. It has a resolution of 1 second and can report time, date, month, day and year with leap-year compensation up to 2100.

The RTC chip has seven internal 8-bit data registers that hold the clock information. When requested, they are read sequentially by the underlying RTC driver and the values are stored in 7 RTC data memory mapped locations, as listed in Figure 9, in the same format as they are in the RTC internal registers (Binary-Coded Decimal format). There is also an RTC driver control register mapped at address &FF9F.

To read from the RTC, the RTC Read Enable bit must be set high at address &FF9F. Then the underlying RTC driver will start reading data from the RTC chip and during reading the RTC Busy will remain high. It is the user's responsibility to clear the Read enable flag, otherwise the driver will keep reading from the RTC, the busy flag will never go low and the rest of the I2C devices will not be able to properly access the bus. When reading is done and all the RTC data mapped locations are updated, the busy flag goes low and the RTC Read Ack flag is updated ('1' for a successful read, '0' otherwise).

The user can set the clock time/date by writing to any individual or to all of the RTC memory mapped data registers at addresses &FF98 - &FF9E. While writing is being performed, the RTC Busy flag at memory location &FF9F remains high. At each of the 7 RTC memory mapped locations there is an additional read-only bit (bit 8) which functions as a Write Acknowledge flag that indicates if a requested write to the specific clock register was successful. By default, the flag is set to '1' except if a write fails then it is set to '0' and remains so until another newly requested write operation to that register is successfully performed.

Function	Address	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Seconds	FF98	Ack	0	10 Seconds			Seconds			
Minutes	FF99	Ack	0	10 Minutes			Minutes			
Hours	FF9A	Ack	0	$12/\overline{24}$	$\frac{\overline{AM}/PM}{\text{20 Hour}}$	10 hour	Hours			
Day	FF9B	Ack	0	0	0	0	0	Day		
Date	FF9C	Ack	0	0	10 Date		Date			
Month	FF9D	Ack	0	0	0	10 Month	Month			
Year	FF9E	Ack	10 Year				Year			
Control	FF9F							RTC Busy	RTC Read Ack	Read Enable

Figure 9: RTC memory mapping

The day-of-week register increments at midnight. Values that correspond to the day of week are user-defined, but must be sequential (i.e. if 1 equals Sunday, then 2 equals Monday and so on). The clock runs in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12-hour or 24-hour mode-select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit, with logic high being PM. In the 24-hour mode, bit 5 is the 20-hour bit (20–23 hours). If the 12/24-hour mode select is changed, the hours register must be re-initialised to the new format.

Figure 10 outlines the sequence of operations required to read from and write to the RTC.

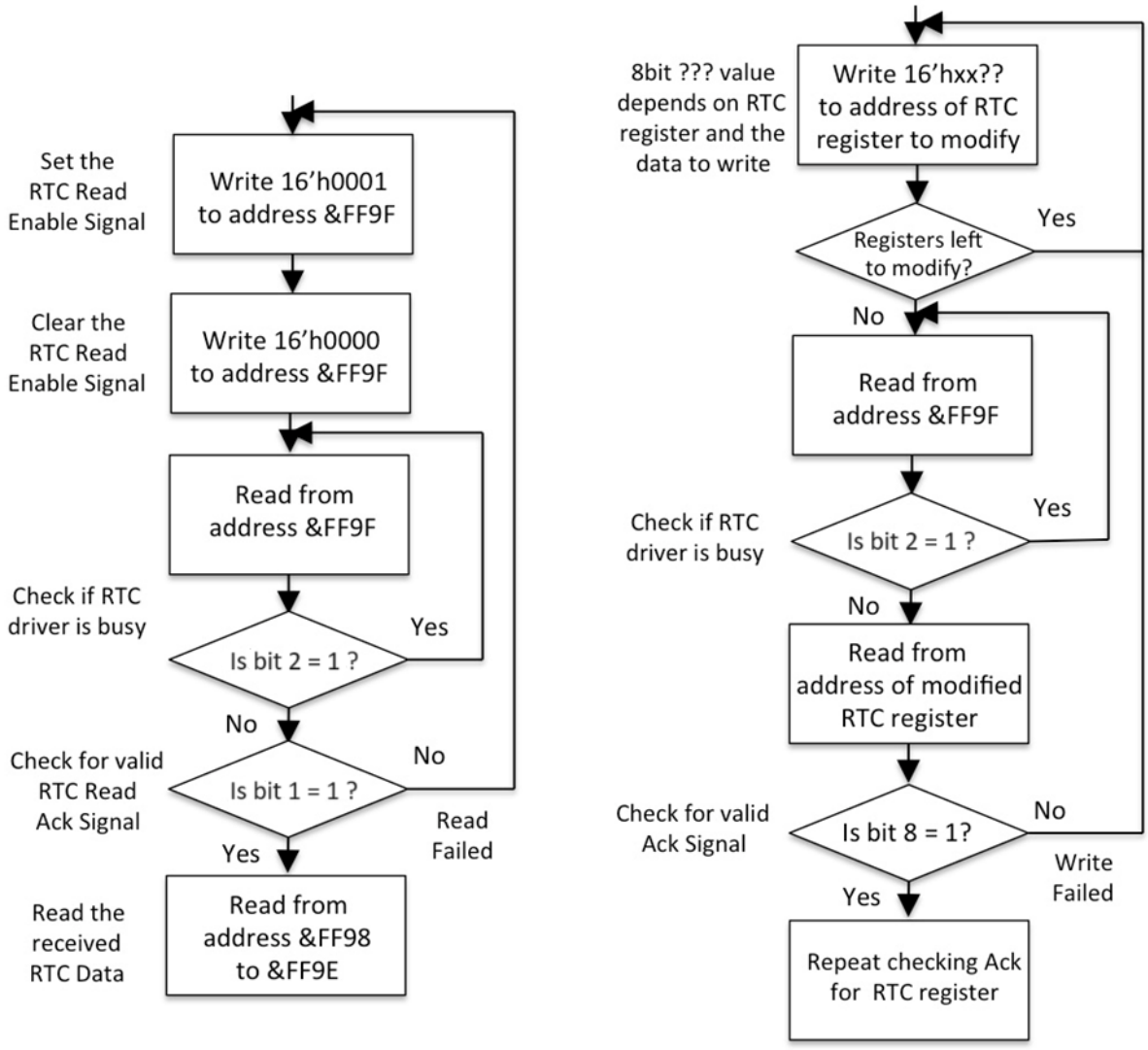

Figure 10: RTC Read (left) and Write (right) Process

Analogue to Digital Converter (ADC)

The board contains an analogue to digital converter (ADC) chip that is connected to the FPGA using the I2C bus. The ADC uses two memory locations, one to store data read from the ADC (address &FFA0) and another to control the conversion process (address &FFA1).

Memory location &FFA0 stores a 12-bit value that represents the value of the analogue voltage on the input of the ADC, as shown in Figure 11.

Bits	15:12	11:0
Function	Not used	ADC Data

Figure 11: ADC data memory location &FFA0

Bits	Signals	Function
15:3	Not used	None
2	Reading acknowledge	Flag is set to 1 when data has been successfully read and to 0 when reading has failed
1	ADC busy	Signal remains high when data is being read from the ADC and goes low when reading has finished
0	Read enable	When signal is set high, data from the ADC will be read next time the ADC is connected to the I2C bus

Figure 12: ADC control flags at memory location &FFA1

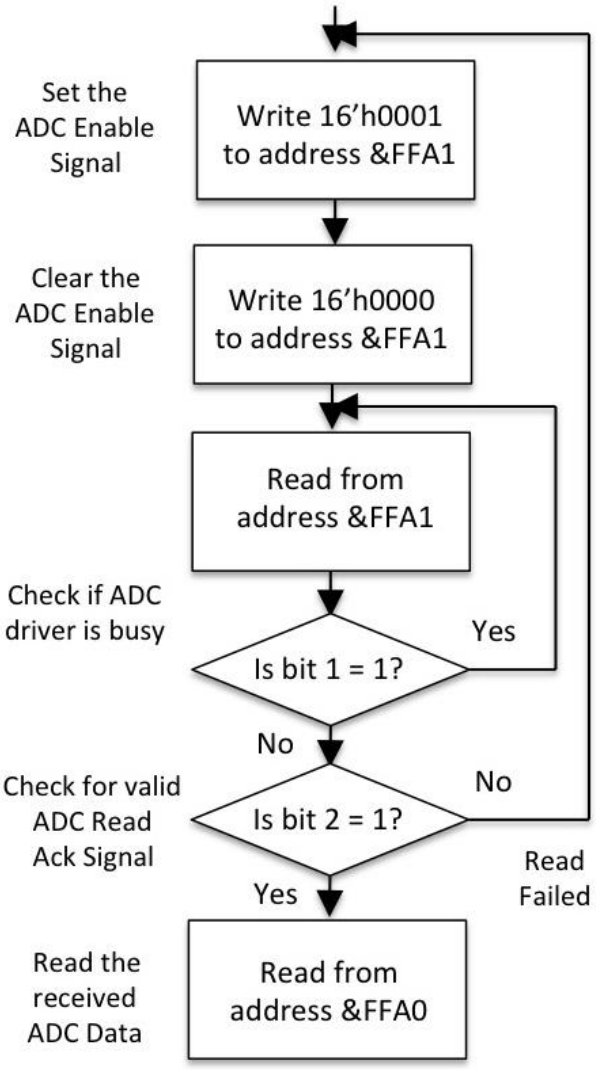

Figure 13: ADC reading process

The second memory location, &FFA1, controls the conversion process using 3 bits, as shown in Figure 12.

Figure 13 outlines the process for reading from the ADC. To start reading, bit 0 of &FFA0 has to be set high. Then the busy flag will be set to 1 by the driver and it will remain in this state until reading has finished when it will go low. After the reading has been taken bit 2 indicates if data in register has been read properly.

Note: it is the programmer's responsibility to clear the read enable flag, otherwise the driver will keep reading from the ADC and busy flag will not go low.

Digital to Analogue Convert (DAC)

A Digital to Analog Converter (DAC) converts an input digital signal (a binary number) to a voltage signal output. This is connected to the FPGA through the I2C bus.

The DAC data is mapped to memory address &FFA2 as shown in Figure 14. When the user writes a value to the first 12 bits, it is converted to an analogue output voltage by the DAC. Also, there are DAC read-only flags mapped to memory location &FFA3 as shown in Figure 15. When data is written, the busy flag goes high and remains in that state until writing is finished. After the busy flag goes low, the acknowledge signal can be checked. '1' means

that writing was successful and '0' that something went wrong. The suggested flow is shown in Figure 16.

Bits	15:13	12	11:0
Function	Not used	Ack	DAC Data

Figure 14: DAC data memory location &FFA2

Bits	15:2	1	0
Function	Not used	DAC Write Acknowledge	DAC Busy

Figure 15: DAC control flags at memory location &FFA3

Free Running Counter

There is a 16bit free running counter, incrementing at 1MHz (the speed STUMP is running) mapped at memory address FFA4.

Audio Switches

There are 3 switches(SW1, SW2, SW3) below the volume control, that select the inputs and outputs to/from the audio devices on the board – as shown in Figure 17. The default values are all switches at setting B, where the buzzer control line from the FPGA is connected to the Buzzer device, while the output signal from the DAC(as controlled by the data written to the DAC data register) is fed back to the ADC and can be read by requesting a read from the ADC. Alternatively the DAC can be connected to the speaker/headphones, to hear the audio signal produced, or the Buzzer can be connected to the speaker/headphones. Also the ADC can get an analog input from the I/O header Pin4 (this is below SW1).

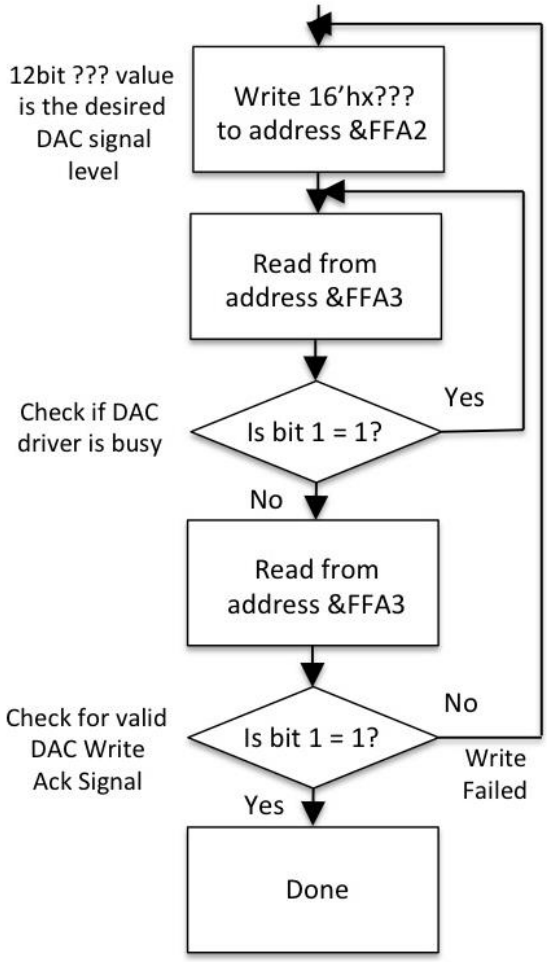

Figure 16: DAC writing process

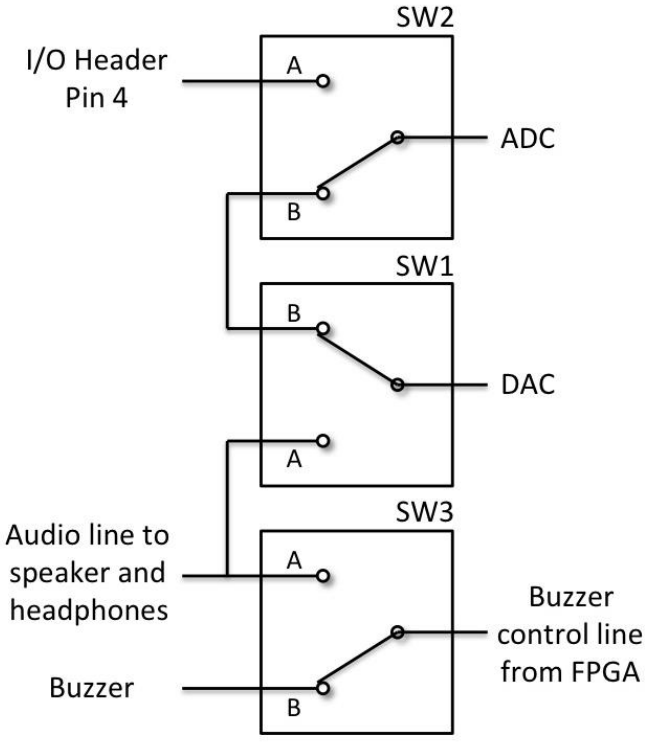

Figure 17: Configuration of Audio Switches