Other I/O

LCD display

Flash ROM

SPI EPROM

Keyboard (PS/2)

UART connectors

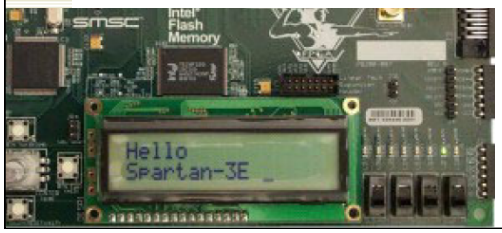
DAC

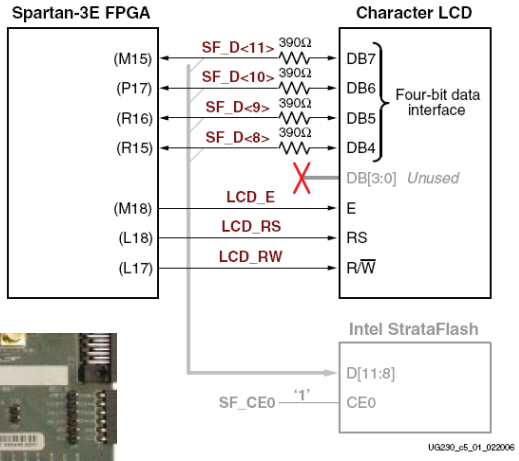
ADC

LCD Display

- 2-line, 16 character LCD display
  - 4-bit interface
  - Relatively easy to use once you have it mapped into your processor's memory-mapped I/O
  - Send characters to it, they show up on the screen
  - Not fast!
    - Scrolling at half-second intervals is about as fast as you can go and still have a clear display

LCD

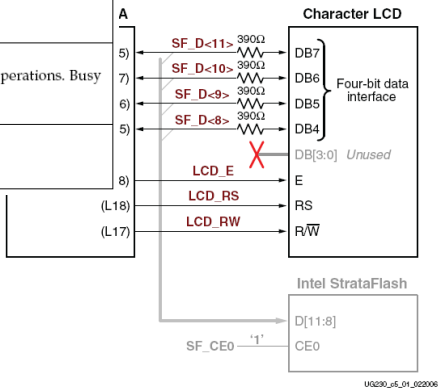




LCD Control

Table 5-1: **Character LCD Interface**

Signal Name	FPGA Pin	Function	
SF_D<11>	M15	Data bit DB7	Shared with StrataFlash pins SF_D<11:8>
SF_D<10>	P17	Data bit DB6	
SF_D<9>	R16	Data bit DB5	
SF_D<8>	R15	Data bit DB4	
LCD_E	M18	Read/Write Enable Pulse 0: Disabled 1: Read/Write operation enabled	
LCD_RS	L18	Register Select 0: Instruction register during write operations. Busy Flash during read operations 1: Data for read or write operations	
LCD_RW	L17	Read/Write Control 0: WRITE, LCD accepts data 1: READ, LCD presents data	



LCD Control

Table 5-1: **Character LCD Interface**

Signal Name	FPGA Pin	Function	
SF_D<11>	M15	Data bit DB7	Shared with StrataFlash pins SF_D<11:8>
SF_D<10>	P17	Data bit DB6	
SF_D<9>	R16	Data bit DB5	
SF_D<8>	R15	Data bit DB4	
LCD_E	M18	Read/Write Enable Pulse 0: Disabled 1: Read/Write operation enabled	
LCD_RS	L18	Register Select 0: Instruction register during write operations. Busy Flash during read operations 1: Data for read or write operations	
LCD_RW	L17	Read/Write Control 0: WRITE, LCD accepts data 1: READ, LCD presents data	

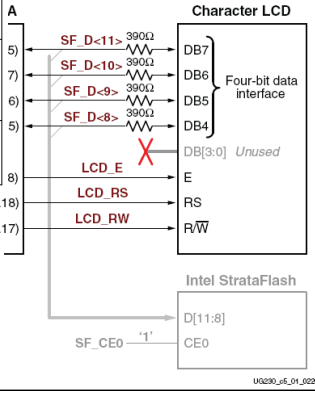


Table 5-2: **LCD/StrataFlash Control Interaction**

SF_CE0	SF_BYTE	LCD_RW	Operation
1	X	X	StrataFlash disabled. Full read/write access to LCD.
X	X	0	LCD write access only. Full access to StrataFlash.
X	0	X	StrataFlash in byte-wide (x8) mode. Upper address lines are not used. Full access to both LCD and StrataFlash.

UG230_c5_01_022006

LCD Data

- Three memory areas inside LCD
  - DD RAM – memory to hold the characters being displayed
    - Two rows of 16 characters to display
    - Also 24 extras per line that can be scrolled
  - CG ROM – Pre-defined character map
    - 192 pre-defined characters
  - CG RAM – RAM to hold 8 custom characters
    - 5x8 bit character/glyphs

DD RAM

- DD RAM – memory to hold the characters being displayed

Character Display Addresses

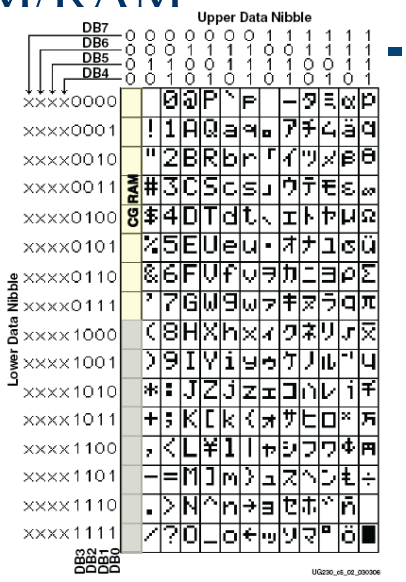
																Undisplayed Addresses		
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	…	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	…	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	…	40

Figure 5-3: **DD RAM Hexadecimal Addresses (No Display Shifting)**

  - Data written to each of these locations is the 8-bit address of a character in the CG ROM/ RAM

CG ROM/RAM

- For example, 8'h53 = S
  - Note the Japanese kana characters…
- Also, notice the 8 CG RAM locations
  - Addresses 8'h00 to 8'h07



CG RAM

- This example is custom character 0'h03
  - Note that there are 8 rows in custom character 3
  - So, it takes 8 writes to make a custom character
  - Row address is incremented automatically…

						Upper Nibble				Lower Nibble			
						Write Data to CG RAM or DD RAM							
A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Character Address			Row Address			Don't Care			Character Bitmap				
0	1	1	0	0	0	-	-	-	0	■	0	■	0
0	1	1	0	0	1	-	-	-	■	0	■	0	■
0	1	1	0	1	0	-	-	-	0	■	0	■	0
0	1	1	0	1	1	-	-	-	■	0	■	0	■
0	1	1	1	0	0	-	-	-	0	■	0	■	0
0	1	1	1	0	1	-	-	-	■	0	■	0	■
0	1	1	1	1	0	-	-	-	0	■	0	■	0
0	1	1	1	1	1	-	-	-	0	0	0	0	0

Figure 5-5: **Example Custom Checkerboard Character with Character Code 0x03**

Operation Overview

- Pick an LCD screen location
  - Write an 8-bit character address to that location
  - Then it shows up on the screen
- Pick a CG RAM location
  - Write 8 bytes starting at that location
  - Now you can use that new custom character
- Do it all with just a four-bit interface…
  - Lots of little nibble writes….

Command Set

- Commands are sent upper-nibble first

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Cursor Home	0	0	0	0	0	0	0	0	1	-
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display On/Off	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	-	-
Function Set	0	0	0	0	1	0	1	0	-	-
Set CG RAM Address	0	0	0	1	A5	A4	A3	A2	A1	A0
Set DD RAM Address	0	0	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Address	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG RAM or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Read Data from CG RAM or DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0

Command Set

Clear Display

Clear the display and return the cursor to the home position, the top-left corner.

This command writes a blank space (ASCII/ANSI character code 0x20) into all DD RAM addresses. The address counter is reset to 0, location 0x00 in DD RAM. Clears all option settings. The I/D control bit is set to 1 (increment address counter mode) in the Entry Mode Set command.

Execution Time: 82 µs – 1.64 ms

Return Cursor Home

Return the cursor to the home position, the top-left corner. DD RAM contents are unaffected. Also returns the display being shifted to the original position, shown in Figure 5-3.

The address counter is reset to 0, location 0x00 in DD RAM. The display is returned to its original status if it was shifted. The cursor or blink move to the top-left character location.

Execution Time: 40 µs – 1.6 ms

Command Set

Entry Mode Set

Sets the cursor move direction and specifies whether or not to shift the display.

These operations are performed during data reads and writes.

Execution Time: 40 µs

Bit DB1: (I/D) Increment/Decrement

0	Auto-decrement address counter. Cursor/blink moves to left.
1	Auto-increment address counter. Cursor/blink moves to right.

This bit either auto-increments or auto-decrements the DD RAM and CG RAM address counter by one location after each Write Data to CG RAM or DD RAM or Read Data from CG RAM or DD RAM command. The cursor or blink position moves accordingly.

Bit DB0: (S) Shift

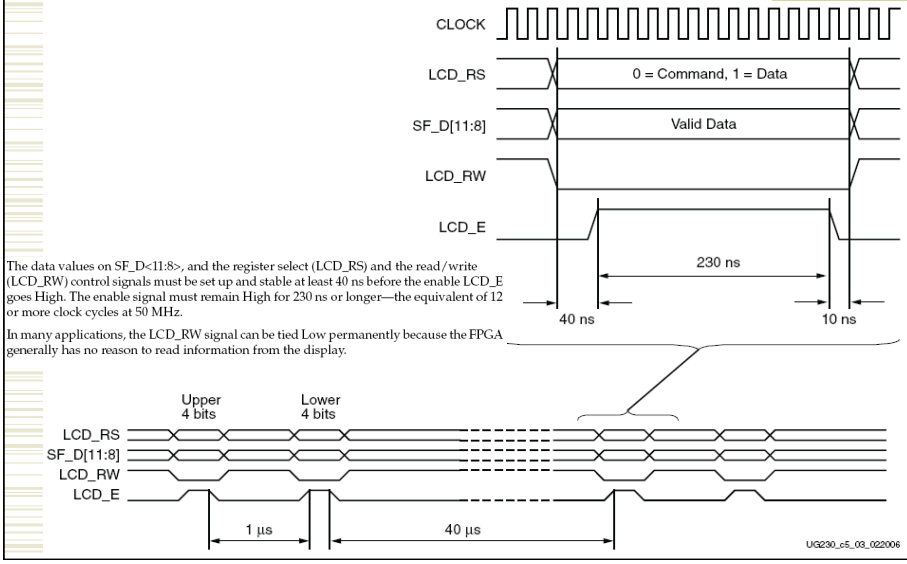
0	Shifting disabled
1	During a DD RAM write operation, shift the entire display value in the direction controlled by Bit DB1 (I/D). Appears as though the cursor position remains constant and the display moves.

Command Set

- Commands are sent upper-nibble first

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Cursor Home	0	0	0	0	0	0	0	0	1	-
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display On/Off	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	-	-
Function Set	0	0	0	0	1	0	1	0	-	-
Set CG RAM Address	0	0	0	1	A5	A4	A3	A2	A1	A0
Set DD RAM Address	0	0	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Address	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG RAM or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Read Data from CG RAM or DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0

Write Timing

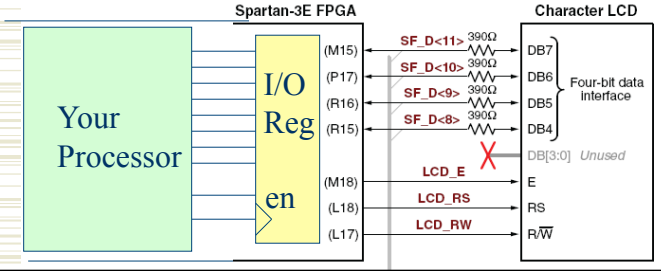


The data values on SF_D<11:8>, and the register select (LCD_RS) and the read/write (LCD_RW) control signals must be set up and stable at least 40 ns before the enable LCD_E goes High. The enable signal must remain High for 230 ns or longer—the equivalent of 12 or more clock cycles at 50 MHz.

In many applications, the LCD_RW signal can be tied Low permanently because the FPGA generally has no reason to read information from the display.

Memory Mapped I/O

- So, as a practical matter, the easiest way to deal with the LCD is to map the interface to a memory-mapped location
  - Now you can, under program control, change the values on the data and control wires



Writing to the address of the LCD Reg will update its value

Initialization

Power-On Initialization

The initialization sequence first establishes that the FPGA application wishes to use the four-bit data interface to the LCD as follows:

- Wait 15 ms or longer, although the display is generally ready when the FPGA finishes configuration. The 15 ms interval is 750,000 clock cycles at 50 MHz.
- Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
- Wait 4.1 ms or longer, which is 205,000 clock cycles at 50 MHz.
- Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
- Wait 100 µs or longer, which is 5,000 clock cycles at 50 MHz.
- Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
- Wait 40 µs or longer, which is 2,000 clock cycles at 50 MHz.
- Write SF_D<11:8> = 0x2, pulse LCD_E High for 12 clock cycles.
- Wait 40 µs or longer, which is 2,000 clock cycles at 50 MHz.

Remember, this display is SLOW compared to 50MHz!!!

Configuration

Display Configuration

After the power-on initialization is completed, the four-bit interface is now established. The next part of the sequence configures the display:

- Issue a Function Set command, 0x28, to configure the display for operation on the Spartan-3E Starter Kit board.
- Issue an Entry Mode Set command, 0x06, to set the display to automatically increment the address pointer.
- Issue a Display On/Off command, 0x0C, to turn the display on and disables the cursor and blinking.
- Finally, issue a Clear Display command. Allow at least 1.64 ms (82,000 clock cycles) after issuing this command.

Using the Display

To write data to the display, specify the start address, followed by one or more data values.

Before writing any data, issue a Set DD RAM Address command to specify the initial 7-bit address in the DD RAM. See Figure 5-3 for DD RAM locations.

Write data to the display using a Write Data to CG RAM or DD RAM command. The 8-bit data value represents the look-up address into the CG ROM or CG RAM, shown in Figure 5-4. The stored bitmap in the CG ROM or CG RAM drives the 5 x 8 dot matrix to represent the associated character.

If the address counter is configured to auto-increment, as described earlier, the application can sequentially write multiple character codes and each character is automatically stored and displayed in the next available location.

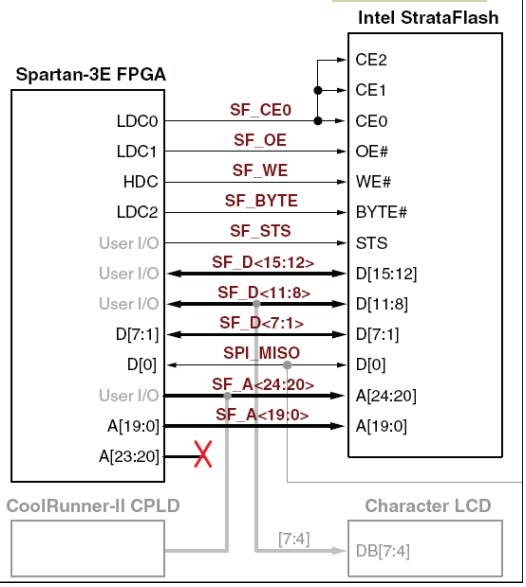
Continuing to write characters, however, eventually falls off the end of the first display line. The additional characters do not automatically appear on the second line because the DD RAM map is not consecutive from the first line to the second.

Remember timing!

- The LCD_E enable pulse must be high for at least 230ns (12 clock cycles at 50MHz)
- The two nibbles must be separated by 1µs (50 cycles)
- Two different commands must be separated by 40µs (2000 cycles)
  - But, these are easily done in an assembly language program… (as are the even longer configuration delays)

Strata Flash

- 16 MByte (8 Mword) flash ROM
  - Designed to hold configuration data for the Spartan part
  - But, can be used for general non-volatile data



Strata Flash

- Some data lines are shared with the LCD
  - But, if you don't read back from the LCD they can both work together

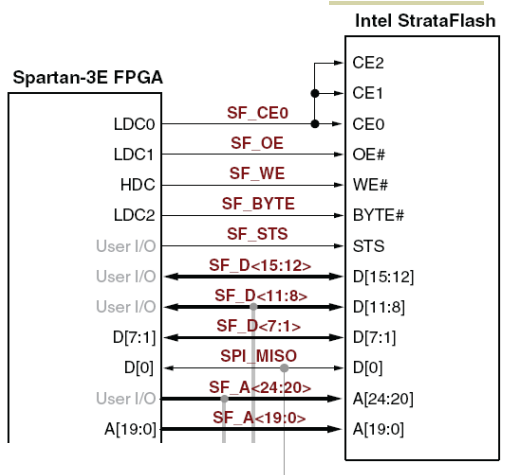
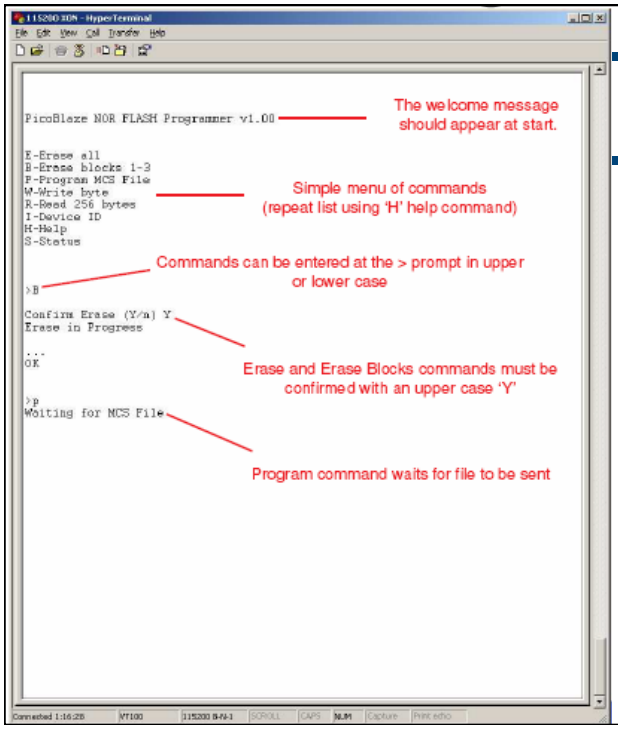


Table 11-2: **FPGA Control for StrataFlash and LCD**

SF_CE0	LCD_RW	Function
1	1	The FPGA reads from the character LCD.
0	0	The FPGA accesses the StrataFlash PROM.

Writing to the Strata Flash

- Tricky!
- Luckily, there is reference design on the Xilinx web site that implements a Flash programmer
  - You can use this to load data to your board
  - See class web site in the xilinx examples directory
  - www.eng.utah.edu/~3710/xilinx-docs/examples
  - s3esk_picoblaze_nor_flash_programmer



Xilinx Flash Project

Reading from the Flash

- Not as tricky
  - But, the flash has a 75ns access time
  - So, it will take four 50MHz cycles to read data
    - Each cycle is 20ns
  - Set SF_oe and SF_ce active (low) and wait for four cycles (80ns) before grabbing return data…
  - As usual map the flash into your processor's memory-mapped address space

Xilinx Example

Reading the StrataFLASH NOR memory is relatively straightforward. The only issue for PicoBlaze is that it does not have a 24-bit address range and therefore multiple ports are used to achieve the operation.

```
SF_byte_read: OUTPUT s9, SF_addr_hi_port
              OUTPUT s8, SF_addr_mi_port
              OUTPUT s7, SF_addr_lo_port
              LOAD s1, 05
              OUTPUT s1, SF_control_port
              LOAD s1, 06
              LOAD s1, 06
              INPUT s0, SF_data_in_port
              OUTPUT s1, SF_control_port
              RETURN
```

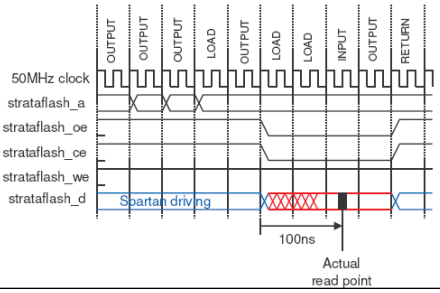
Set 24-bit address form which to read

Set controls for read
- Bit 0 - strataflash_read='1'
  Enables memory outputs (strataflash_oe='0')
  Tri-states the Spartan outputs (strataflash_d='Z')
- Bit 1 - strataflash_ce='0'
  Enables memory
- Bit 2 - strataflash_we='1'
  Write enable is off (read operation)

Read data

Deselect StrataFLASH memory

All PicoBlaze instructions execute in 2 clock cycles and the design uses the 50MHz clock source on the board. This makes all timing of the design easy to predict and to ensure that the specifications for the StrataFLASH memory are met.



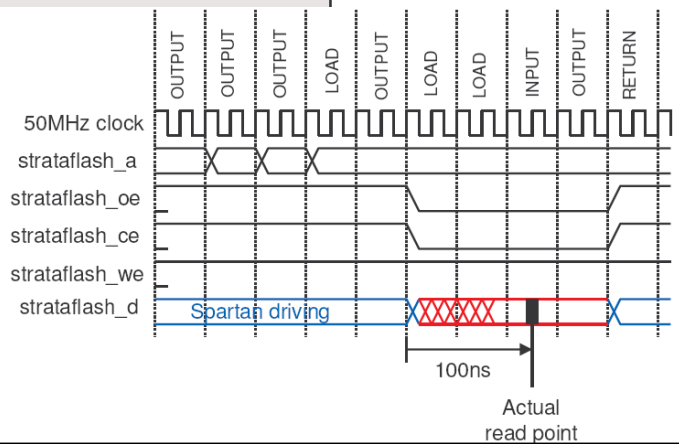
The access time of the memory is 75ns (see Intel data sheet for details). By including an additional LOAD instruction, the time between setting the controls to read the memory and the actual point of reading is increased by 40ns and the access time in adequate.

Note that the input port multiplexer is pipelined which means that the data from the memory is captured on the first clock edge of the INPUT instruction (as indicated) and then passed into the 's0' register on the second clock edge.

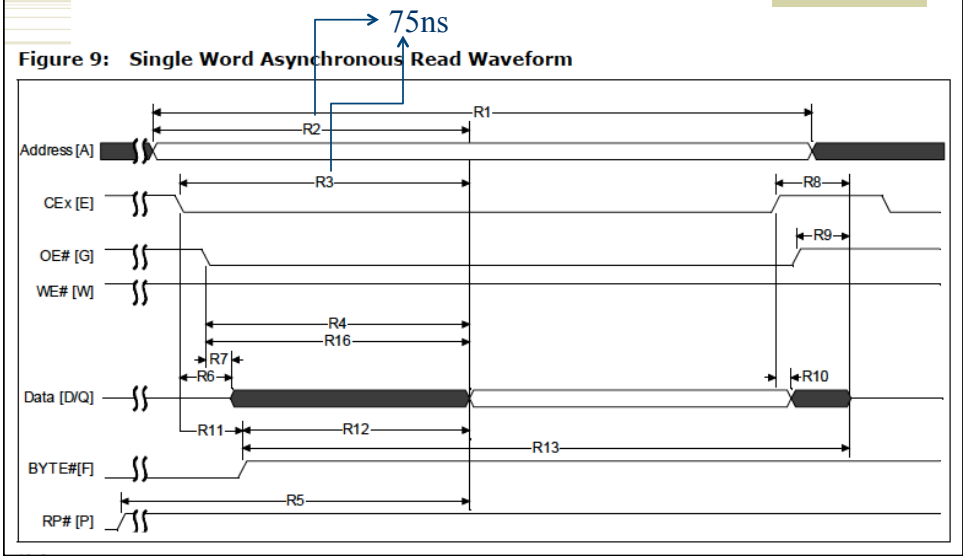
Hint – Data is read from the memory when it is in 'read array' mode (which is the default mode after power up). However, the same read operation is used to access memory status and device information when in other modes.

Xilinx Example

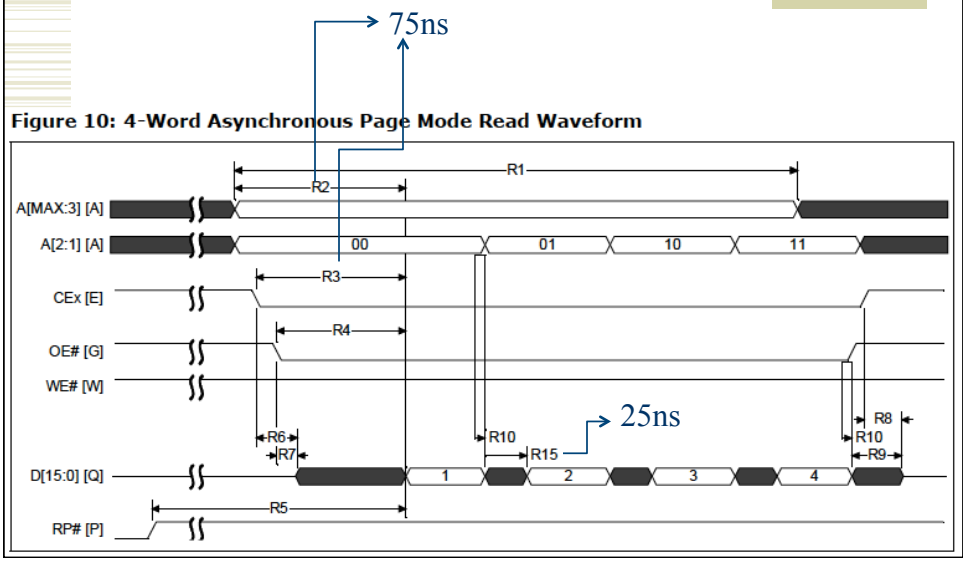
```
SF_byte_read: OUTPUT s9, SF_addr_hi_port
              OUTPUT s8, SF_addr_mi_port
              OUTPUT s7, SF_addr_lo_port
              LOAD s1, 05
              OUTPUT s1, SF_control_port
              LOAD s1, 06
              LOAD s1, 06
              INPUT s0, SF_data_in_port
              OUTPUT s1, SF_control_port
              RETURN
```



Read Waveforms



Page Mode Read



SPI Serial Flash

- 16Mbit – SPI serial protocol
- Mostly used for Xilinx configuration
  - But, you can use it for data if you want to
- You can program it using the Impact tool
- As with all Flash – reading is (relatively) easy, writing is more complex
  - In this case, reading one byte takes 40 clock ticks…

SPI Serial Flash

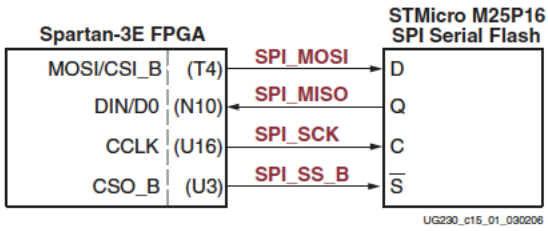


Figure 12-1: **Spartan-3E FPGAs Have an Optional SPI Flash Configuration Interface**

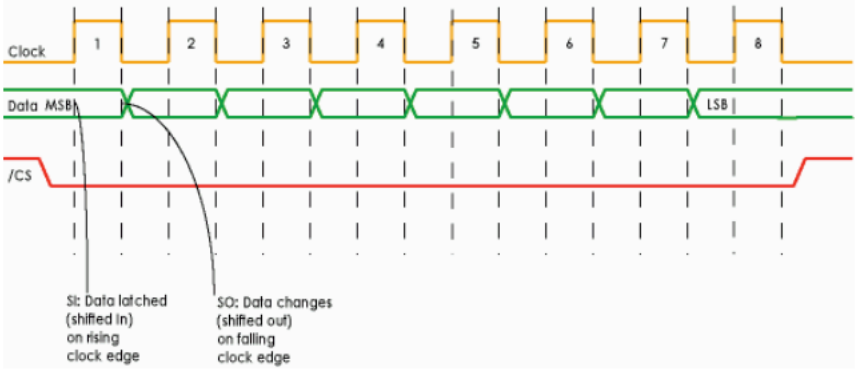
Table 12-1: **SPI Flash Interface Signals**

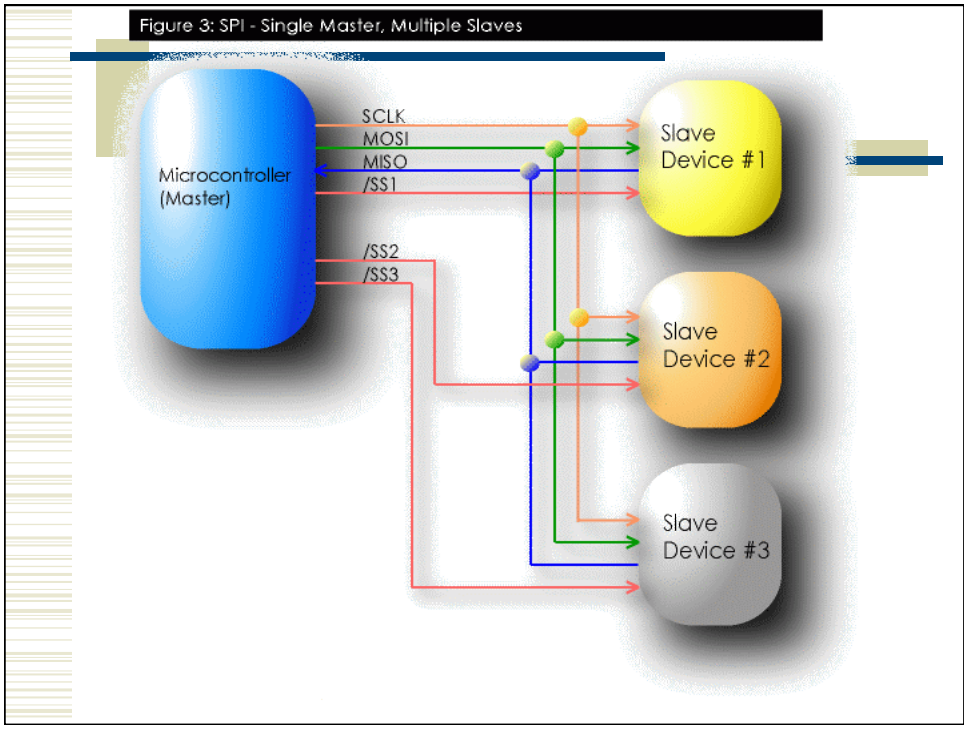
Signal	FPGA Pin	Direction	Description
SPI_MOSI	T4	FPGA→SPI	Serial data: Master Output, Slave Input
SPI_MISO	N10	FPGA←SPI	Serial data: Master Input, Slave Output
SPI_SCK	U16	FPGA→SPI	Clock
SPI_SS_B	U3	FPGA→SPI	Asynchronous, active-Low slave select input

Serial Output

- Two pins: Clk and Data
  - New data presented at Data pin on every clock
  - Looks like a shift register

Figure 4: Microwire Protocol





SPI Serial Flash

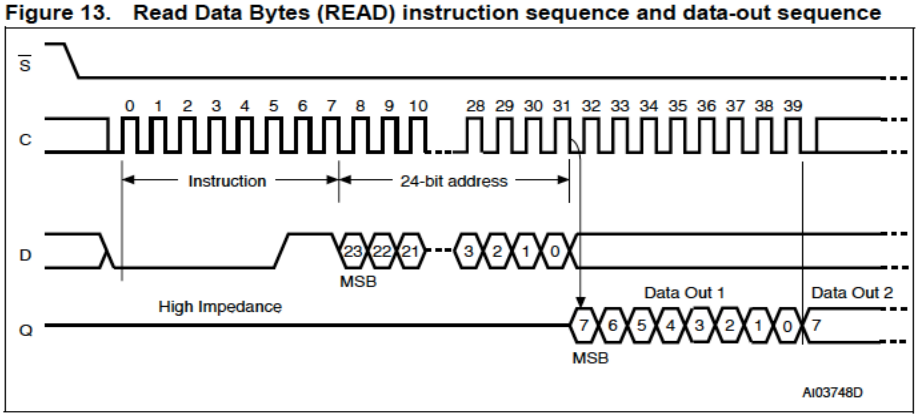
Table 4. Instruction set

Instruction	Description	One-byte instruction code		Address bytes	Dummy bytes	Data bytes
WREN	Write Enable	0000 0110	06h	0	0	0
WRDI	Write Disable	0000 0100	04h	0	0	0
RDID	Read Identification	1001 1111	9Fh	0	0	1 to 20
RDSR	Read Status Register	0000 0101	05h	0	0	1 to ∞
WRSR	Write Status Register	0000 0001	01h	0	0	1
READ	Read Data Bytes	0000 0011	03h	3	0	1 to ∞
FAST_READ	Read Data Bytes at Higher Speed	0000 1011	0Bh	3	1	1 to ∞
PP	Page Program	0000 0010	02h	3	0	1 to 256
SE	Sector Erase	1101 1000	D8h	3	0	0
BE	Bulk Erase	1100 0111	C7h	0	0	0
DP	Deep Power-down	1011 1001	B9h	0	0	0
RES	Release from Deep Power-down, and Read Electronic Signature	1010 1011	ABh	0	3	1 to ∞
	Release from Deep Power-down			0	0	0

SPI Serial Flash

32 clocks before data starts coming back (runs up to 75MHz)
Then 8 more ticks to get the data (MSB first)

Figure 13. Read Data Bytes (READ) instruction sequence and data-out sequence

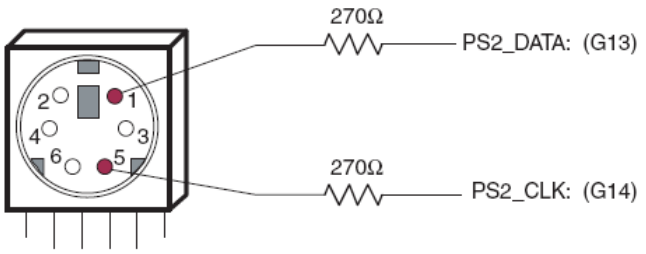


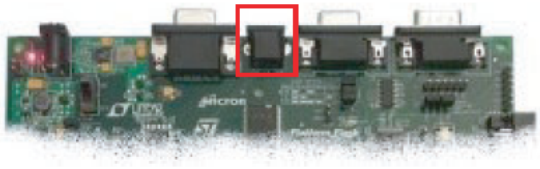
1. Address bits A23 to A21 are Don't care.

PS/2 Keyboard Interface

- Standard keyboard interface
  - Serial protocol similar to UART, but with its own clock
  - When you press a key, the keyboard sends a "make code" (one, sometimes two, bytes)
  - When you release the key, the keyboard sends a "break code" (two, sometimes three, bytes)
  - Collectively, these are known as "scan codes"

PS/2 Keyboard Interface





UG230_c8_01_021806

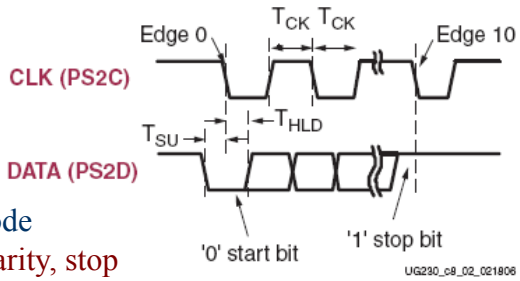
PS/2 Keyboard Interface

Table 8-2: **PS/2 Bus Timing** 20-30 kHz

Symbol	Parameter	Min	Max
T_{CK}	Clock High or Low Time	30 µs	50 µs
T_{SU}	Data-to-clock Setup Time	5 µs	25 µs
T_{HLD}	Clock-to-data Hold Time	5 µs	25 µs

Codes are sent LSB first with Odd parity

Note that 11 bits are sent for each code start, 8-data, odd parity, stop



UG230_c8_02_021806

Figure 8-2: **PS/2 Bus Timing Waveforms**

Scan Codes (Make Codes)

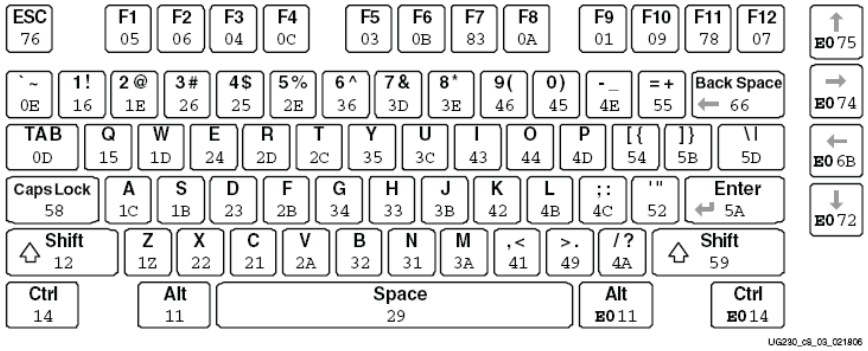


Figure 8-3: **PS/2 Keyboard Scan Codes**

Break codes are the same, but prefixed with 0xF0
for example – Q break code is 0xF0 0x15, ➔ is E0 F0 74

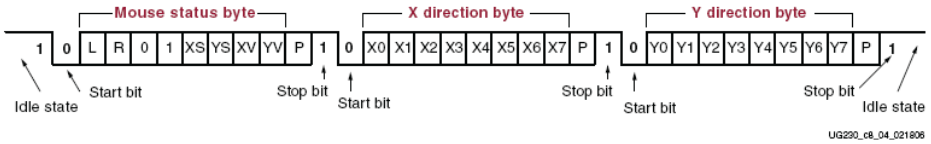
ASCII codes

$B_3B_2B_1B_0$	$B_6B_5B_4$ 000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	—
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

PS/2 Things to Keep in Mind

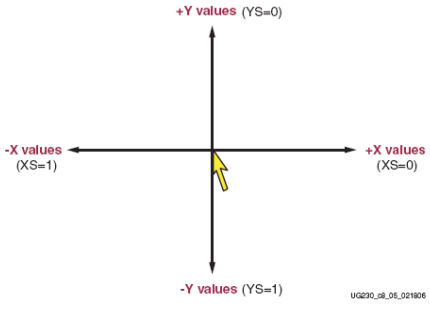
- When you press and hold a key, the make code is sent every 100ms or so
- If no key is pressed, both clk and data are in their idle state
- Probably want a PS/2 controller that grabs codes and puts them in a register that can be read by your program (memory mapped I/O)
- Probably want to set a bit that says "new code" that gets cleared when the code is read

PS/2 Mouse



Whenever the mouse moves it Sends three bytes.

Status tells you state of buttons sign of X and Y, and overflow for X and Y



UART

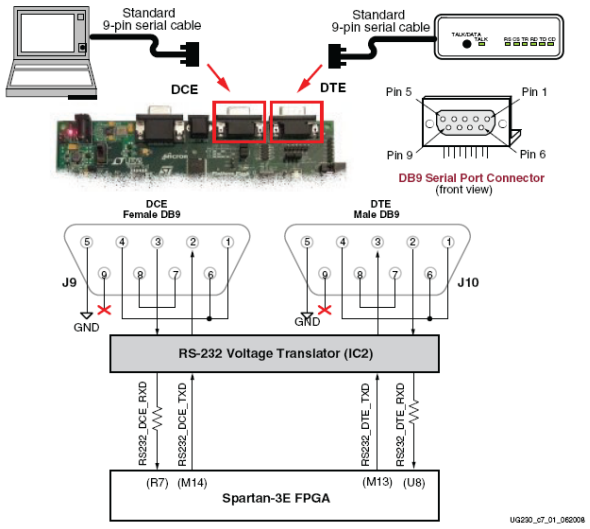


Figure 7-1: **RS-232 Serial Ports**

Two main parts:
Connectors
Voltage translator

You provide the UART circuit!

(See 3700 UART for details)

UART Basics

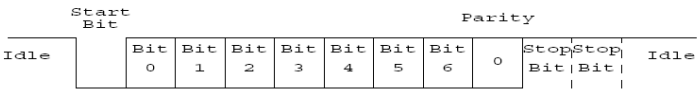


Figure 2: Data Byte Transmission Format

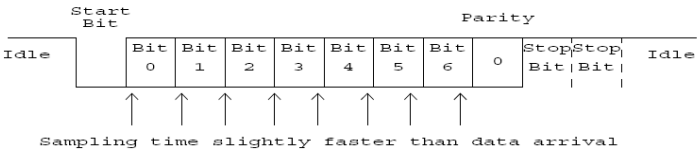


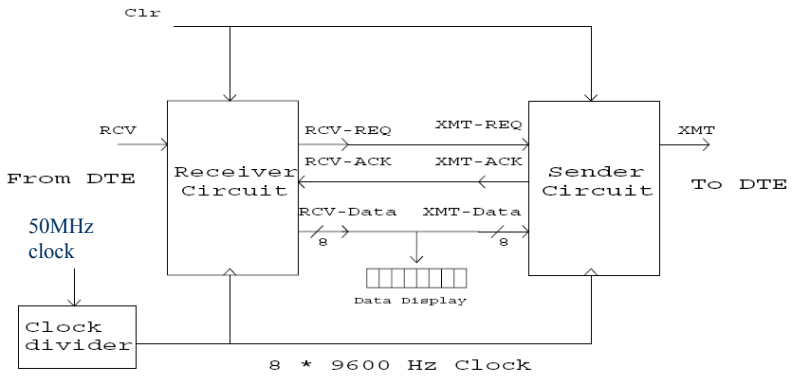
Figure 3: Framing Error

9600 * 8 = 76.8kHz
50MHz/651 = 76.805kHz

UART Basics

$B_3B_2B_1B_0$	$B_6B_5B_4$ 000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	—
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

UART Basics



Use rcv-req as a flag to be read by your program?
Assert xmt-req by your program to initiate send?

Digital to Analog Converter

- Four-channel 12-bit DAC
- Serial SPI protocol - up to 50MHz
- 32-bit data format

SPI ADC

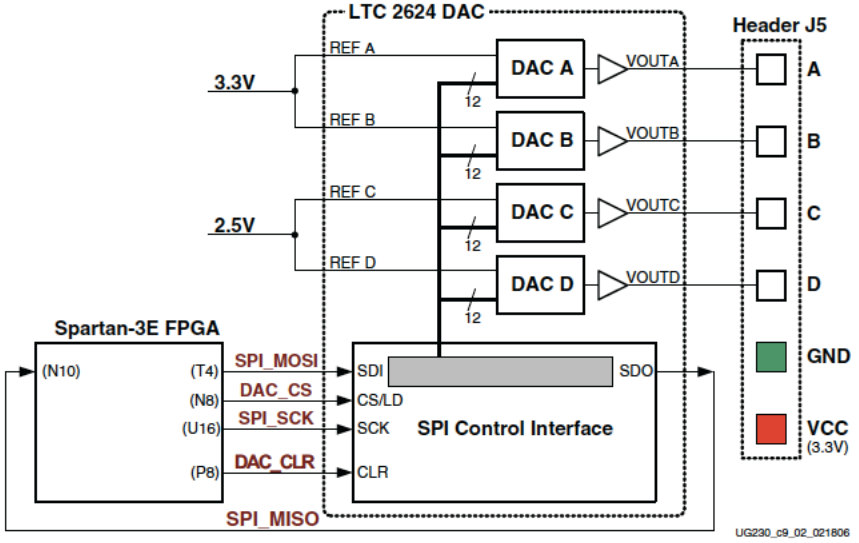


Figure 9-2: **Digital-to-Analog Connection Schematics**

SPI ADC

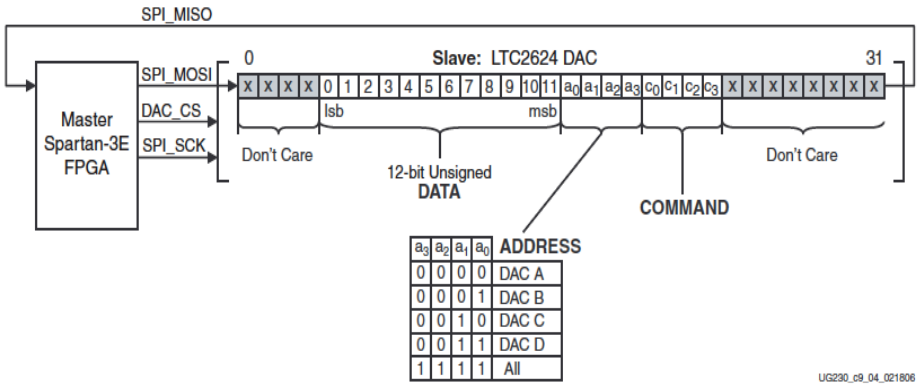


Figure 9-4: **SPI Communications Protocol to LTC2624 DAC**

Other SPI Parts

- Remember to disable the other SPI devices…

Table 9-2: **Disabled Devices on the SPI Bus**

Signal	Disabled Device	Disable Value
SPI_SS_B	SPI serial Flash	1
AMP_CS	Programmable pre-amplifier	1
AD_CONV	Analog-to-Digital Converter (ADC)	0
SF_CE0	StrataFlash Parallel Flash PROM	1
FPGA_INIT_B	Platform Flash PROM	1

Analog Capture

- Programmable scaling pre-amplifier
- 14-bit ADC
- SPI interface to both of them

Analog to Digital Converter

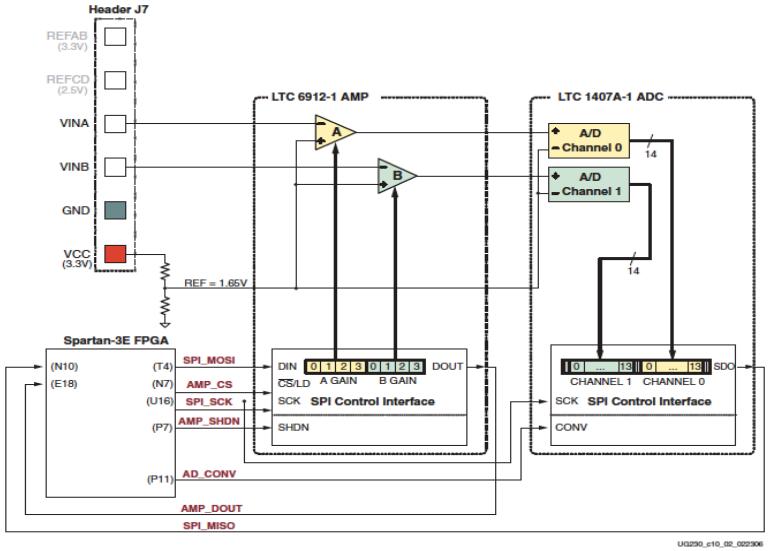


Figure 10-2: **Detailed View of Analog Capture Circuit**

SPI to Pre-amp

Table 10-2: **Programmable Gain Settings for Pre-Amplifier**

Gain	A3	A2	A1	A0	Input Voltage Range	
	B3	B2	B1	B0	Minimum	Maximum
0	0	0	0	0		
-1	0	0	0	1	0.4	2.9
-2	0	0	1	0	1.025	2.275
-5	0	0	1	1	1.4	1.9
-10	0	1	0	0	1.525	1.775
-20	0	1	0	1	1.5875	1.7125
-50	0	1	1	0	1.625	1.675
-100	0	1	1	1	1.6375	1.6625

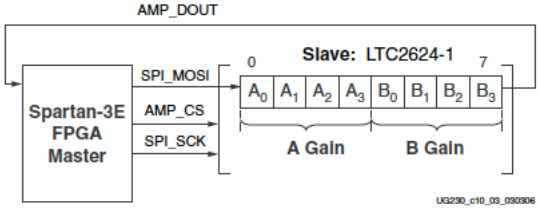


Figure 10-3: **SPI Serial Interface to Amplifier**

SPI to ADC

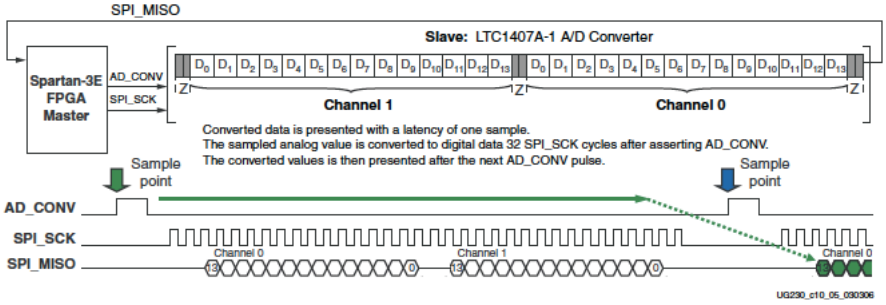


Figure 10-6: **Analog-to-Digital Conversion Interface**

Figure 10-7 shows detailed transaction timing. The AD_CONV signal is not a traditional SPI slave select enable. Be sure to provide enough SPI_SCK clock cycles so that the ADC leaves the SPI_MISO signal in the high-impedance state. Otherwise, the ADC blocks communication to the other SPI peripherals. As shown in Figure 10-6, use a 34-cycle communications sequence. The ADC 3-states its data output for two clock cycles before and after each 14-bit data transfer.

Summary

- All I/O can be mapped into your memory space
  - You have lots of room left over in the addressable space if you use block RAMs only
  - Might need custom FSMs to actually talk to the I/O
- Control the devices under program control
  - Some memory locations will be data, some will be control
  - Writing or reading these locations will have I/O side effects
  - Remember to consider timing!
- Think about how your program will interact with I/O

Memory Map