Embedded Systems Lab 2: DE2 Graphics and Hardware

Part 1: LCD Display

This section is designed to get the student used to reading and interpreting pre-written code. It also introduces the student to extracting information from data sheets and information tables and a first peek at a commonly missed difference between simulation, desktop implementation and deployment – start up timing.

Set up

Download the Lab 2 Part 1 package from the course website and extract to your working directory.

Understanding the code

Driving the LCD is split in to two parts. `LCD.v` contains the instructions and data to move to the LCD while `LCD_Controller.v` is responsible for actually driving the LCD pins.

An extract of the pin functions and instruction set for the LCD are shown below. The data to be transferred to the LCD is encoded in the look-up table in LCD.v

Table 1 LCD Pin Assignments

Pin Functions

Signal	No. of Lines	I/O	Device Interfaced with	Function
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/$\overline{W}$	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation.

Table 2 LCD Instructions

Table 6 Instructions

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{op} or f_{osc} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Table 3 The ASCII character set

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Activities

The data and instructions to the LCD are 8-bits wide. Why are the entries in LCD.v's `LUT_DATA` 9 bits wide? What role does the most significant bit play (hint: the highest bit of `LUT_DATA` is only read once, where and why)?

The instructions to initialise the LCD are given at the beginning of the LUT. Convert each hexadecimal instruction to its binary equivalent and look them up in the table of LCD instructions (Table 2 above). Comment each of the first 5 LUT entries describing what each entry does. Where full descriptions of the bit purposes aren't given above (e.g. the Entry Mode Set I/D bit), just write the bit name and value in the comment.

Display characters are given by their ASCII representation. Using the table of ASCII codes (Table 3 above), change the message displayed on the LCD to whatever you wish.

The `Reset_Delay` module makes sure the LCD doesn't start receiving commands for some time after the FPGA starts up. Noting that the clock is 50MHz, how long does that module wait before it sets the reset line high?

Try commenting out the `Reset_Delay` module instantiation and replacing it with

```
assign DLY_RST = 1;
```

The program should still work. Given that, what's the point of having this delayed reset? Hint: Refer to the Reset Function extract of the LCD data sheet shown below and think: What's different

between how you're using the FPGA in labs compared to how it might be used in an Embedded System?

Reset Function

Initializing by Internal Reset Circuit

An internal reset circuit automatically initializes the HD44780U when the power is turned on. The following instructions are executed during the initialization. The busy flag (BF) is kept in the busy state until the initialization ends (BF = 1). The busy state lasts for 10 ms after V_{CC} rises to 4.5 V.

1. Display clear
2. Function set:
   DL = 1; 8-bit interface data
   N = 0; 1-line display
   F = 0; 5 × 8 dot character font
3. Display on/off control:
   D = 0; Display off
   C = 0; Cursor off
   B = 0; Blinking off
4. Entry mode set:
   I/D = 1; Increment by 1
   S = 0; No shift

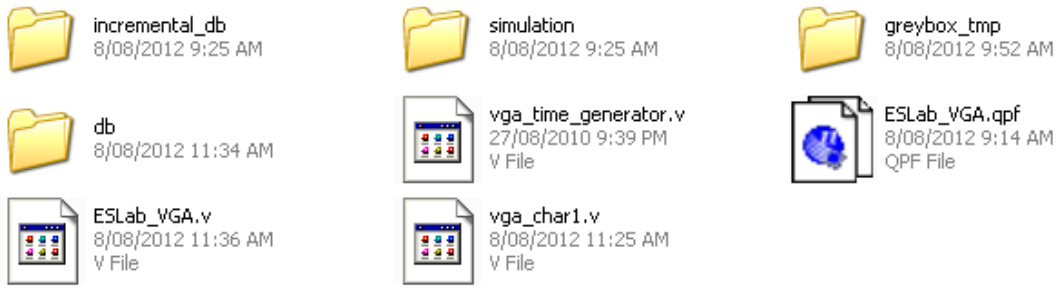
Note: If the electrical characteristics conditions listed under the table Power Supply Conditions Using Internal Reset Circuit are not met, the internal reset circuit will not operate normally and will fail to initialize the HD44780U. For such a case, initial-ization must be performed by the MPU as explained in the section, Initializing by Instruction.

Part 2: Fun with Clocks

In this section, we will learn how to drive the VGA display, use the Quartus MegaFunction Wizard to generate pre-written parameterised modules and learn some of the subtler aspects of clock generation.

Set up

Download the Lab 2 Part 1 package from the course website and extract to your working directory. You should have the following files:



Connect a VGA cable from the output of the DE2 board to the input on the back of your main monitor. When testing the VGA function, you will use the control keys on your monitor to switch backwards and forwards between the PC output and the board output.

Understanding the Code

The code is split in to two parts; `ESLab_VGA.v` provides all the top level connections, while `vga_time_generator.v` takes care of generating synchronization and timing signals to the display.

The time generator produces two three signals of interest, `CounterX` and `CounterY` contain the current row and column being drawn to the screen and `VGA_BLANK_N` is high when these two are valid. In order to draw to the screen at a particular location, all one has to do is wait for the counters to be valid and at the correct location, then set the RGB signals to the desired colour. The example code draws a vertical red line and a horizontal green line in exactly this way.

Activities

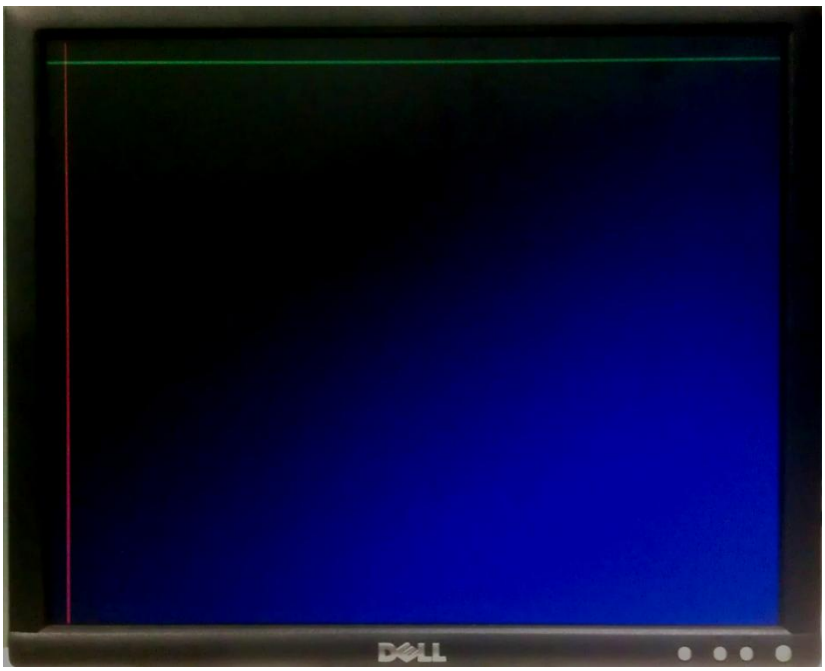
Part 1: Basic VGA

How many bits are used for each colour channel and therefore how many different colours can be displayed on the screen with this code?

At what resolution is the screen being driven?

The VGA timing generator has 'Porch' timings. These are extra pixels added off the screen at the top, bottom, left and right to provide padding around the main image. When the VGA code is writing to this porch area, the `VGA_BLANK_N` is set to 0 but does `CounterX` and `CounterY` start inside the porch area or inside the main screen? That is, is the first visible pixel in the X direction `CounterX == 0` or `CounterX == hfporch` (=16)? Answer this question by looking at the code in `vga_timing_generator.v` then confirm experimentally.

The blue channel currently isn't utilised for anything. Write code to generate a blue diagonal gradient as shown below. It might be helpful to start with a vertical gradient first.

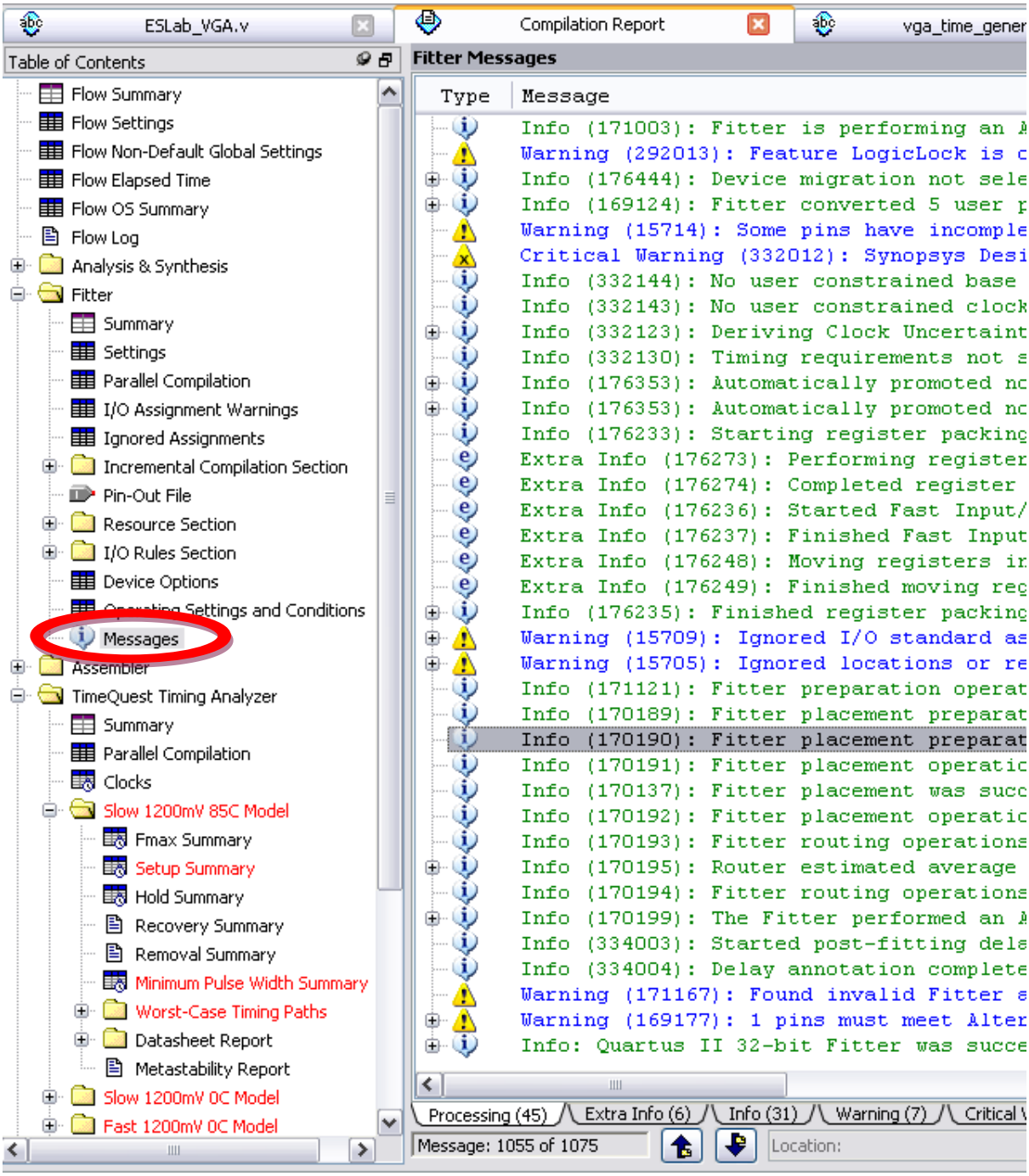


Part 2: Clock Generation and the Megafunction Wizard

The VGA Timing Generator requires a 25MHz clock input. This is generated from the 50MHz signal with the following code:

```
always @( posedge CLOCK_50 )
            VGA_CLK <= VGA_CLK + 1;
```

In lectures, it was explained that generating your own clocks by dividing external ones in logic is generally a bad idea. Logic propagation times through the chip are much greater than clock propagation times, so clock skew issues can arise. In this particular case though, it's quite safe. To see why, open the "Compilation Report" tab, expand the "Fitter" item and click "Messages". A screen shot of this is shown below.



The Quartus fitter is smart enough to recognize some logic nets as actually being clocks and "Promotes" them to clock nets, for which clock skew doesn't matter as much. Find the message in the window that confirms that the `VGA_CLK` net has been recognized as a clock signal and is therefore perfectly safe to use in this fashion. Which other net has been promoted to a clock? Look at the code for that second net, does its behaviour look approximately "clock-like"?

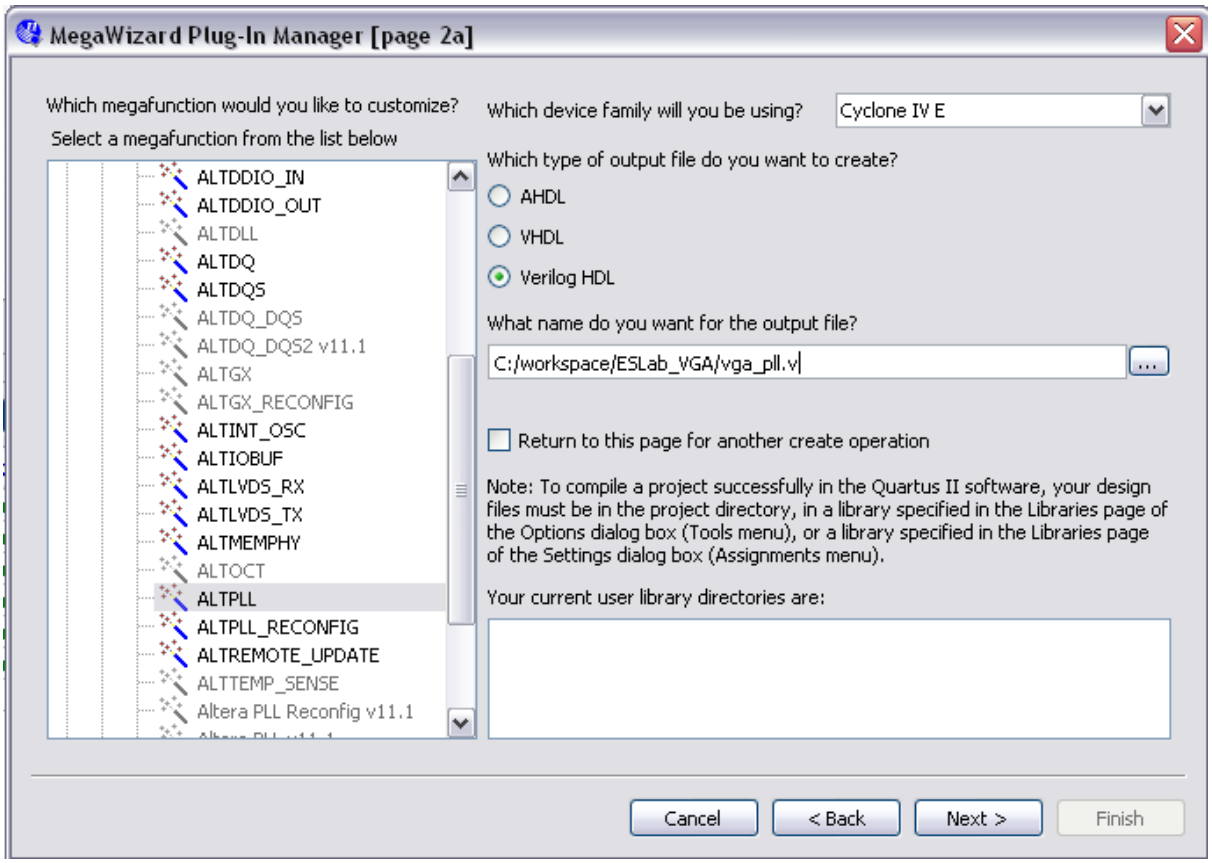
Expand the "TimeQuest Timing Analyzer" section of the Compilation Messages tab, then the "Slow 1200mV 85C Model" section and click "Fmax Summary". The `VGA_CLK` must run at 25MHz, has the promotion to clock net meant that that frequency can be achieved?

There are a limited number of clock nets for the Fitter to work with and it might not always get this promotion correct. To make the clock generation explicit, one might use a PLL.

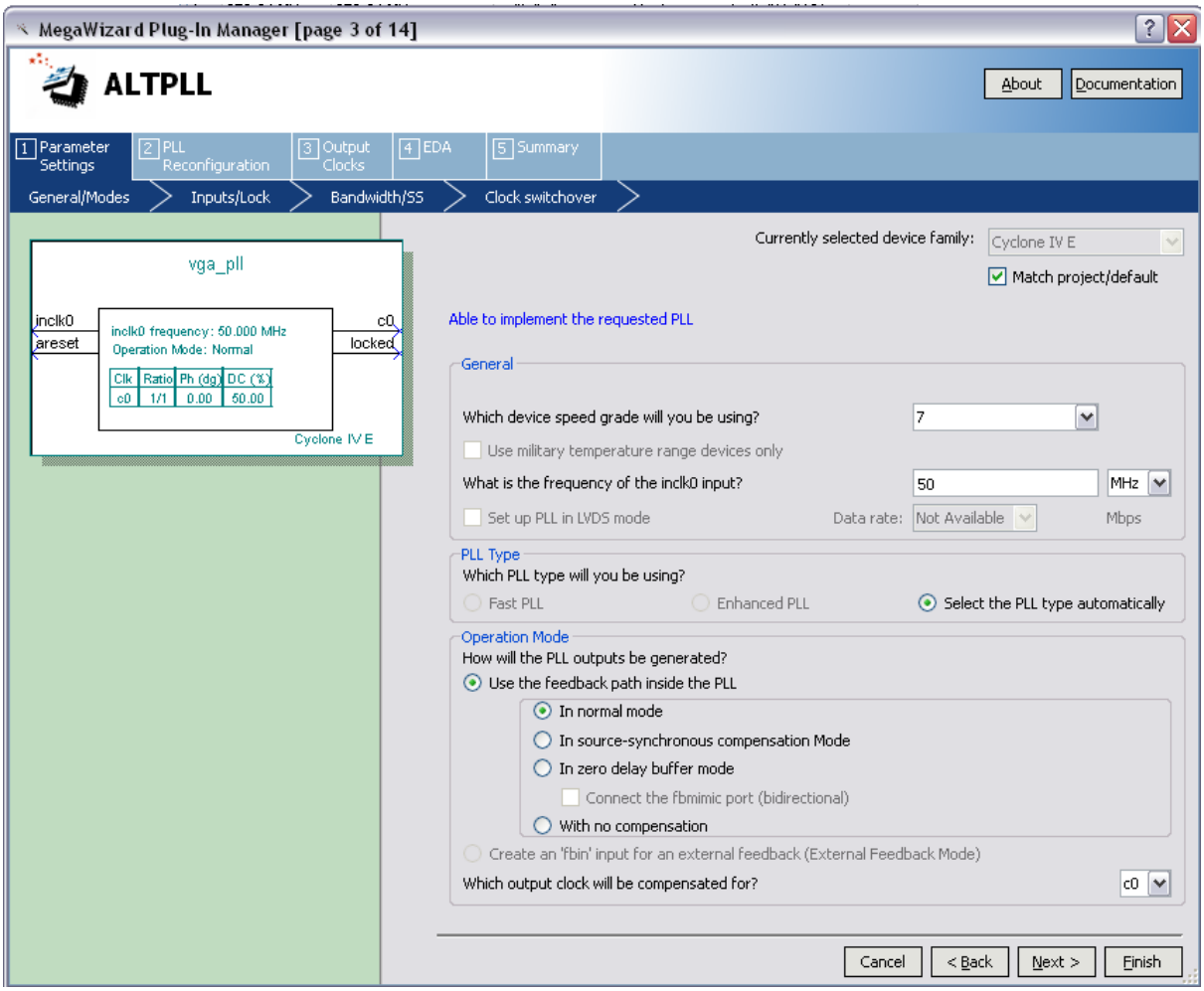
From the Tools menu, select Megawizard Plug-in Manager. Click Next to create a new custom megafunction variation.

Look through the list of available logic blocks, there are options to create floating- or fixed-point arithmetic blocks, communications interfaces and also the Nios II soft-core processor that will be examined in a future lab.

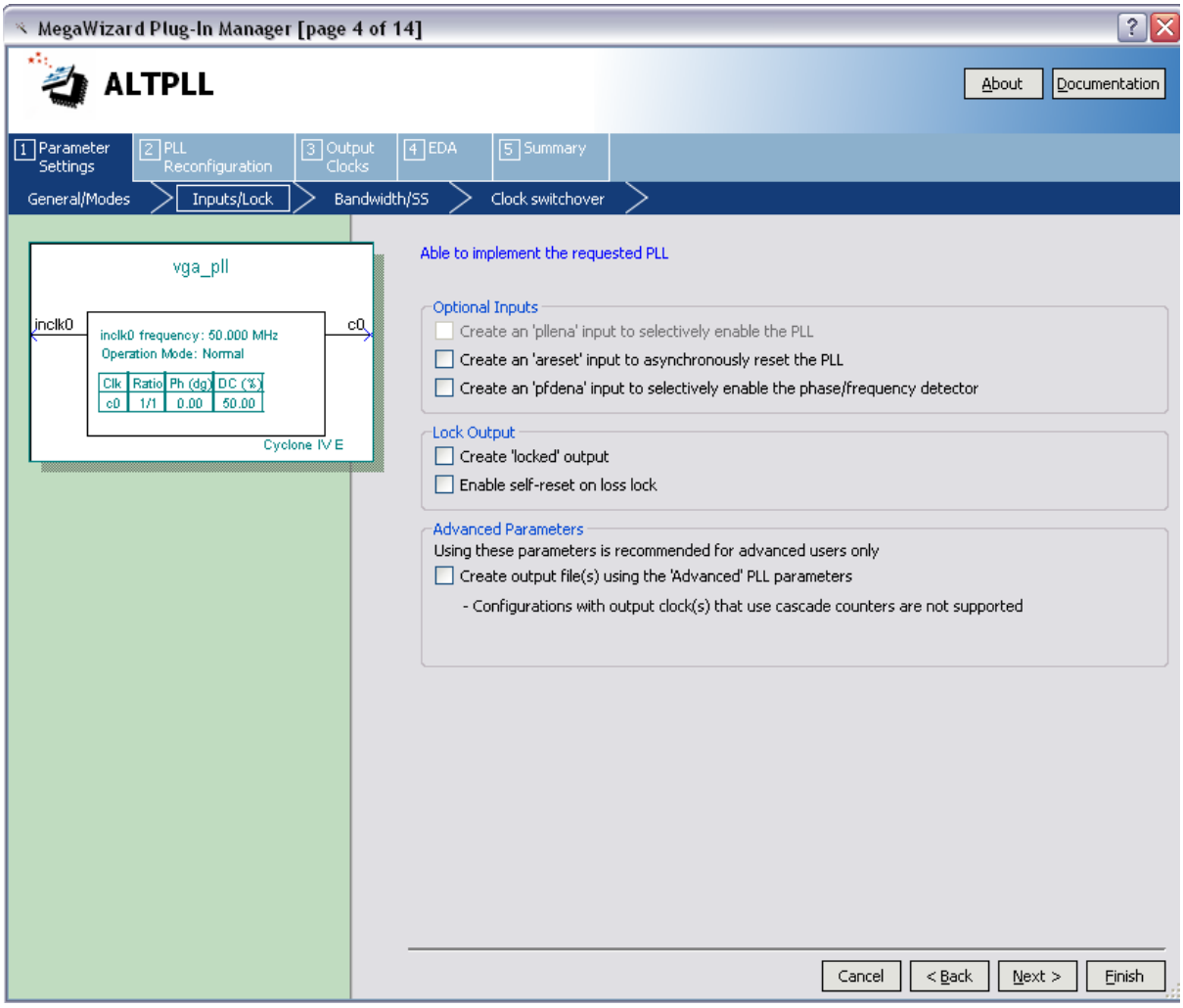
Select the ALTPLL block under the I/O tab and enter '`vga_pll.v`' on the end of the output file path. Ensure that the Verilog HDL language and Cyclone IV E device family are selected. Click Next.



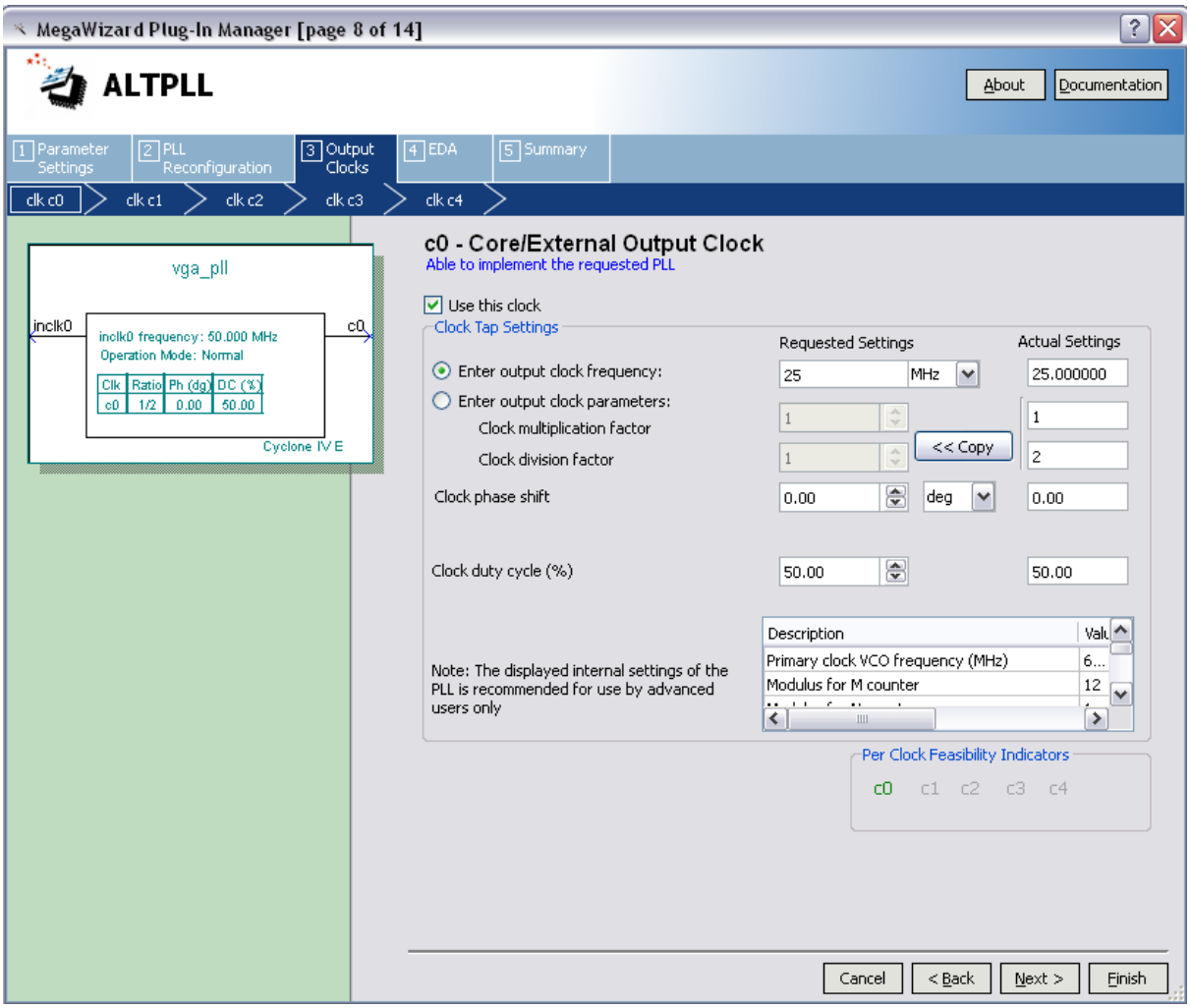
Set the Input Clock frequency to 50MHz and click Next.



Uncheck all the options for extra signals as shown below.

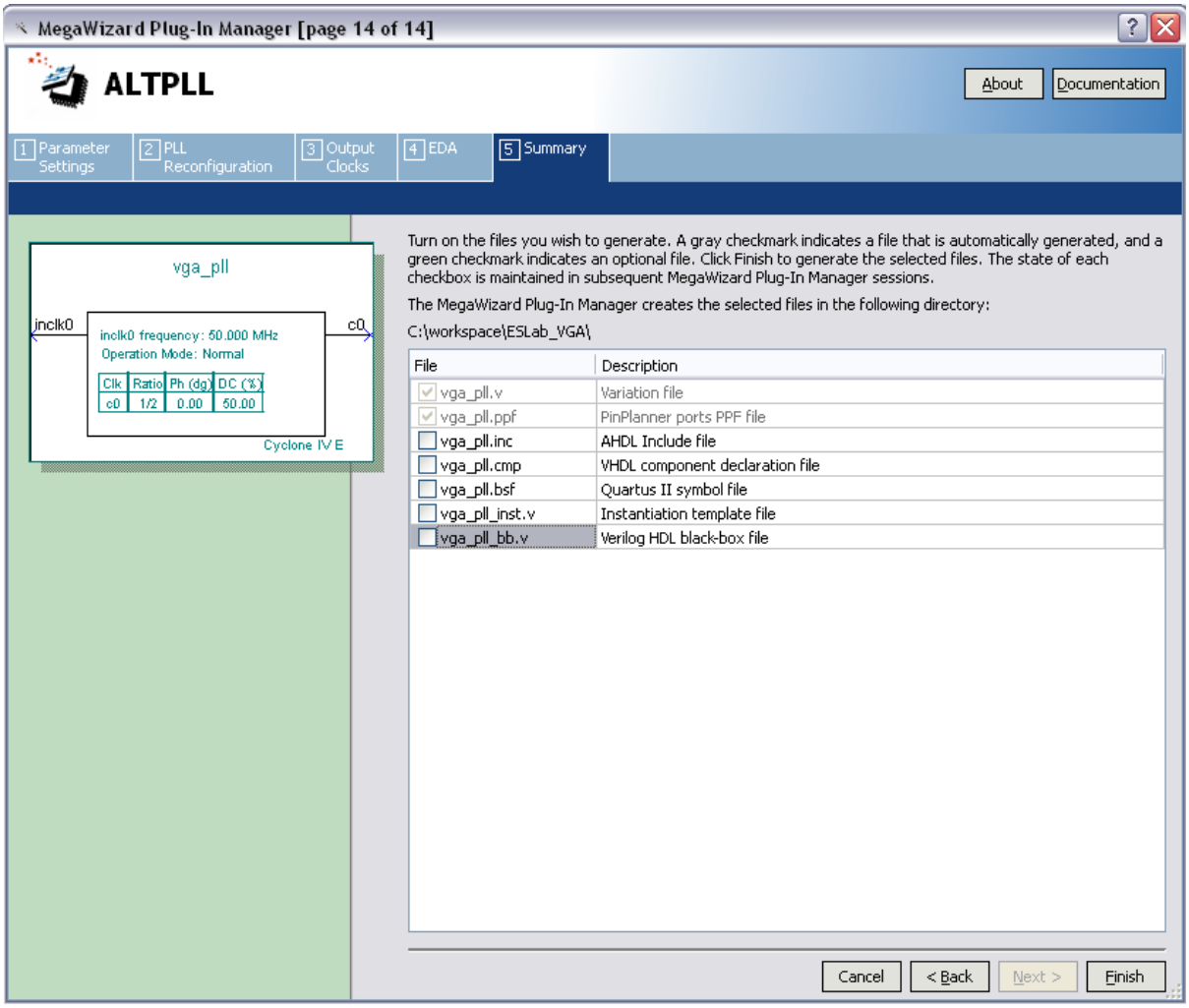


Click Next several times leaving the tabs at their default values until you reach the configuration for clk co shown below. Check "Enter output clock frequency" and set it to 25MHz.



Each PLL can generate up to five clocks (co to c4), each with their own frequency, phase shift and duty cycle. In this case we only need a single clock, so once co is configured, click Finish.

Uncheck the `vga_pll_bb.v` field and click Finish again.



On the "Files" tab of the Project Navigator, you will now see `vga_pll.qip`. Expand that, you will see the source file `vga_pll.v`. Have a look through that file, nothing should be edited by hand however you need to note the module port list so you know how to instantiate the device.

Back in `ESLab_VGA.v`, comment out the `always` block that used to generate the `VGA_CLK` code and write the following instead:

```
vga_pll vpll (.inclk0(CLOCK_50), .c0(VGA_CLK));
```

Remove the '`reg`' specifier from the `VGA_CLK` input (PLLs drive wires) then recompile the design and verify that it still works.

Part 3: VGA Characters

When correctly instantiated, the following module will draw a 10 x 200 pixel white box to the screen with the top-left corner given as an input to the module.

```
module vga_char1 (
      counterX,
      counterY,
      baseX,
      baseY,
      vga_r,
      vga_g,
      vga_b);

input [11:0] counterX;
input [11:0] counterY;
input [11:0] baseX;
input [11:0] baseY;

output [9:0] vga_r;
output [9:0] vga_g;
output [9:0] vga_b;

wire inBox;

assign inBox = counterX > baseX &&
                            counterX < baseX + 10 &&
                            counterY > baseY &&
                            counterY < baseY + 200;

assign vga_r = inBox ? 10'h3ff : 10'b0;
assign vga_g = inBox ? 10'h3ff : 10'b0;
assign vga_b = inBox ? 10'h3ff : 10'b0;

endmodule
```

This box is a very simple way to represent a '1'. Write modules for each of the numbers 0-9, either built out of blocks or using more complicated and pretty tests if you wish. Write code at the top level that uses the switches `SW[2:0]` to select which character is shown on the screen.

Extra Time

If you have spare time, see whether you can get two-digit numbers to display correctly.