EE 552 Final Report
DATAD Binary Keyboard

Group Members

DENILLE GARCIA (ellined@hotmail.com)
ALY DHANJI (adhanji@ualberta.ca)
TREVOR SEMENIUK (semeniuk@ee.ualberta.ca)
AHMED ALKHATIB (aa1@ee.ualberta.ca)
DERRICKRASUGU (drasugu@ualberta.ca)

Abstract

Portable has become a common word in today's society. A portable laptop computer gives people access to familiar desktop functions anywhere. But sometimes the use of a laptop far exceeds what is required. So people use portable pocket PCs, small devices usually powered by Windows CE to give the resemblance of the familiar computer screen. For example the popular Palm Pilots. The problem is, the word portable usually also means sacrifice. Inputting information into these devices tends to be clumsy and tedious. One existing solution is the fold out keyboard. The disadvantage to this method is that the keyboard requires a flat surface, which usually limits the mobility of the user.

The solution DATAD has developed is an apparatus with the ability to interface into such devices but without the loss of mobility. The binary keyboard is lightweight, shaped in a familiar form, and has a sufficient number of input switches to allow for character representation of all letters in the English alphabet, as well as additional obscure characters and commands such as the ",", ".", "SPACE", "Back-SPACE", and "carriage return". The prototype developed contains certain limitations due to time constraints experienced during the design. First, the device does not incorporate the "carriage return" feature. Also the system only uses upper case letters. Although the opportunity still exists to add these features, it was determined not to add them to the first generation device due to problems experienced during implementation. Also the device will not be completely mobile, mainly due to the fact that the FPGA used is not mobile.

Binary Keyboard Datasheet

Description

The binary keyboard provides an alternative method to interface with pc-type devices. The primary target for the device is the pocket PC and organizer market but it is not limited to this particular family. All coded aspects of the binary keyboard are incorporated into two packages: char_display.vhd and transmitter.vhd. The first package ties the keyboard in with the VGA and the second with the PS/2. For proper operation of the device all portions of the package must be compiled and loaded into an FPGA. The keyboard can then be used for character input in replace of existing methods, such as script recognition or portable standard keyboard devices.

Features

1. Six-bit resolution allows for representation for all characters of the English alphabet as well as standard computer commands.
2. Keyboard allows for interface through standard VGA port.
3. Keyboard is designed for PS/2 protocol.
4. The device is lightweight and ergonomically designed for relative comfort.

Electrical Parameters

Keyboard Supply Voltage Requirement: 5V DC

Signal Propagation

Package	Max Path	Max Time	Min Path	Min Time	Min Period	Max Speed
char_display.vhd	reset – data_out1	22.2 ns	clock – vga_red/ reen/blue	12.8 ns	112.5 ns	8.88 MHz
transmitter.vhd	Clock – data_out	27.9 ns	Clock -- PCclock	11.2 ns	60 ns	16.7 MHz

FPGA IO Signals

Signal	I/O	Description	# of Pins Req'd
Thumb	Input	Binary thumb switch	1
Index	Input	Binary Index Finger switch	1
Middle	Input	Binary Middle Finger switch	1
Ring	Input	Binary Ring Finger switch	1
Little	Input	Binary Little Finger switch	1
Toggle	Input	Binary Toggle Switch (Caps Lock)	1
VGA	Output	Output Display	5
PS/2	Output	PS/2 Interface	2

Table 1: FPGA IO Signals

FPGA Logic Usage

FPGA logic block usage was estimated using MAX PLUS II.

Architecture	Number of Blocks Required	% Used
Data Control	230/1152	20.0%
VGA	691/1152	60.0%
PS/2	196/1152	17.0%
Total	1117/1152	97%

Table 2: Logic Block Usage

FPGA Pin Assignments

Signal	I/O	Description	Pin Assign.	UP1 pin
+5 V	Common	5 V DC Power Out	---	
Gnd	Common	Ground	---	
clock	Common	Common Clock	91	Global clk pin
reset	Common	Common Reset	28	PB 1
Thumb	Input	Binary thumb switch	45	15
Index	Input	Binary Index Finger switch	48	17
Middle	Input	Binary Middle Finger switch	50	19
Ring	Input	Binary Ring Finger switch	53	21
Little	Input	Binary Little Finger switch	55	23
Toggle	Input	Binary Toggle Switch (Caps Lock)	N/C	N/C
vga_red	Output	VGA Red data	236	VGA connector
vga_green	Output	VGA Green data	237	VGA connector
vga_blue	Output	VGA Blue data	238	VGA connector
vga_h_sync	Output	VGA horizontal sync	240	VGA connector
vga_v_sync	Output	VGA vertical sync	239	VGA connector
PS/2	Output	PS/2 Interface		PS/2 connector

Table 3: FPGA Pin Assignments

PS/2 Interface

Pin	Name	Dir	Description
1	DATA	IN/OUT	ey Data
2	n/c	-	Not connected
3	GND	------	Ground
4	VCC	OUT	Power , +5 VDC
5	CLK	OUT	Clock
6	n/c	-	Not connected

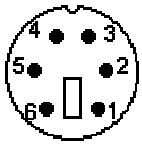

Table 4: PS/2 Pin-out [2]

VGA Interface

Pin	Name	Description
1	vga_red	Red Video
2	vga_green	Green Video
3	vga_blue	Blue Video
13	Vert_sync	Vertical Sync
14	Hor_sync	Horizontal Sync

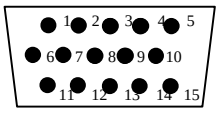

Table 5: VGA Pinouts [4]

Keyboard Interface

The keyboard interface used is connected through a 10-pin two-row ribbon cable.

Pin	Name	Description
1	Common	Ground
2	Common	Ground
3	Common	Ground
4	Common	Ground
5	Toggle	Keyboard toggle switch
6	Pinky	LSB, little finger switch
7	Ring	Ring switch
8	Middle	Middle switch
9	Index	Index switch
10	Thumb	MSB, thumb switch

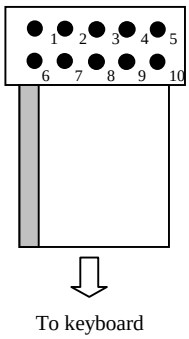

Table 6: Keyboard Interface Pinout

1.0 Introduction

1.1. Design Goal

The goal of this project is to build multifunctional input device. The device should incorporate sufficient flexibility such that the device is independent of the chosen interfacing platform. The prototype should be semi-portable and allow for simple operation for individuals.

1.2. Design

The approach that has been adopted is to produce a portable handheld binary keyboard. The device will identify a binary alphabet and output the information using ASCII code. To represent all the letters in the alphabet the binary keyboard will contain 5 switches representing each finger. With 5 switches 32 characters can be represented which is more than enough to represent the alphabet.

The system would output the information in two ways. The first method would be through a standard VGA port. The use of a VGA port allows the user of the device to connect to a standard computer monitor or any other device that has a VGA port. The output of the keyboard would then display instantly on the view screen. The second is the PS/2 interface The use of this interface allows for the system flexibility since the PS/2 standard is used in a number of different devices ranging from computers, to pocket organizers. In fact the PS/2 is so versatile that the Sony 300CD mega-changer uses the protocol to allow users to input CD titles.

2.0 Design Approach

2.1. Binary Alphabet

The binary alphabet will be implemented using 5 bits, with each bit representing each of the five fingers. This will allow for 32 different characters. The case when no buttons are pressed will be used to represent an "end of character". After each character the user will return to the "end of character" case. This will allow for the system to properly capture each character. The remaining 31 will represent characters and common punctuation or commands. Also, each character will be considered to be UPPER CASE. Lower case may be later implemented by adding an additional bit to represent CAPS lock. The binary alphabet and corresponding ASCII code is shown below in table 1.

Thumb	Index	Middle	Ring	Little	Character	ASCII
0	0	0	0	0	N/c	n/c
0	0	0	0	1	A	41
0	0	0	1	0	B	42
0	0	0	1	1	C	43
0	0	1	0	0	D	44
0	0	1	0	1	E	45
0	0	1	1	0	F	46
0	0	1	1	1	G	47
0	1	0	0	0	H	48
0	1	0	0	1	I	49
0	1	0	1	0	J	4A
0	1	0	1	1	K	4B
0	1	1	0	0	L	4C
0	1	1	0	1	M	4D
0	1	1	1	0	N	4E
0	1	1	1	1	O	4F
1	0	0	0	0	P	50
1	0	0	0	1	Q	51
1	0	0	1	0	R	52
1	0	0	1	1	S	53
1	0	1	0	0	T	54
1	0	1	0	1	U	55
1	0	1	1	0	V	56
1	0	1	1	1	W	57
1	1	0	0	0	X	58
1	1	0	0	1	Y	59
1	1	0	1	0	Z	5A
1	1	0	1	1	SPACE	20
1	1	1	0	0	BKSP	08
1	1	1	0	1	.	2E
1	1	1	1	0	,	2C
1	1	1	1	1	CR	0D

Table 1: Binary Alphabet

2.2. Binary Keyboard Design

2.2.1. Physical Design

In physically implementing the keyboard device a number of different issues must be addressed:

i. The keyboard must comfortably fit into an individual's hand.
ii. The binary keys for the keyboard should be easy for the user to press for all fingers. Since the mobility of the little finger tends to be limited on its own, the shape of the keyboard should account for this. This is a key issue in designing the physical structure.
iii. The keyboard should be lightweight. If the keyboard is too heavy, the user may become fatigued due to prolonged use.

To create the physical design of the binary keyboard the upper portion of an old joystick was used. With this method the above concerns were addressed fully. This device was shape and lightweight design created a good starting point for the keyboards. The joystick was previously shaped to fit an individual's hand, and already contained two buttons and a toggle switch. The toggle switch may be used to latter represent Caps lock. Three additional buttons were added to the device to complete the keyboard. Also the device has been initially designed for right-handed users. Changing the device would simply involve moving the buttons.

2.2.2. Keyboard Interface Design

To interface the keyboard to the FPGA an active low philosophy was taken. Each switch was tied to a common ground within the keyboard. The inputs for the FPGA were tied to a pull up resistor and a 5V power supply. When the switch is closed the power supply will be shorted to ground pulling the input voltage to zero. When the switch is open the input voltage was pulled up to +5V. In this way, floating inputs were avoided. The keyboard circuit is located below in figure 2.

3.0 Software Design

The driver for the keyboard was separated into three main stages: data handling, and data output. **Section 6.0** contains a more detailed description of the VHDL code.

3.1. Data Handling

The primary task of the data handling stage is to process the inputs from the keyboard switches, decode the data, and transmit the data into the output streams. A key part of the data handling stage is the input stage. The data input stage must take into account that not all switches may arrive at the same time.

The following gives a detailed representation of how data is handled from input to display. This process makes use of various delay signals to load various components. To give an insight into how this design works the top-level diagram in ***Appendix 1*** will be described in the following. All the inputs into the NOR gate and the SR Latch are debounced using debouncer component. This debouncer is basically a D Flip flop that is samples input data every millisecond or 25,000 clock cycles based on a 25 MHz clock.

This design takes into assumption that there are no floating inputs. As a result when the keyboard is powered all the components reset. The SR latch resets from the output of the **NOR** gate which is high at this instance. The inputs into the register and decoder are also set to zero. Delay **D2** delays the reset signal into the SR latch long enough to clear the register and decoder. Delay **D2** also allows the system to keep the register load signal High.

The SR latch component is basically an SR flip flop that uses data in as the set signal but only has one reset signal. This reset signal is the output of the NOR gate that is delayed to ensure valid data propagates through the data path.

The idea behind using the **NOR** gate is that once the user begins pressing keys, no load signals will be activated until after all keys have been released. To clarify this idea the propagation of signals out of the **NOR** gate into the SR latch reset and the register load will be explained.

- Initially once the glove is powered the output of the **NOR** gate is high. The signal out of the NOR gate is debounced to eliminate any noise that may arise due to the switches in use. The output from the NOR gate is used to reset the SR latch and also load other components, the register and ASCII Decoder. This signal is also used to load the PS/2 controller and VGA driver. Therefore once the NOR output is high, after propagation through delay **D2**, the SR latch resets. This occurs in microseconds.

- Once data input begins the **NOR** output goes low. This change does not affect the load control as will be explained briefly. This low signal has also no effect on the SR latch. Once input is done and all switches are released, the **NOR** gate signal goes high immediately thus activating load control. This load control will be kept

high for as long as specified. It is important that the load signal goes low before the SR flip-flop resets so that the registers do not get cleared. Data is therefore loaded from the SR flip-flop once the load signal goes high.

- Once debounced, the NOR output signal propagates through delay **D2** and the register SR latch resets. It is worth noting that by the time the SR latch resets the register load signal is already low keeping the output of the register available to other peripheral component for a longer period of time.
- The register is a five-bit register. This register is loaded using an active high load signal. The register also has an active low global reset input.

- The **PS/2** interface receives its five bit data from the register. The PS/2 enable signal is a delayed and controlled version of the NOR gate output. This signal comes straight from the NOR gate to keep the PS/2 enable signal high for a relatively longer period of time (approximately 3 Clock cycles of a 25Khz Clock). This enable signal turns high only when the register data has been set. This signal should terminate in good time such that it is ensured that invalid data is not transmitted. More details about the **PS/2** are covered in the PS/2 controller section.

- The 5-bit to 8-bit ASCII decoder receives its data from the register and the enable signal is a delayed version of the register load signal, which also terminates at about the same time as the register load signal. The output of the ASCII decoder does not clear when the load signal goes low but rather is refreshed when new data arrives.

- The **VGA** interface requires an enable signal and eight-bit ASCII. The enable signal is only required to stay high for one clock cycle, because of the refresh rate of the VGA. In a situation where one clock cycle causes unstable output to the VGA the period over which the VGA load signal is high is increased, and the VGA code is modified such that it does not refresh more than once for every VGA enable signal high event. More details about the **VGA** are covered in the VGA driver section.

- The load control component controls the duration of all the load signals. The load control circuit diagram is contained in the appendix B. All the signals are active high. As a result, it is needed to load data only when load signals are high and for a specified duration. To achieve this, as can be seen in the circuit diagram an AND and XOR gate are used with one delay mechanism. Once the output of the NOR gate goes high, the output of the delay mechanism will be assumed to be low, until the input signal propagates. During this duration of time, the output of the AND gate is low, thus the output of the XOR gate will be high. Once the signal propagates through the delay mechanism the XOR output is set low. When the output from the NOR gate is low the XOR output will always be low because of the AND gate.

3.2. Data Output

The data from the decoder will be outputted with two different approaches. The first approach is the use of a standard PS/2 interface. In this way, the binary keyboard can interface to pocket PC systems or regular desktop computers. The second method will be through VGA. This method allows for direct interfacing with a monitor, or any device with a VGA port.

3.2.1. PS/2 Interface

The PS2 Controller is comprised of five parts.

1) Binary Keyboard to PS2 data decoder.
2) Scan Code generator
3) Shift Register [1]
4) Transmitter Controller [1]
5) Transmitter Component -- This integrates the parts together [1]

The PS2 keyboard and PS2 mouse both use the same type of communication method between the device and the computer. Therefore, it was possible to re-use some of the code developed for the Spatial Mouse[1]. In particular, the code used from the Spatial Mouse project is required to generate a serial bit-stream that is synchronized with a data clock used to send data to a computer.

In order for a keyboard to transmit a character to a computer, it has to generate a scan code that the computer can interpret. The scan code is comprised to two parts: a make code, and a break code. Unfortunately there is no given formula to calculate the make and break codes for any given key on a keyboard. The only way to know what scan code is associated with a certain key is to look it up on a table. However, there is a relation between the make and break code. The break code is simply the hexadecimal number $F0 followed by the make code. Whenever a key is pressed and released on a keyboard, the keyboard sends three data packets of 11 bits. Each packet contains 8 data bits, a start bit of 0, a stop bit of 1, and an odd parity bit. The first data packet is simply the make code of the key pressed. The second packet is the hexadecimal number $F0, and finally the third packet is the make code again. For example, the character “A” has a make code of $1C. Therefore, whenever “A” is pressed and released, the keyboard sends a serial bit stream to a computer which contains the data $1C, followed by $F0, and finally $1C again. Therefore, in order to implement a PS2 keyboard controller we have to ensure that we generate the scan codes for the character we wish to display, and send it serially to the computer, using the PS2 communication protocol.

In the PS2 controller, the Binary to PS2 decoder generates the make code for a character by looking it up on a table of make code values. The make code is then sent to the scan code generator that passes the transmitter controller the make code, followed by $F0, followed by the make code again. The only time the scan code generator passes the scan codes is after an enable signal is asserted. Otherwise the scan code generator simply ignores any data changes on the input.

3.2.2. VGA

The purpose of the VGA interface is to accept data from the binary keyboard, decode this data to establish the correct character that should be displayed on the VGA monitor, and then sent this character to the monitor. It also prints the project title on the monitor, above the characters received from the binary keyboard.

Three character roms and three .mif files are used to generate the title and characters. One .mif file contains the pixel definition of each character used in the project, the second .mif file contains the title message which is to be displayed. The third .mif file contains the 26 letter alphabet which is part of the binary keyboard interface, as well as a space character which functions as a backspace.

Appendix E: References

1. Spatial Mouse project. PS2 Mouse TX. April 2001.m
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/2001_w/interfacing/PS2_Mouse_TX/

2. Christopher Kowalski. E-Field Probe group. March 7, 2001.
http://www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/2000f/interfacing/keyboard/

3. Hao Luan , Bo Liu and Albert Chan . Dicerace Game code. April 9, 1998.
http://www.ee.ualberta.ca/~boliu/projects/ee552/ee552.html

4. UAB "BBD SOFT". 2000-2001. http://www.bbdsoft.com/video.html

5. Adam's Micro-Resources. 1999-1998.
http//govschl.ndsu.nodak.edu/~achapwes/PICmicro/keyboard/scancode2.html