ECE385
DIGITAL SYSTEMS LABORATORY
Introduction to USB and IPs

Abstract and Goals

The goal here is to take your existing MicroBlaze system and modify it to support the MAX3421E chip. The MAX3421E is a USB transceiver (sometimes called a Serial Interface Engine) which serves as a USB host. The USB host will then allow you to connect to USB peripherals, such as mice, keyboards, disk drives, game controllers, etc. You will need to add several GPIO modules to control the basic functionality (e.g., reset) on the MAX3421E, as well as a SPI peripheral which will be used to send and receive data from the MAX3421E. The MicroBlaze processor will then run some provided driver code, which will handle the large number of states/cases required to enumerate and read data from a USB keyboard (using the Human Interface Device or HID class).

Modifying the MicroBlaze configuration

For now, we will assume that you are starting from the baseline MicroBlaze configuration as stated in the previous tutorial (INMB), which only has a single GPIO for the blinking LED and the UART.

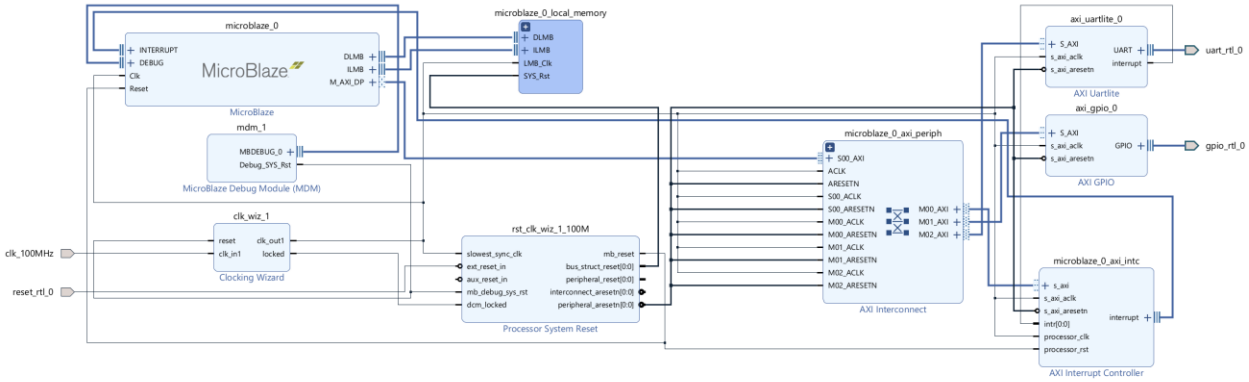


Figure 1 – Baseline MicroBlaze Configuration

The first step is to remove the LED GPIO (e.g., axi_gpio_0), and then add the follow AXI GPIO modules: Make sure you modify their parameters according to the table below, and name them according to the table below. The name is important because parts of the provided USB host code expect to be able to control the hardware pins. You can rename a module by right clicking on the module and selecting Block Properties. Note that the gpio_usb_keycode PIO is dual channel, to allow us to send up to 8 keycodes (using 8-bits per code) – though a standard USB

keyboard will only support 6 simultaneous key presses. Other parameters such as default values may be left at their defaults.

Table 1 - AXI GPIOs

Name	Direction	Width	Enable Interrupt	Dual Channel
gpio_usb_rst	All Outputs/	1	No	No
gpio_usb_int	All Inputs	1	Yes	No
gpio_usb_keycode	All Outputs/All Outputs	32/32	No	Yes

We also need to add an AXI Timer module. USB has many timeouts that require timekeeping in milliseconds. For example, a specific USB mouse might have a polling rate of 10ms which is communicated to the host via the USB descriptor upon enumeration. It is then up to the USB host to keep track of when it has been 10ms since the last readout from the mouse. The timer allows the MicroBlaze core to keep track of when 10 milliseconds (e.g., 1,000,000 ticks at the CPU speed of 100MHz) have passed.

Add the AXI Timer to your design and use the following settings:

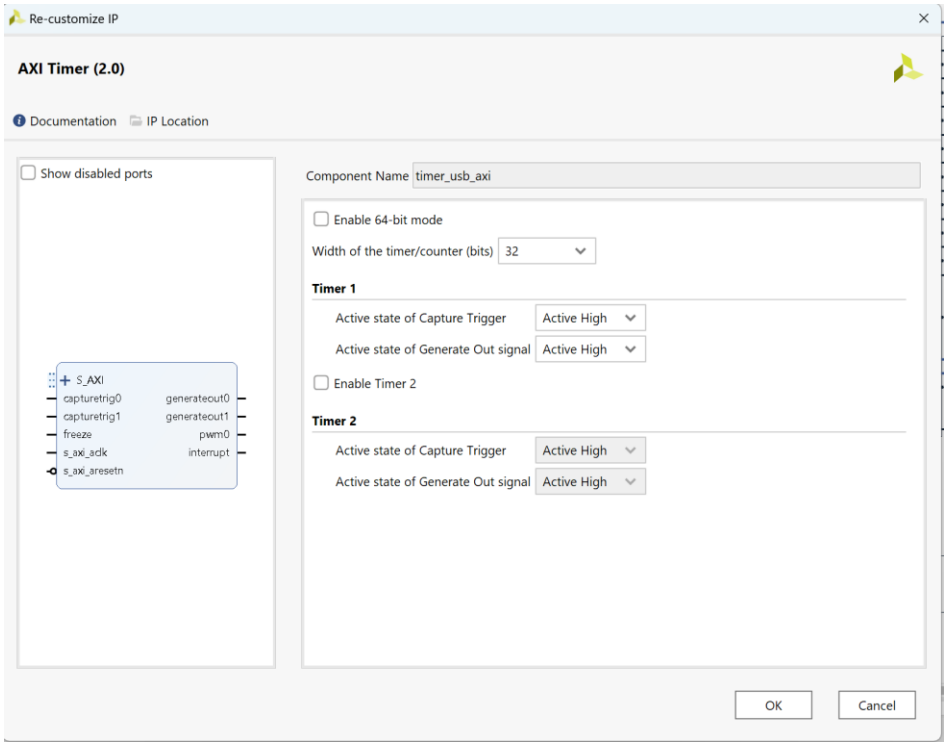


Figure 2 - AXI Timer

Note that the width is only 32-bits, and given that the timer runs at 100MHz, the timer will overflow quickly. However, our timeouts are short and non-critical for USB. Note the timer's name, which is important to match.

Finally, we must add the SPI peripheral, we will use the “AXI Quad SPI”. We will discuss SPI extensively in lecture, but essentially it is a synchronous serial protocol used to communicate with the MAX3421E. Configure it with the following manner (don’t forget to name it correctly).

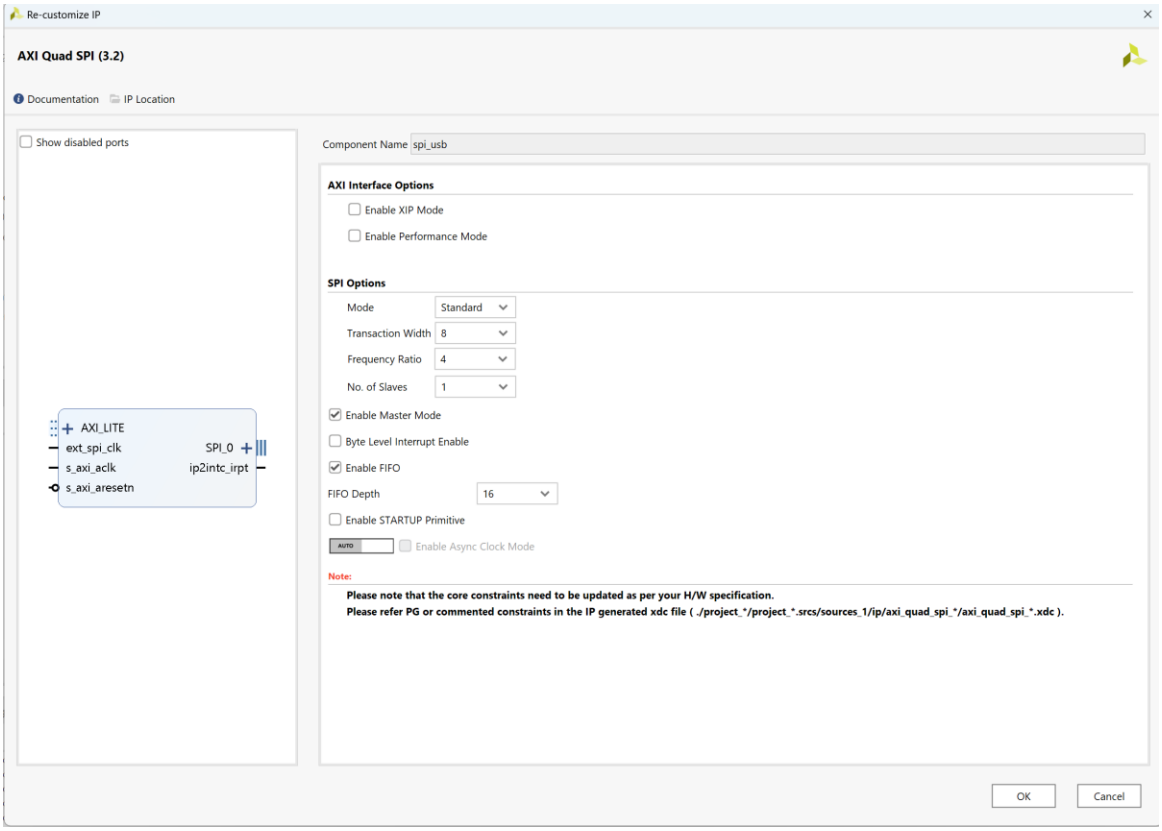


Figure 3 - SPI Peripheral

One point of interest is that the SPI peripheral explicitly has a Frequency Ratio, which is the divisor by which the main clock is divided to generate the SPI clock. In this case, it is 4, which should yield a SPI clock of 25MHz. This is within the specification of the MAX3421E which supports up to 26MHz.

At this point, you may use connection automation to make the general connections. Usually, Vivado is quite good at expanding the width of the peripheral AXI bus and connecting your modules to the AXI bus. However, there are two areas where connection automation usually fails. One is the interrupt controller connections. Recall that we removed the concatenate module previously. We need to replace it now that we have several interrupt sources as its job is to concatenate all the individual interrupts into a single signal to send to the interrupt controller. Add a concatenate module and change it to have 4 ports. Wire the 4 interrupt sources (timer, UART, SPI, and gpio_usb_int) into the 4 ports (the order here doesn’t matter).

The next area where connection automation typically fails is at determining the inputs/outputs to your block design. It will have created the input/output ports (the arrows) correctly, but it has assigned them generic names (gpio_rtl_3). You will need to rename them as follows by right clicking on the port (chevron) and selecting External Interface Properties.

Table 2 - AXI GPIO Input/Output names

Name	Port Name
gpio_usb_rst	`gpio_usb_rst`
gpio_usb_int	`gpio_usb_int`
gpio_usb_keycode	`gpio_usb_keycode_0/gpio_usb_keycode_1`

Finally, connection automation will connect the SPI port for Quad SPI mode by default, while the MAX3421E expects standard SPI (as we discussed in lecture) while quad SPI is typically used for higher speed devices such as flash memories. We will delete the default SPI connection (spi_rtl_0) and click the sideways hamburger menu to expand the individual signals from the SPI. Create 2 1-bit output ports (usb_spi_mosi/sclk) and 1 1-bit input port (usb_spi_miso). You will also need to create a vector [0:0] for usb_spi_ss[0:0]. This is because the module output is a vector/packed array, and Vivado will not allow you to connect a scalar to a packed array. Note that you can create ports by right-clicking on any empty space and clicking create port. To avoid cluttering your block design, move the input (usb_spi_miso) back to the extreme left of the diagram once you have connected it. Finally, connect ext_spi_clk on the SPI peripheral to your AXI clock. Note how I mentioned that the maximum speed for the MAX3421E was 26MHz, rather than 25MHz. If you wanted to get a little bit more performance, you could theoretically generate a 26MHz clock separately and clock the SPI peripheral asynchronously to our AXI clock (though this is not worth it for a 1MHz improvement).

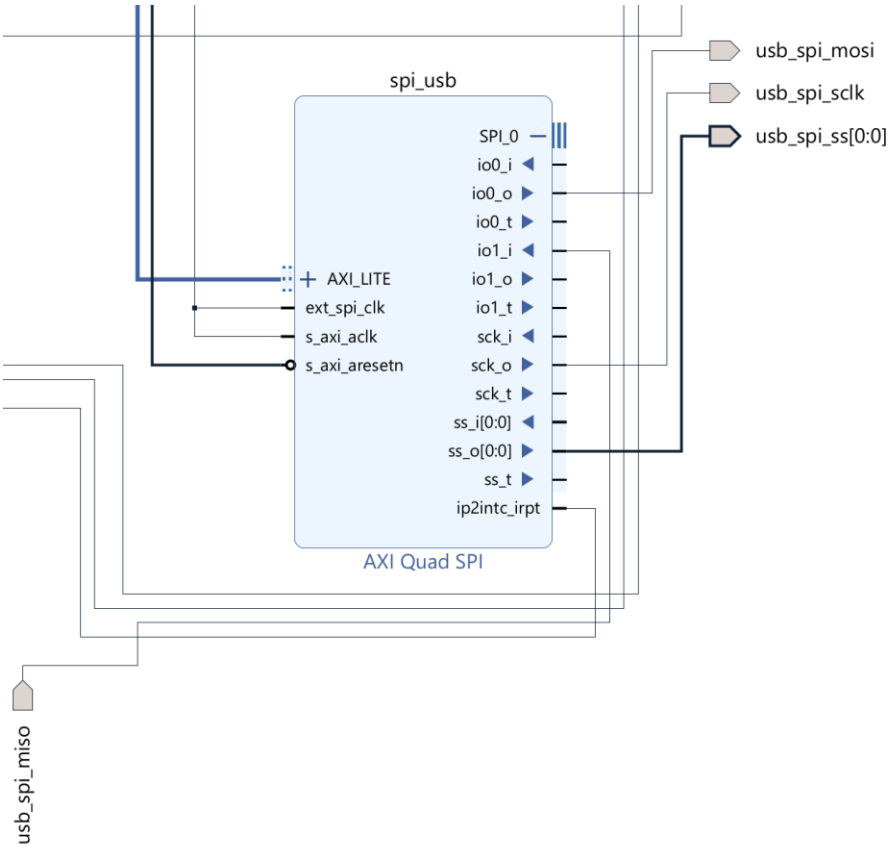


Figure 4 - Connecting the Quad SPI Peripheral for Standard SPI

Finally, right click somewhere (but not on a module) in your diagram tab and select “Validate Design”. This will ensure that all the connections which were made are valid. Your final design should resemble the following:

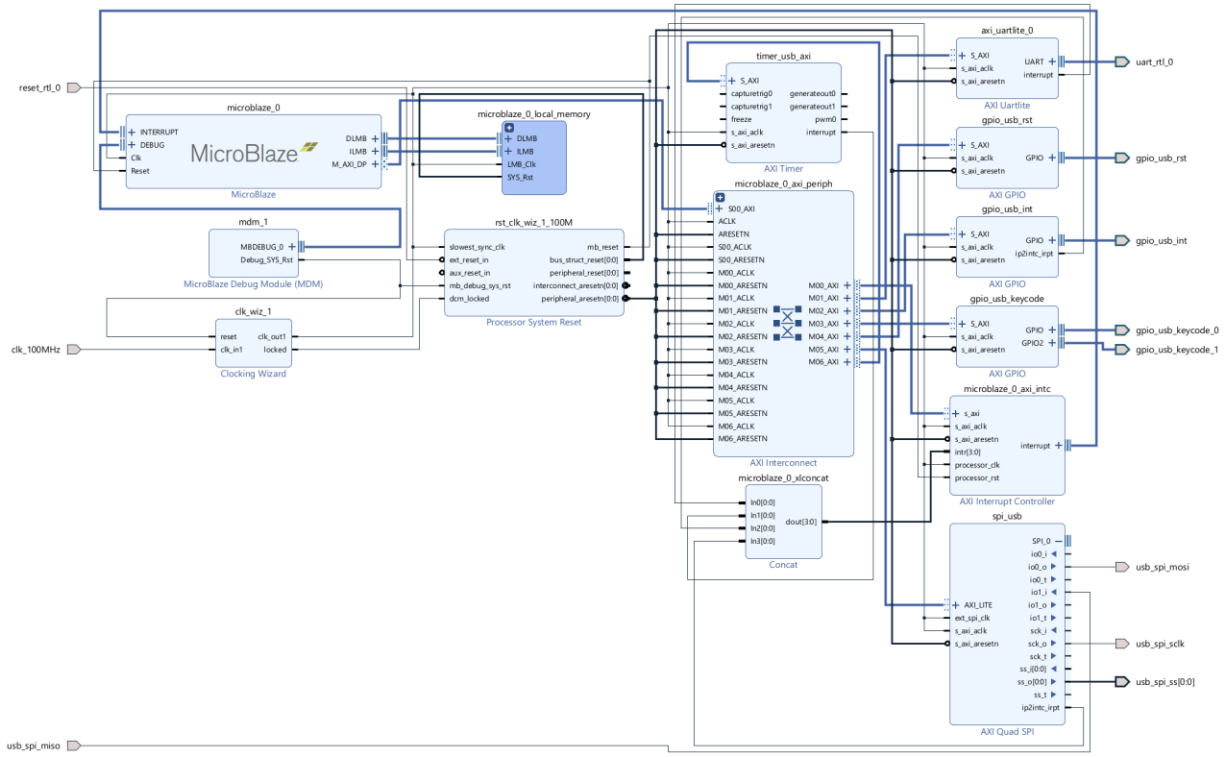


Figure 5 - MicroBlaze with USB Peripherals

Save your design and close out the block design editor. Note that in this case I’ve named the block design ‘mb_usb’, though it can have a different name (you just have to edit to provided top-level to match up).

You can now return to Vivado and import the provided files for the lab. However, before we start the FPGA build, we need to create a new IP repository and import the Real Digital VGA to HDMI transmitter, which is provided. If you haven’t already, unzip the hdmi_tx_1.0 folder of the provided files to your project directory. Go to the Project Manager (left bar) and click on IP Catalog. Then right click on Vivado Repository and click Add Repository. Navigate to where you unzipped the folder in your project directory and select OK. It should find 1 IP and import it into a new local repository.

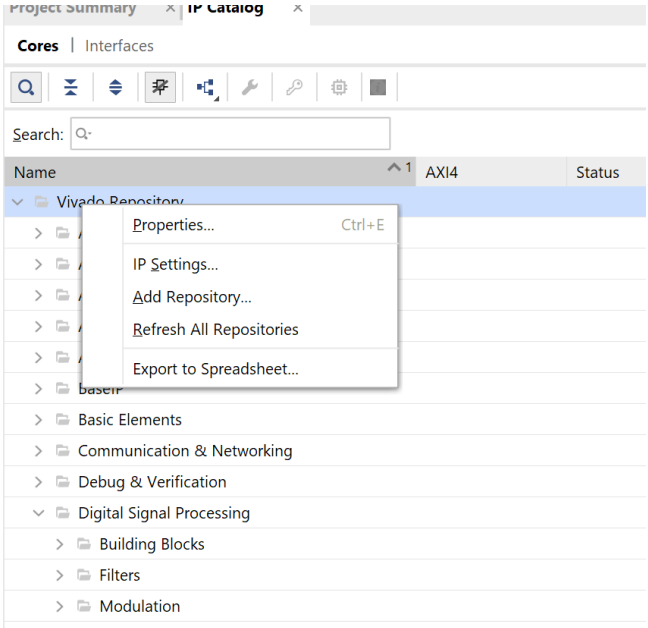


Figure 6 – Add Repository

Now navigate to the new IP you imported. Double click on the IP to create an instantiation, as you previously did with the on-chip memory for the SLC3. Configure the HDMI IP as follows:

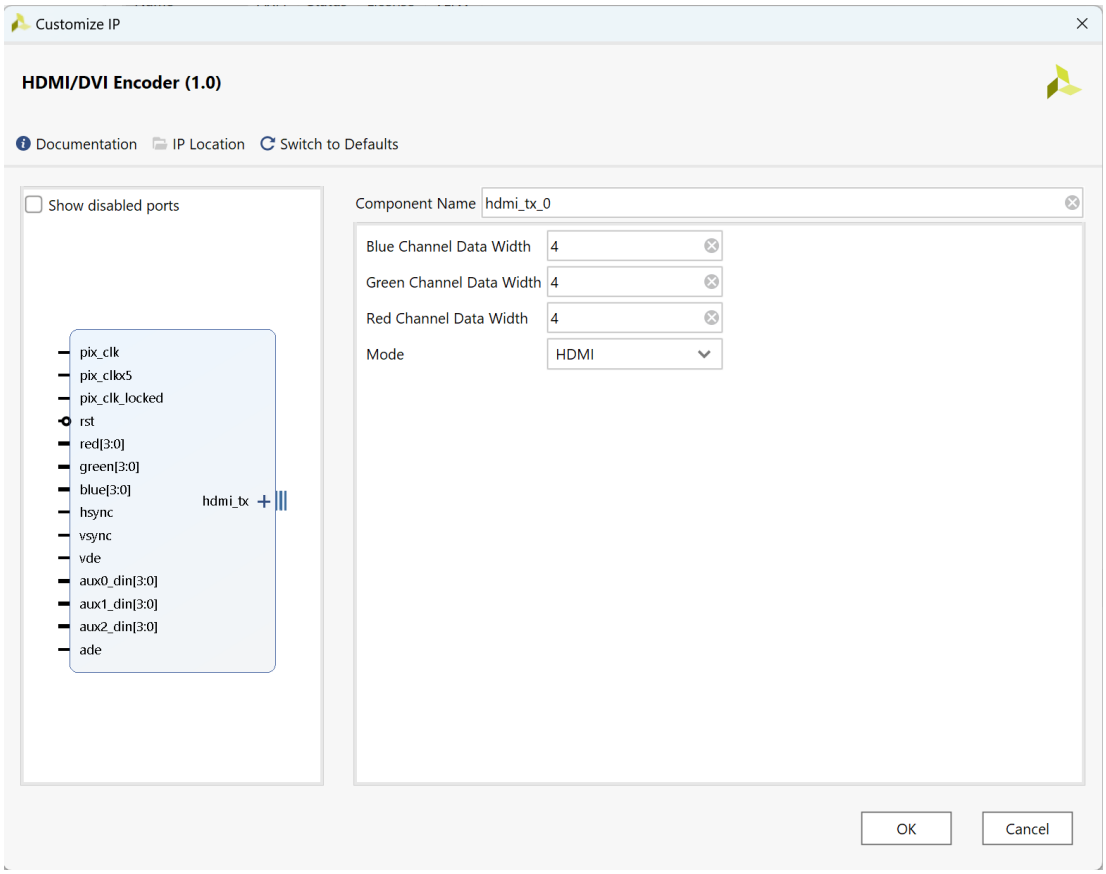


Figure 7 - HDMI IP Configuration

Note that this only allows us 2^12 colors in total, or 4096 possible colors. HDMI 1.0 which is what we will implement, allows for a maximum of 8-bits per channel or 32-bit (4 billion) colors, which you may revisit for your final project. Click on and continue with the 'out of context' run.

You also need to create an instance of the "Clocking Wizard" IP. HDMI requires a pixel clock and a 5x TMDS clock. The pixel clock will be 25MHz (for 640x480x60Hz), and the 5x TMDS clock will be 125MHz. We will discuss HDMI in lecture, but this is a good time to do a little research on HDMI and how TMDS (Transition-Minimized Differential Signaling) works. Find the Clocking Wizard IP and double click to customize it. Tab over to output clocks and enter the following parameters. Other parameters should be left as the default.

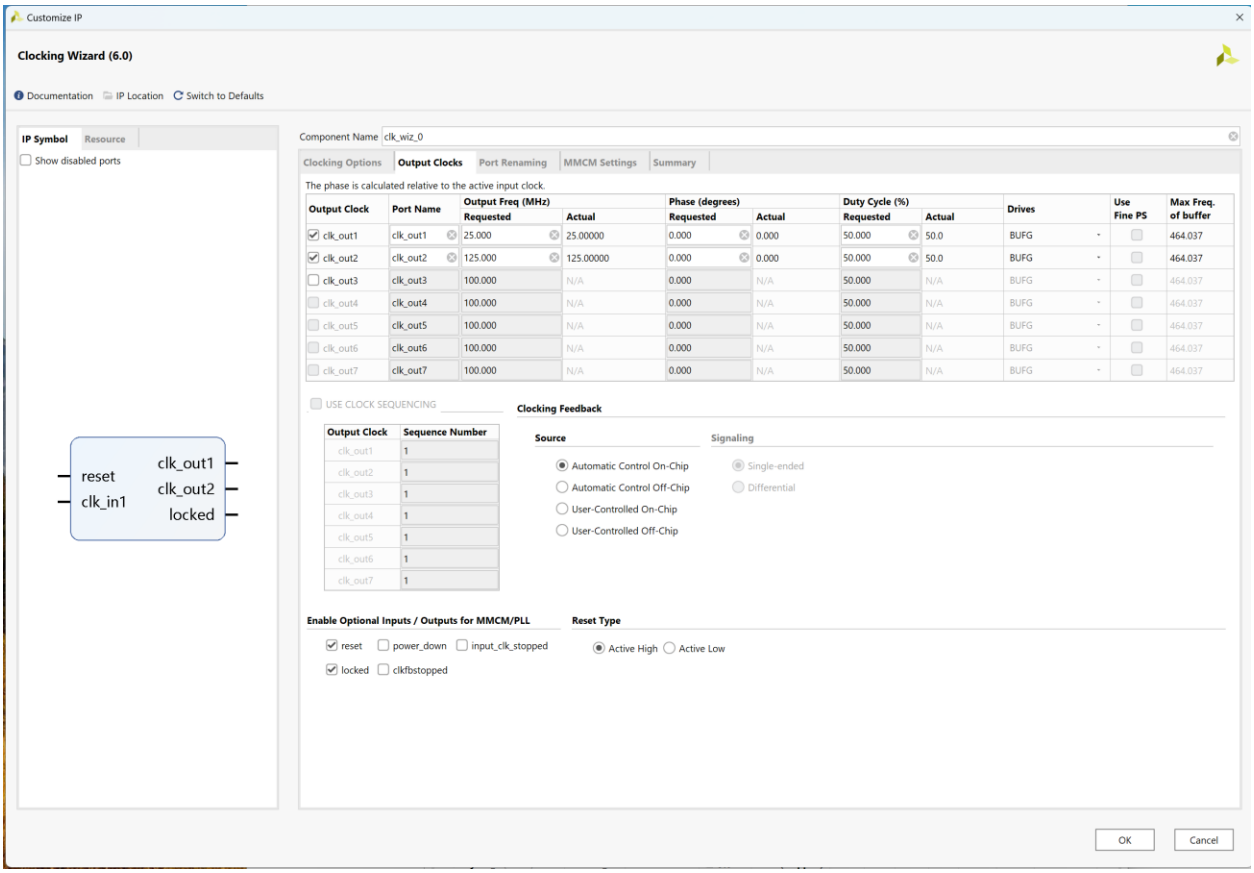


Figure 8 - Clocking Wizard for HDMI

Finally, import the provided .sv files and pin assignments (make sure check the box to copy these into your project). You may also have to import the HexDriver module from previous labs. Run synthesis and make sure there are no errors or critical warnings (typical errors come from having misnamed signals or misnamed modules). Also, open the implemented design and go to the assignment editor to make sure all the top-level ports are mapped to I/O pins.

Remember to then Export the hardware, click next and make sure you select “Include Bitstream” in the following screen:

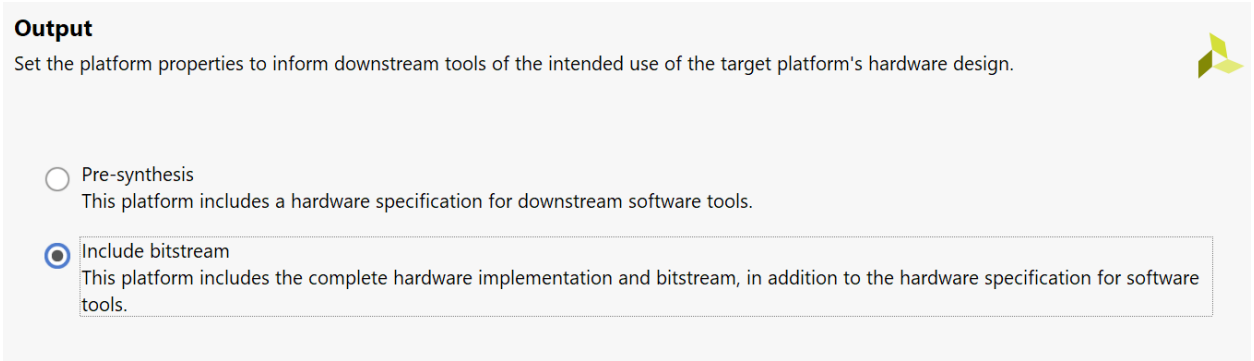


Figure 9 - Make sure you "Include Bitstream."

It is possible to update your projects from the previous lab, but it may be easier just to create a new workspace and a new software project according to INMB “Software Setup”. Create the application project from the “Hello World” template as before and import the provided .c and .h files (note that you will need to maintain the directory structure, so you should make a new folder in your software project named “lw_usb” and then move the relevant files into the new folder). Also be sure to delete the provided helloworld.c.

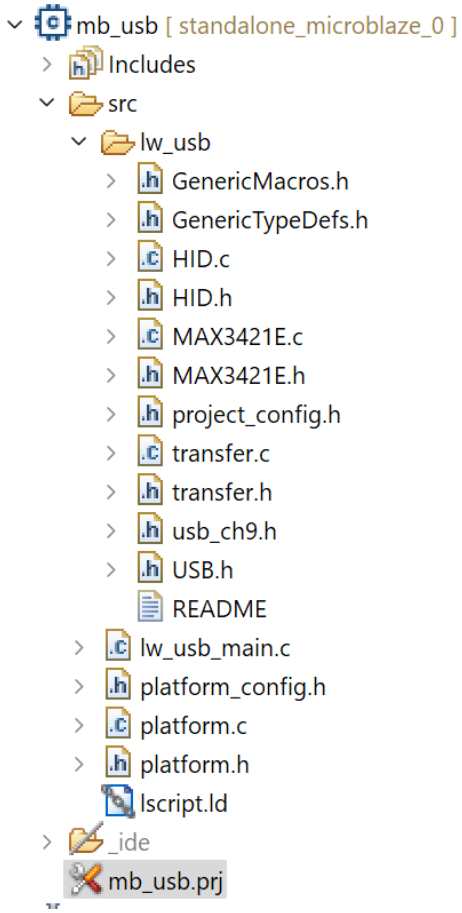


Figure 10 - Imported Source Files

USB Host Programming with the MAX3421E Chip

The provided USB host driver is mostly complete, with support for USB keyboards and USB mice. However, you will be required to fill in some functions inside the MAX3421E.c file. These functions pertain to reading and writing registers through SPI to the MAX3421E chip.

For the USB code to work, you must fill in the relevant MAX3421E.c functions:

```
void MAXreg_wr(BYTE reg, BYTE val)
```

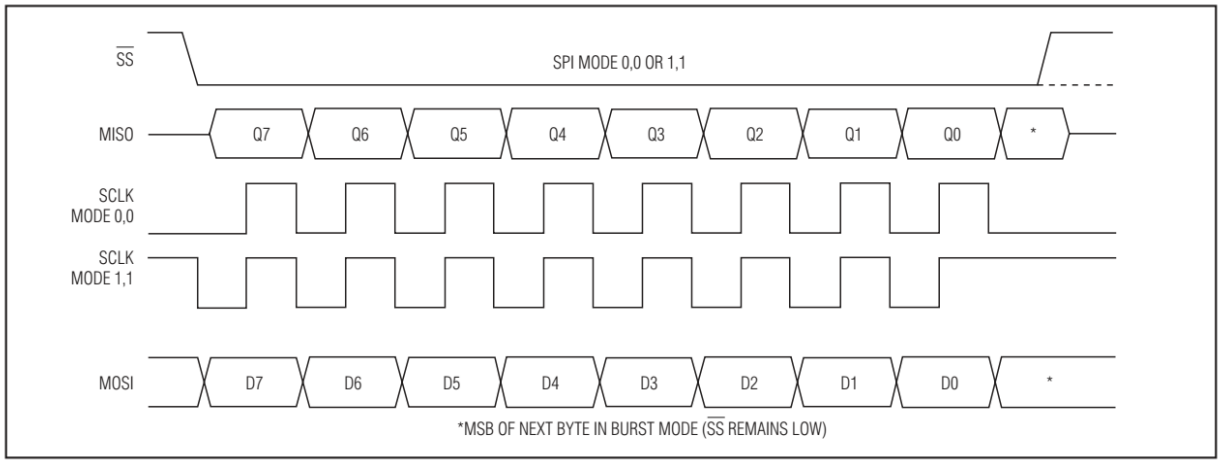
```
BYTE* MAXbytes_wr(BYTE reg, BYTE nbytes,
BYTE* data)
```

```
BYTE MAXreg_rd(BYTE reg)
```

```
BYTE* MAXbytes_rd
```

SPI is a synchronous serial bus, consisting of 4 signals called CLK, MOSI, MISO, and SS. CLK is the clock signal, which is transmitted from the master device (MicroBlaze) to the slave devices (MAX3421E). MOSI is a data signal which stands for master-out slave-in, which transmits data from the Nios II -> MAX3421E – synchronous to the CLK. MISO is a data signal which stands for master-in slave-out, which transmits data from the MAX3421E -> MicroBlaze – also synchronous to the CLK. SS stands for slave select, which allows a specific slave device to be selected when SS is low. Multiple slave devices may share the same SPI bus by sharing MOSI, MISO and CLK lines, but SS must be a unique connection between the master device and each slave device.

The timing diagram for the MAX3421E generally looks as follows. Note that there are 4 SPI modes corresponding to the various polarities between clock and data, but the MAX3421E supports both 0,0 and 1,1. Our SPI peripheral (above) should be pre-configured to operate in mode 0,0 with a clock rate of 2.5 MHz. Note that the following is a *full duplex* transfer, as 8 bits are read and written (through MISO and MOSI) respectively during the same 8 clock cycles. The MAX3421E starts up in half duplex mode, but the first SPI operation (from MAX3421E.C) configures the FDUPSPI bit, switching it to full duplex mode. You should understand this behavior in the MAX3421E by reading the datasheet starting from page 19. Note that to (for example) read a register on the MAX3421E, first you need to write the register address through SPI and then perform a read through SPI.



Once you understand the various operations the USB device drivers need to communicate via SPI (the comments in the provided function stubs will tell you what each function should do), you must implement them using the AXI Quad SPI peripheral. Although the peripheral is well documented (Xilinx AXI Quad SPI Peripheral) the documentation mostly covers the low level hardware, while we would like to use the provided Xilinx device drivers (xspi.h). For most vendors, the easiest way to understand the libraries is to find an example, in this case the following example: XSPI Polled Example is a reasonable approximation of what we need the SPI port to do.

Although for this lab, it is only necessary to read 1 keycode (the list of keycodes is provided below), the USB HID protocol actually supports up to 6 simultanous keycodes. Finally, the provided USB driver will also accept a standard USB mouse which you may make use of for your final project.

0	1	2	3	4	5	6	7	8	9	10	11
-	err	err	err	A	B	C	D	E	F	G	H
12	13	14	15	16	17	18	19	20	21	22	23
I	J	K	L	M	N	O	P	Q	R	S	T
24	25	26	27	28	29	30	31	32	33	34	35
U	V	W	X	Y	Z	1	2	3	4	5	6
36	37	38	39	40	41	42	43	44	45	46	47
7	8	9	0	Enter	Esc	BSp	Tab	Space	- / _	= / +	[/ {
48	49	50	51	52	53	54	55	56	57	58	59
] / }	\ / |	...	; / :	' / "	` / ~	, / <	. / >	/ / ?	Caps Lock	F1	F2
60	61	62	63	64	65	66	67	68	69	70	71
F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	PrtScr	Scroll Lock
72	73	74	75	76	77	78	79	80	81	82	83
Pause	Insert	Home	PgUp	Delete	End	PgDn	Right	Left	Down	Up	Num Lock
84	85	86	87	88	89	90	91	91	91	94	95
KP /	KP *	KP -	KP +	KP Enter	KP 1 / End	KP 2 / Down	KP 3 / PgDn	KP 4 / Left	KP 5	KP 6 / Right	KP 7 / Home
96	97	98	99	100	101	102	103	104	105	106	107
KP 8 / Up	KP 9 / PgUp	KP 0 / Ins	KP . / Del	...	Applic	Power	KP =	F13	F14	F15	F16
108	109	110	111	112	113	114	115	116	117	118	119
F17	F18	F19	F20	F21	F22	F23	F24	Execute	Help	Menu	Select
120	121	122	123	124	125	126	127	128	129	130	131
Stop	Again	Undo	Cut	Copy	Paste	Find	Mute	Volume Up	Volume Down	Locking Caps Lock	Locking Num Lock
132	133	134	135	136	137	138	139	140	141	142	143
Locking Scroll Lock	KP ,	KP =	Internat	Internat	Internat	Internat	Internat	Internat	Internat	Internat	Internat
144	145	146	147	148	149	150	151	152	153	154	155
LANG	LANG	LANG	LANG	LANG	LANG	LANG	LANG	LANG	Alt Erase	SysRq	Cancel
156	157	158	159	160	161	162	163	164	165	166	167
Clear	Prior	Return	Separ	Out	Oper	Clear / Again	CrSel / Props	ExSel			
224	225	226	227	228	229	230	231				
LCtrl	LShift	LAlt	LGUI	RCtrl	RShift	RAlt	RGUI				

List of Keycodes