MIPSfpga by Imagination

Lab 9

Advanced I/O: SPI LCD

These materials produced in association with Imagination.
Join our University community for more resources.
community.imgtec.com/university

 © Imagination Technologies 2017

MIPSfpga Lab 9: Advanced I/O: SPI LCD

1. Introduction

In this lab you will add another peripheral interface to the MIPSfpga system. You will also learn about the Serial Peripheral Interface (SPI) standard and about Liquid Crystal Displays (LCDs). First, you will build an SPI interface to the MIPSfpga system, and then you will use that interface to write to an LCD. At the end of this lab, you will write a C program to play a number guessing game that uses the LCD to output text to the user.

To complete this lab, you will need an LCD with an SPI input. For example, we show how to use one available from Digikey as described in Table 1.

Table 1. LCD Parts

Part	Description	Supplier	Price
LCD: Part # 1481-1063-ND	1 × 8 LCD, no backlight	Digikey: http://www.digikey.com/product-search/en?x=0&y=0&lang=en&site=us&keywords=1481-1063-ND	$15.29
1 uF capacitor (x2)	Capacitor		

2. Serial Peripheral Interface (SPI)

SPI is a synchronous serial protocol that is relatively straightforward and fast. It uses only 2-3 pins for the interface: Serial Clock (SCK), Serial Data Out (SDO), and Serial Data In (SDI). One device, called the *master* device, generates SCK and SDO. A second device, called the *slave*, accepts those inputs and sometimes also generates SDI to be fed back to the master device as an input. Figure 1 shows the connection between an SPI master and slave device and the waveforms for the master device's signals. The master communicates 8 bits of data at a time (most significant bit first) to the slave using SCK and SDO. At the same time the slave may send 8 bits of data back to the master on the master's SDI input. SCK asserts a clock edge only when SDO has valid values on it.

 © Imagination Technologies 2017

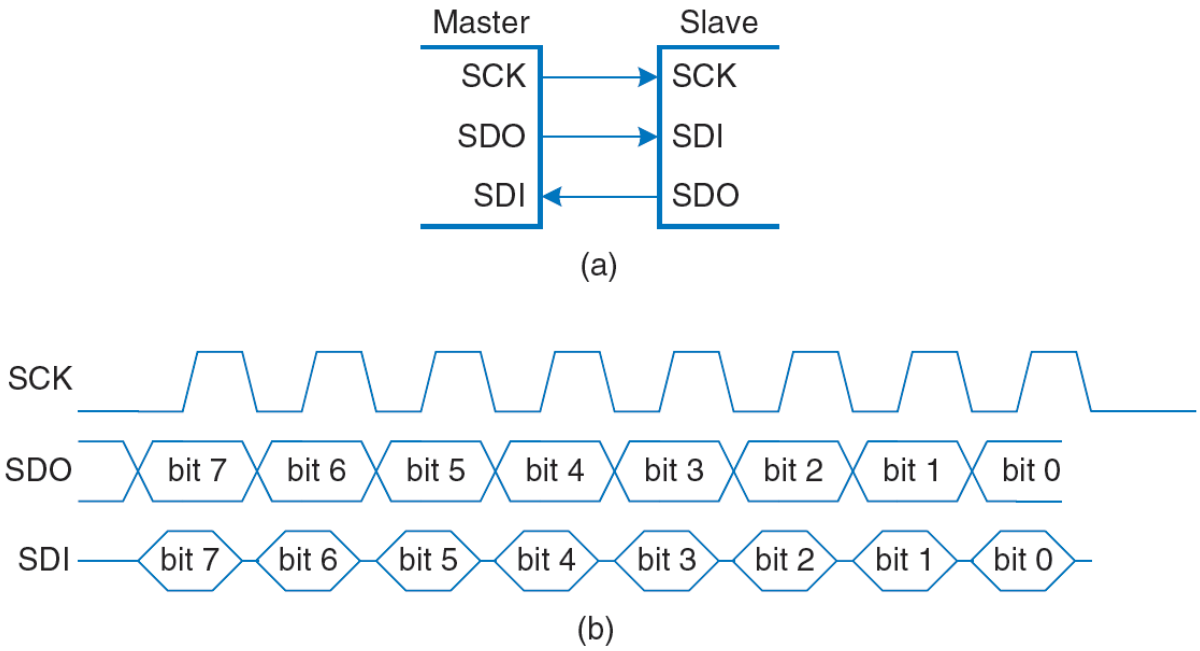

Figure 1. SPI connection and waveforms for master (courtesy *Digital Design and Computer Architecture*, 2nd Edition, Harris & Harris, © 2012 Elsevier)

In our setup, the MIPSfpga system is the master and the liquid crystal display (LCD), that we will be described later, is the slave. MIPSfpga sends values to the LCD but receives no information back from the LCD, so the SDI pin is unused.

Add an SPI interface to the MIPSfpga system by memory-mapping the 8-bit value to write over the SPI interface to address 0xbf80003c. A write to this address will initiate a transmission of the least significant 8 bits of the written value over the SPI interface. Because the transmission over the serial interface takes some time, the user must check that the transmission has completed before writing a new value to be transmitted. The SPI interface must assert a signal, SPI_DONE, when the transmission is complete.

You will need to memory-map the SPI_DONE signal to a second address: 0xbf800040. The SPI hardware that you create sets the least significant bit of this memory-mapped register when the transmission is complete. Thus, the user must check that that this memory-mapped register is 1 before writing a new data value to be transmitted (i.e., to address 0xbf80003c).

Write a new hardware module called mfp_ahb_spi to support the SPI transmission. Below is the module declaration, also located in:

`MIPSfpga_Labs\Labs\Xilinx\Part2_IO\Lab09_SPILCD\VerilogFiles\rtl_up\system\mfp_ahb_spi.v`

```
module mfp_ahb_spi(
           input        clk,
           input        resetn,
           input [7:0]  data,
           input        send,
           output       done,
           output       sdo,
           output       sck);
```

The SPI interface has the inputs and outputs described in Table 2. The clock and reset inputs (`clk` and `resetn`) are the 50 MHz system clock and the low-asserted reset. The 8-bit `data` input is the data to transmit serially. When the `send` input asserts, the mfp_ahb_spi module should deassert `done` and send the 8-bit `data` serially out through the `sdo` output, most significant bit first, as shown in Figure 1. When transmitting data, `sck` should run at about 1 kHz. The SPI interface runs relatively slowly because it will transmit over wires connected to the Nexys4 DDR board.

When the transmission is complete, the module should assert the `done` output signal.

Table 2. mfp_ahb_spi module interface signals

Signal Name	Description
clk	50 MHz MIPSfpga clock
resetn	Low-asserted reset signal
data[7:0]	8-bit data to transmit
send	1 when new data is available on data[7:0]
done	1 when done transmitting data sent, 0 when transmitting data
sdo	Serial output data
sck	Slow (~1 kHz) clock for sending serial data

Write the body of the mfp_ahb_spi module and create any submodules needed. Then test and debug the module.

3. Connecting the SPI Interface through Memory-Mapped I/O

Now memory-map the 8-bit `data` and 1-bit `done` signals to addresses 0xbf80003c and 0xbf800040, respectively. A write to 0xbf80003c should initiate the sending of the 8 least significant bits over SPI. For example, the following MIPS assembly code sends the value 0x31 over SPI.

```
lui   $8, 0xbf80
addiu $9, $0, 0x31
sw    $9, 0x3c($8)
```

A program reads the done signal and waits until it is 1 before sending another 8-bit data value over SPI. The following code reads the done signal into $11.

```
lui   $8, 0xbf80
lw    $11, 0x40($8)
```

After memory-mapping the data and done signals, test a short snippet of MIPS assembly code in simulation. Place your test code at the reset address 0xbfc00000 (i.e., in ram_reset_init.txt).

4. Liquid Crystal Displays (LCDs)

A liquid crystal display (LCD) is a small display that shows characters to a user. LCDs are commonly used in appliances, such as copy machines and microwaves, that need to show a small amount of information to the user. They come in varying sizes. The one we use, shown in Figure 2, is an 8 × 1 LCD. It displays one 8-character line of information.

Figure 2. DOGM081 8 x 1 Liquid Crystal Display (LCD) (*Courtesy DOGM081 datasheet*)

Many LCDs, including this one, have an SPI interface that can be driven by a microprocessor, such as MIPSfpga. Figure 3, taken from the DOGM081 datasheet, shows the LCD pin connections when driving it in 3.3V SPI mode. The datasheet for this LCD is available here: Lab09_SPILCD\LCD_datasheet.pdf. SI is the SPI data in (SDI) pin. CLK is the SPI clock (sck). CSB (i.e., Chip Select Bar) is the low-asserted chip-select pin. It must be 0 to enable the chip. $\overline{RESET}$ is the low-asserted reset input. **Unlike shown in Figure 3** from the datasheet, $\overline{RESET}$ (pin 40) should be tied to the system reset signal (CPU_RESETN) on the Nexys4 DDR board. Thus, when the MIPSfpga system is reset by the Nexys4 DDR red CPU RESET pushbutton, the LCD also resets. The RS input is 0 when the master sends a command to the LCD (for example, during initialization) and 1 when the master sends a character to be displayed.

 © Imagination Technologies 2017

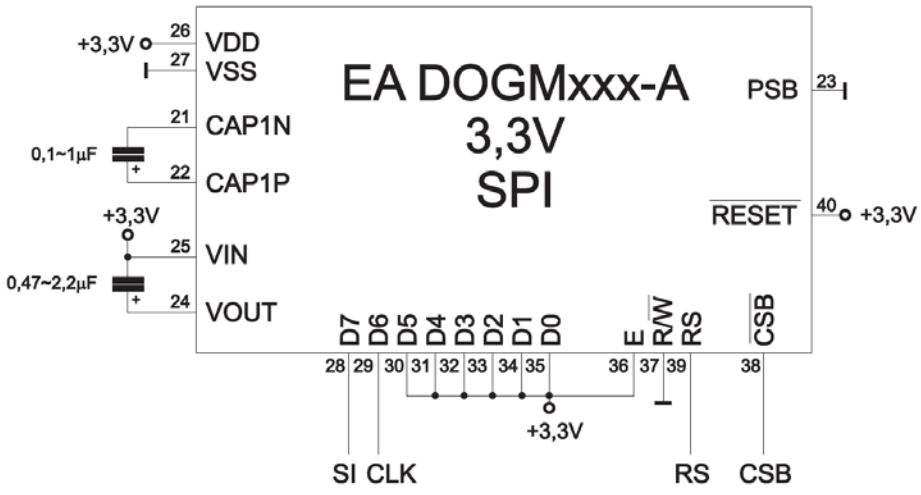

Figure 3. DOGM081, 3.3V SPI interface (courtesy DOGM081-A datasheet)

5. Driving the LCD's RS Input

To complete the connection for driving the LCD, create hardware to drive the RS pin on the LCD. RS is 1 when sending a character to display and 0 when sending a command. Instead of dedicating another memory-mapped address for RS, we can prepend it to the 8-bit data signal that has already been memory-mapped to address 0xbf80003c.

After implementing the hardware to drive RS, the following MIPS assembly code should display 'A' (ASCII 0x41) on the LCD (after the LCD has been initialized).

```
lui   $8, 0xbf80
addiu $9, $0, 0x41   # $9 = ASCII representation of 'A'
ori   $9, $9, 0x100  # prepend 1 to $9 to indicate sending data
sw    $9, 0x3c($8)   # write $9 to SPI memory-mapped register
```

6. Connecting the LCD to MIPSfpga

After completing and simulating all of the functionality described above for driving the LCD over SPI, you are ready to test your system in hardware. Wire up the DOGM081-A LCD and connect it to the Nexys4 DDR board using the connections shown in Figure 3 and Table 3 below. On the Nexys4 DDR board, use the PMOD D port to connect the MIPSfpga system to the LCD. Figure 5 shows the pin numberings of the PMOD connectors of the Nexys4 DDR board.

The SDI input of the LCD (labeled SI in Figure 3) should connect to PMOD D pin 3 (JD[3]). The LCD's SCK input (labeled CLK in Figure 3) should connect to PMOD D pin 2 (JD[2]). The LCD's RS input should connect to PMOD D pin 1 (JD[1]). **And, unlike Figure 3, the CPU RESET button should drive pin 40 through JD[4]**. Modify the top-level modules (mfp_sys.v and mfp_nexys4_ddr.v) to make these connections. Also modify the Xilinx Design Constraint file

(mfp_nexys4_ddr.xdc) to complete the connections from the MIPSfpga system to the Nexys4 DDR board.

Table 3. Table of DOGM081 LCD pin connections

DOGM081 pin	Connection	Description
21	Negative side of C1 (0.1-1 uF capacitor)	
22	Positive side of C1 (0.1-1 uF capacitor)	
23	GND	
24	Positive side of C2 (0.47-2.2 uF capacitor)	
25	3.3 V **and** negative side of C2 (0.47-2.2 uF capacitor)	
26	3.3 V	
27	GND	
28	SDI – JD[3]	
29	SCK – JD[2]	
30	3.3 V	
31	3.3 V	
32	3.3 V	
33	3.3 V	
34	3.3 V	
35	3.3 V	
36	3.3 V	
37	GND	
38	GND	Chip Select Bar (tied low to select/enable chip)
39	RS = JD[1]	0 = command, 1 = data
40	CPU_RESET_N = JD[4]	Low-asserted Reset: **Should be connected to the system reset pushbutton (CPU_RESETN)** through JD[4] in the mfp_nexys4_ddr module.

Figure 4 shows the physical locations of the LCD pins. Note that pins 1-2 and 19-20 are unconnected. Pins 3-18 do not exist.

 © Imagination Technologies 2017

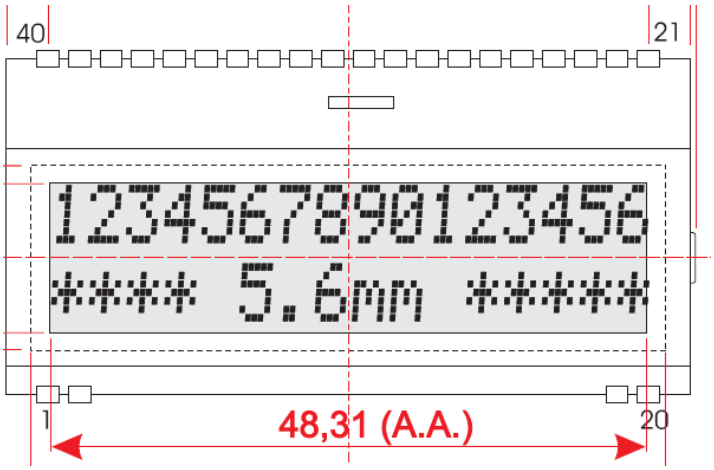

Figure 4. Physical pin locations on LCD (courtesy DOGM081 datasheet)

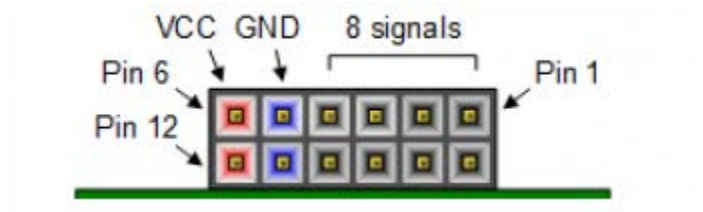

Figure 5. PMOD connector on Nexys4 DDR board (from Digilent's Nexys4 DDR User's Manual)

Before you update the hardware of the MIPSfpga system yourself, you may use the provided bitfile (mfp_nexys4_ddr.bit) in the Lab09_SPILCD directory and the example C program, described next, to test your hardware setup and LCD wiring. The provided bitfile has the hardware support for driving the SPI LCD.

7. Example C Program using SPI to Drive the LCD

Now that you've built the hardware for the SPI interface, we can write code to display text on the LCD. First, we show example code of how to initialize the LCD and then display some characters. Browse to Lab09_SPILCD\CExample. Open the main.c file. The `main` function begins by calling the `initLCD` function that initializes the LCD.

```
void initLCD() {
  unsigned int initCmds[9] = {0x31, 0x14, 0x55, 0x6d, 0x7c, 0x30, 0x0f,
                              0x06, 0x01};
  unsigned int i;
  for (i=0; i<9; i++) {
    writeValToLCD(initCmds[i]);
    delay_ms(2);
  }
}
```

 © Imagination Technologies 2017

`initLCD` sends a series of nine commands over the SPI interface to initialize the LCD. The commands are stored in the `intCmds` array. The for loop sends each command to the LCD over the SPI, using the `writeValToLCD(initCmds[i]);` statement. The commands configure the display as a 1 x 8 LCD, turn the display on, return the cursor home (to the left-most position), and set auto-incrementing, so that each character sent to the LCD will occupy the next slot. Refer to the LCD datasheet for further details about these commands.

One must wait between 26.3 µs – 1.08 ms between commands so that the command can take effect, so the for loop waits 2 milliseconds between commands using the delay_ms command you wrote in Lab 6.

The `writeValToLCD` function writes an 8-bit value to the LCD using this statement:

```
MFP_SPI_DATA = val;
```

Notice that these four lines have been added to mfp_io.h.

```
#define MFP_SPI_DATA_ADDR       0xBF80003c
#define MFP_SPI_DONE_ADDR       0xBF800040

#define MFP_SPI_DATA   (* (volatile unsigned *) MFP_SPI_DATA_ADDR      )
#define MFP_SPI_DONE   (* (volatile unsigned *) MFP_SPI_DONE_ADDR      )
```

That command writes the least significant 8 bits of `val` to the LCD over SPI. Then, the code waits until the done bit is set using this code:

```
void waitTillLCDDone() {
  unsigned int done;

  do {
    done = MFP_SPI_DONE;
  } while (!done);
}
```

After initializing the LCD, the program writes "Hello! MIPSfpga is fun!" to the LCD using the `writeStringToLCD` function.

```
writeStringToLCD("Hello!");
writeStringToLCD("MIPSfpga");
writeStringToLCD("is fun!");
```

The `writeStringToLCD` function clears the LCD, extracts each character of the string, prepends the leading 1 (to be sent to the LCD's RS pin) to indicate that it is data, and writes the value to the LCD over SPI:

```
clearLCD();

while (str[i] != '\0') {
  val = str[i];
  val = val | 0x100;  // prepend bit indicating sending data
  writeValToLCD(val);
  i++;
}
```

 © Imagination Technologies 2017

```
delay_ms(500);
```

It then delays 500 milliseconds and returns. Recall that RS is 1 when sending data (i.e., a character to be displayed) and 0 when sending a command, for example the commands used to initialize the LCD.

The LCD receives the ASCII encoding of each character and displays it. Table 4 lists the ASCII encoding of the most common characters. For example, the letter A is encoded as 0x41, B as 0x42, etc.

Table 4. ASCII encodings (Courtesy *Digital Design and Computer Architecture,* 2nd Edition, Harris & Harris, © Elsevier, 2012)

#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
20	space	30	0	40	@	50	P	60	`	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	"	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	'	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	–	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o		

Load and run the example program first onto the MIPSfpga system (provided bitfile) and then onto your enhanced MIPSfpga system (your own bitfile) using the loadMIPSfpga.bat script, as described in previous labs. After the program has loaded onto the MIPSfpga system, press the system reset button (the red CPU RESET button on the Nexys4 DDR board) to reset MIPSfpga and, importantly, the LCD and then run the program.

8. Number Guessing Game

Now write a number guessing game in C. The program should:

1. Prompt the user by writing "Guess a number (0-10)" to the LCD.
2. Wait for the user to enter a number (in binary) using the switches and press a pushbutton.
3. Display the result:
   a. If the user was correct, print "Yes!" and the guessed number to the LCD.
   b. If the user was incorrect, print "No!" and the guessed number to the LCD. Then prompt the user to guess again. Repeat until the user gets the correct answer.
4. Print "Play again!" to the LCD.

Hint about printing numbers to the LCD: Recall that the LCD displays the character corresponding to the ASCII code sent over SPI. The ASCII encoding for a number is not the number itself. For example, the number 1 is encoded as ASCII 0x31, 2 as 0x32, etc. So, in order to print a number, you must first convert it to its ASCII code. The following code writes the number 8, held in variable `val`, to the LCD. The code first prints `val` to the `tmpStr` character array using the `sprintf` (string printf) function. This converts the number to its ASCII encoding followed by the null character. The code then calls the `writeStringToLCD` with `tmpStr` as its argument to write the number to the LCD.

```
#include <stdio.h>

...
char tmpStr[5];
int val = 8;

sprintf(tmpStr, "%d", val);
writeStringToLCD(tmpStr);
```

 © Imagination Technologies 2017