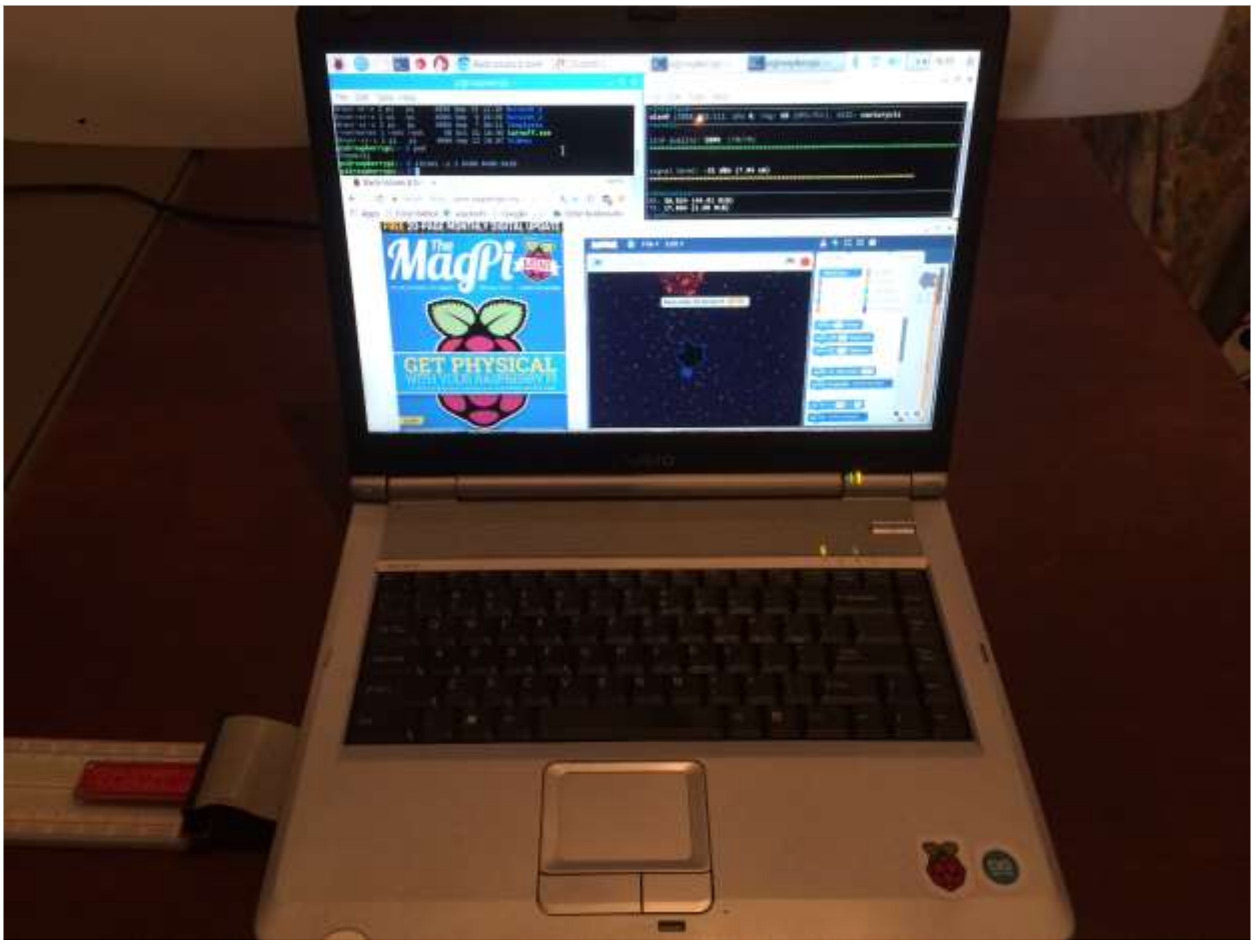
Raspberry Pi & Teensy Laptop Conversion



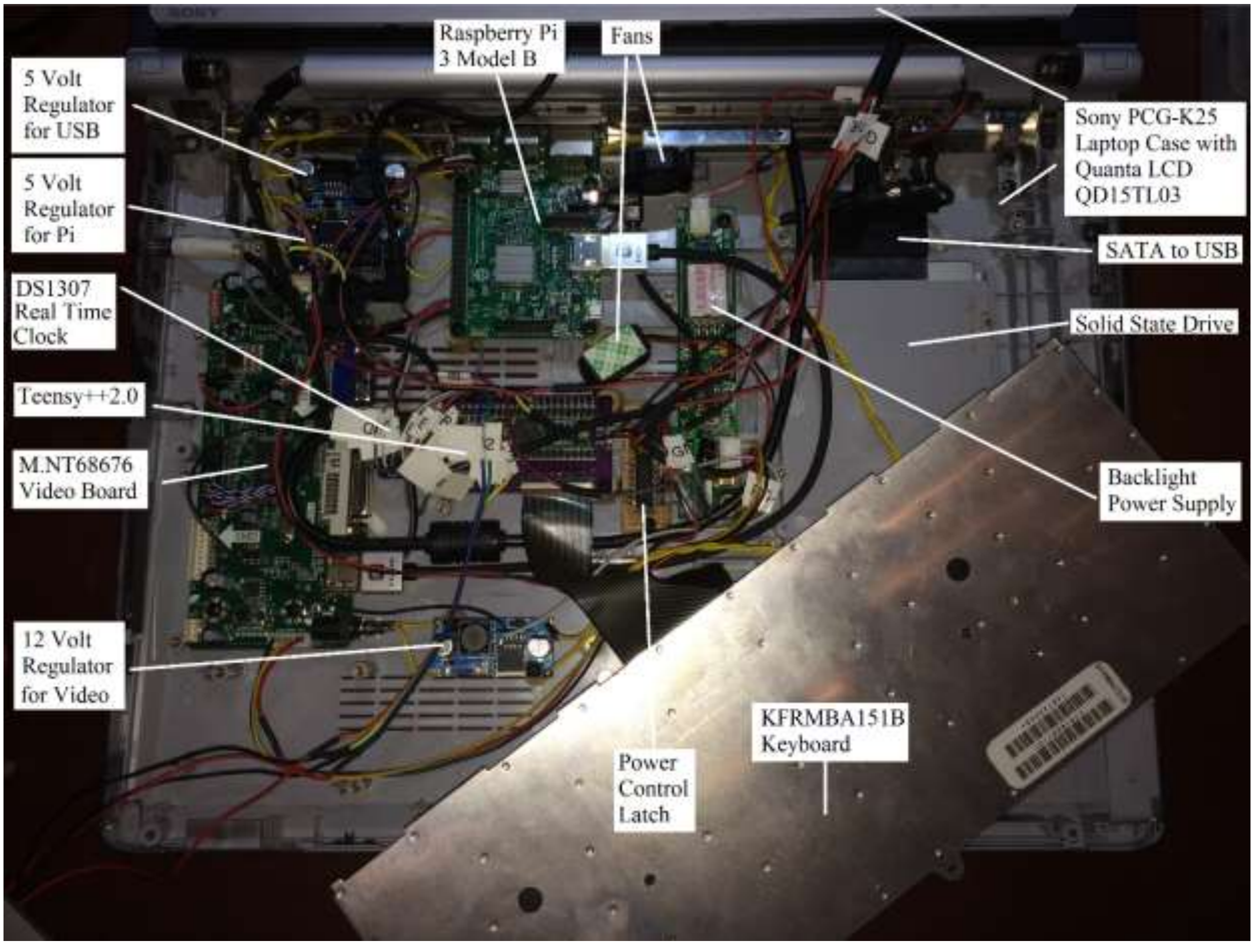
Introduction – This document will describe how I converted a Sony Vaio PCG-K25 computer into a Raspberry Pi 3B and Teensy ++2.0 laptop. Most of the circuits, software, and modifications are based on information available on the internet. The circuits in the laptop are listed below and then described in detail in the following pages:

1. A Teensy ++2.0 is used to interface the keyboard and touchpad with the Pi via USB. This required a new circuit board to route the keyboard connector to the Teensy I/O pins. A Teensyduino sketch scans the keyboard and touchpad and sends changes to the Pi over USB.
2. The HDMI from the Pi is converted to LVDS for the display by an M.NT68676 video board. The video cable had to be lengthened to reach into the laptop base.
3. The control pad for the M.NT68676 video board was replaced by the Teensy.
4. The Pi boots from a 240GB solid state drive using a SATA to USB cable. The micro SD card has been removed.
5. The laptop Wi-Fi antenna is connected directly to the Pi for improved reception. A connector was added to the Pi that fits the laptop antenna cable.
6. A Real Time Clock and the Teensy are connected to the Pi over the I2C bus.
7. Three separate Buck regulators provide power to the M.NT68676 video board, the Pi, and the 4 USB ports. The regulators were modified so they can be enabled with a control signal.
8. The laptop power switch turns on the regulators and the Teensy turns off the regulators. This is accomplished with a NAND gate latch circuit that controls the regulator enable signal.
9. The laptop LEDs were rewired to show incoming power, regulator power, caps lock, and code debug.
10. The Teensy will reset the Pi if Control-Alt-r is typed or shut down the laptop if Control-Alt-s is typed. These actions can also be initiated by the Pi over the I2C bus.
11. The Pi GPIO signals are brought out the side of the laptop for bread boarding.
12. The Pi audio is cabled to a 3.5mm jack on the side of the laptop for headphones.
13. Audio sent to the M.NT68676 board via HDMI is connected to the laptop speakers.
14. A fan blows over the Pi and another fan blows out the back. Testing shows this gives adequate cooling to the Pi when overclocked at 1300 MHz.

A parts list with vendor links is located in the Appendix. I spent over $225 to make this laptop so I can't claim it was cheap to build but I had a lot of fun and learned quite a bit. Many of the circuits are not essential which can bring the price down and several items could only be purchased in multiple quantities. If I build another laptop, I'll already have several of the components in my spares drawer. I experimented with powering the Pi and Teensy from the 5 volt regulator on the video card to eliminate the 3 buck regulators. The test results are in the Appendix.

Battery operation is not included but may be added later. See the Appendix for information regarding the battery charging circuit.

Board Mounting – The locations of all the boards in the laptop are shown in the following picture. Wood dowels were cut to make standoffs for the boards. The dowels were screwed to each board and then the dowels were attached to the laptop floor with JB Weld epoxy.



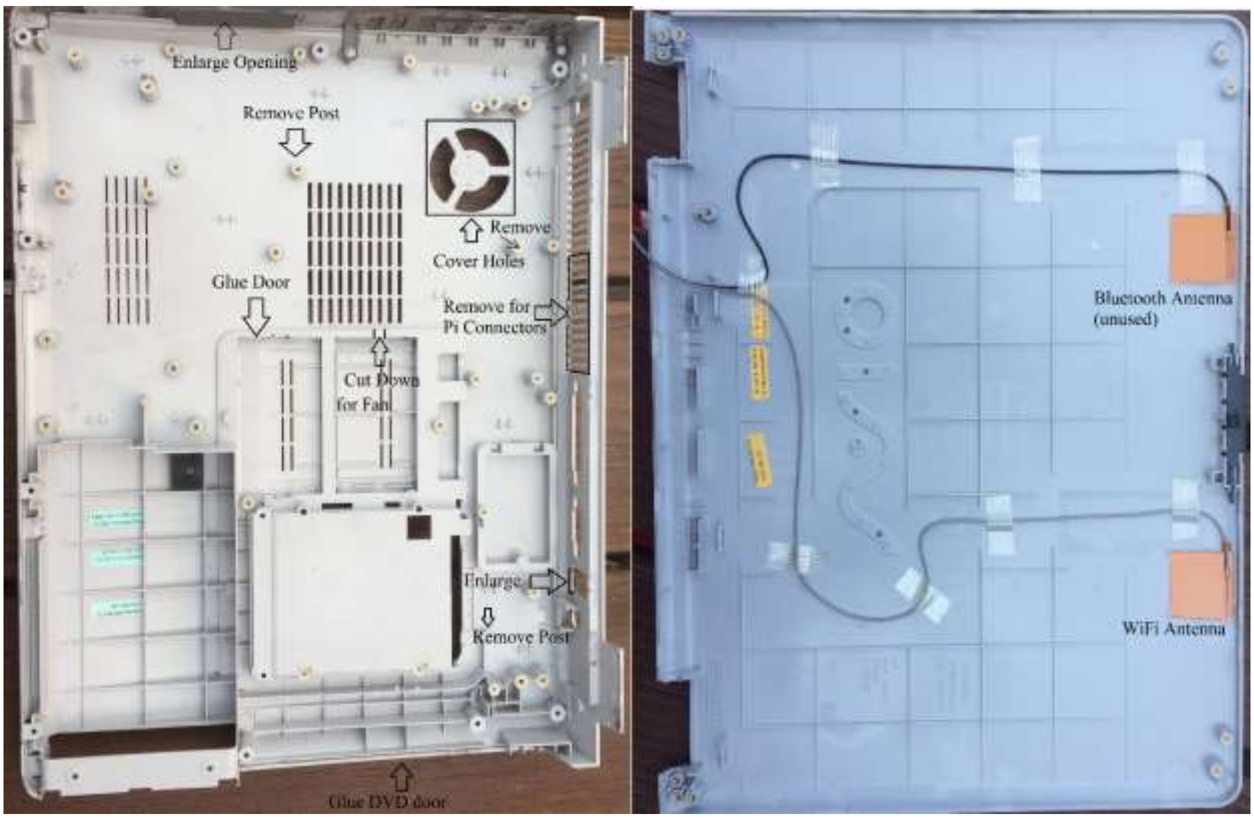
This picture shows the metal shield that helps to stiffen the keyboard.

First Steps – I was given a nonfunctional Sony laptop specifically for this project. Unlike modern ultra-thin laptops, this one has over an inch of thickness at the back so the Pi's USB and network connectors don't need to be removed to fit in the case. The laptop was in good shape but I needed to know if the LCD and keyboard were functional. After some troubleshooting, I was able to get the BIOS to boot up (there was no hard drive) and the display and keyboard worked fine. Using my Seeed DSO oscilloscope, I started probing the 24 keyboard signals at the flexible printed circuit (FPC) connector. I found 16 of the signals were driven by the motherboard and pulsed low every 30 msec. The remaining 8 signals are inputs to the motherboard and were at 5 volts but would pulse low if I pressed certain keys. Powering down the laptop and switching to an ohm meter, I was able to determine the keyboard connections by systematically checking for a connection as I pressed each key. It's a long tedious process but I eventually created a matrix of the keyboard connections. I can see why people write software routines that automatically determine the matrix as each key is pressed in sequence.

I probed the touch pad connector pins with the scope but saw no clock or data pulses because the touchpad had not been initialized by the BIOS. I found the 5 volt and ground connections and two other pins that measured 5 volts but my ohm meter showed no connection to the 5 volt bus. I assumed these pins were the clock and data for the PS/2 interface and would figure which was which once I had the Teensy up and running.

This is what the laptop looked like before everything was removed:

Sony Vaio PCG-K series disassembly – I followed the "Inside my laptop" directions for disassembly of the base and LCD sections. I kept unscrewing everything and removed all metal shielding until I had a bare case as shown below. I used a Dremel tool and a file to make the cuts shown by arrows. JB Weld two part epoxy was used to glue the memory access door shut and to permanently secure the DVD Door. I blocked the 3 fan holes with Duct Tape and then poured in JB Weld. I should have removed all of the original motherboard mounting posts so I didn't have to come back later and Dremel another one each time I tried to mount a board.



Keyboard – I chose a Teensy ++2.0 for my keyboard controller because I wanted extra I/O for controlling the touchpad, video card and anything else that might come up without adding multiplexing chips. As I expected, I've used up all of the Teensy's digital I/O pins. This Teensy has been around for a while so I figured the software was stable and there would be lots of examples to follow. The Teensy developer, PJRC has created an Arduino add-on called Teensyduino that allows the controller to communicate via USB as a Human Interface Device (HID), aka keyboard and mouse. I followed the PJRC steps for installing Arduino and Teensyduino on a PC and then started experimenting with the Keyboard.set_key and Keyboard.set_modifier commands. I used PJRC's "Micro Manager" method to send the keyboard data over USB.

The finished sketch is a complete keyboard controller routine that scans the keys every 30 msec and sends the keypresses to the Pi over USB. I'm a hardware guy so don't expect pretty code but it works and can certainly be improved on. The Modifier keys (Ctrl, Alt, Shift, and GUI) can be pressed at the same time in any order and they will be sent to the Pi. The rest of the keys (called normal keys) are sent in 6 different slots so that you can have more than one key pressed at a time. The program sends the keypress in slot 1 if it's not in use, otherwise it moves on to slot 2 on up to slot 6. If the keyboard had diodes to isolate the keys, then up to 6 keys could be pressed at the same time. Unfortunately there are no diodes so ghosting causes extra keys to be sent after 3 or 4 keys are held down. Go to Geekhack to learn more about keyboard controllers and check out the TMK Keyboard Firmware Collection for an example of a far more sophisticated keyboard routine. For everything that I do, the keyboard has worked perfectly including playing Scratch games, typing this document, and general Linux stuff. An earlier version of my keyboard routine sent all keypresses in slot 1 and there were lots of times that I would type one key and not quite release it before typing another key which would cause the second key to be missed. And a real deal breaker was when my Scratch games wouldn't move diagonally because the up arrow and right arrow couldn't be pressed at the same time.

My Teensy code is available on github.

There are two code oddities that a better coder will hopefully figure out. My guess is that I'm supposed to send some handshake messages but I've made it work by adding delays. If you know a better solution, let me know.

1. My Teensy code waits 30 seconds at startup to let the Pi boot up. I came up with this time delay by trial and error until I found what worked. If the delay isn't right, the Pi does something to the Teensy that causes the Teensy to lock up. You can tell when the Teensy is locked up because the on-board "heartbeat" LED (that I've programmed to blink) is stuck off. When I hook the Teensy to my PC and start up the PC, I get the same error and the 30 second delay fixes the problem.

2. Whenever the Teensy code detects a key is pressed or released, it sends the information over USB but there is a special case for the Caps Lock key. When it is pressed or released, the code must wait 10 milliseconds after sending the information before proceeding to scan for other keys. Without the delay, the Teensy will lock up and the "heartbeat" LED will stop blinking. When I hook the Teensy to my PC, the Caps Lock key works fine without any delay.

The keyboard part number is KFRMBA151B and according to the web, it is used on Sony Vaio Laptops, PCG-K12P, PCG-K13, PCG-K14, PCG-K15, and PCG-K17

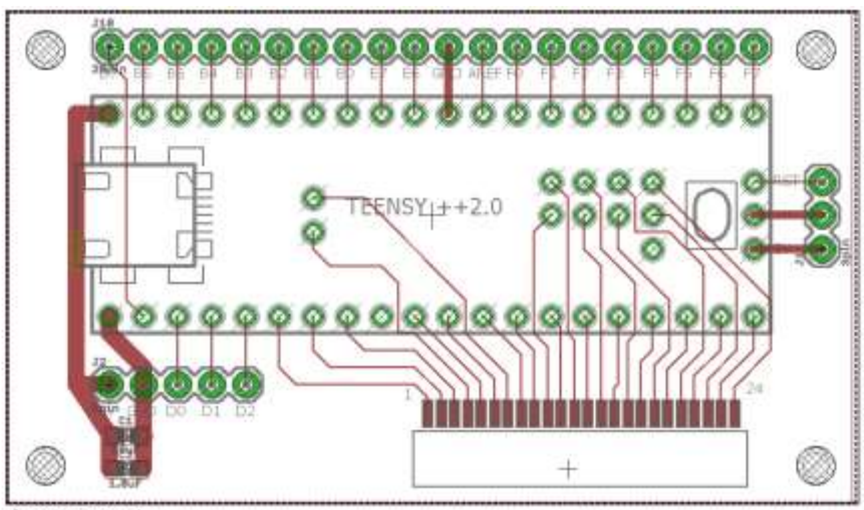
Keyboard Matrix – The 8 columns across the top of the matrix are inputs to the Teensy. The 16 rows on the left side are outputs from the Teensy. Only one row is driven low at a time and the other 15 rows are set to high impedance. The 8 columns have pull ups inside the Teensy so if a key is not pressed, it is read as a logic high. A keyboard scan is repeated at 30msec intervals and any key changes are reported over USB. I couldn't find a connection for the print screen and num lock keys on my keyboard so I'm assuming they're broke. I don't need num lock but I might want a print screen key so I use the Menu key as a print screen.

Firmware		Columns	0	1	2	3	4	5	6	7
	Teensy	Inputs	E0	E4	C1	A0	C2	A5	C4	A6
Rows	Outputs	Conn Pin#	6	7	11	12	13	14	17	18
0	D3	1			CTRL-R				CTRL-L	
1	D4	2		ARROW-L	ARROW-D	ARROW-U	PAGE-D	PAGE-U	END	ARROW-R
2	D5	3		ENTER		]		=	"	
3	E5	4	F12	MENU/PrtSc	/	;	[	P	-	BCKSPACE
4	D7	5	INSERT				\	HOME	L	DELETE
5	E1	8	F10	COMMA	PERIOD	i	ZERO	9	F	F11
6	C0	9	F8	M	B	8	U	O	J	F9
7	A4	10	F7	N	G	Y	K	7	H	6
8	C3	15	F5	V	S	T	R	5	C	F6
9	A1	16	F3	X		E	4	3	D	F4
10	C5	19	F1	Z	SPACE	Q	2	1	W	F2
11	A2	20						SHIFT-L		SHIFT-R
12	C6	21	~		A		TAB	CAPS LCK		ESC
13	A7	22		ALT-R		ALT-L				
14	C7	23					GUI			
15	A3	24	Fn							

My code doesn't report the Fn – Function (multimedia) keys to the Pi but certain Fn – Function key combinations cause the Teensy to control the video converter card. This eliminates the need for the push button keypad that comes with the video card.

Teensy Circuit Board – The 24 signals from the FPC keyboard connector are wired to the Teensy with a 2.5” x 1.5” circuit board I designed using the Freeware version of Eagle software. All signals are on the top layer with plated thru holes for pins. I found schematic and layout symbols on the web and modified those that weren’t exactly what I needed. I had never used Eagle software but I found it fairly easy to figure out with lots of information online when I got stuck. Do the schematic first and it will make a layout with air wires that guide where you need to put traces. Sparkfun has excellent tutorials for Eagle schematic and layout.

The 24 signals on the keyboard FPC cable have a 1mm pitch. I purchased compatible FPC connectors from Ali express. 24 of the Teensy digital I/O pins are routed to the FPC connector pads and the remaining Teensy I/O are routed to thru hole pads for connecting to other devices (Touchpad, Video card, LEDs, and Pi). I added pads for 5 volt bypass caps in case the Teensy or Touchpad had power problems but found they weren’t needed. The layout is shown below without the solid copper fill so you can easily see the signal traces. This is a revised layout that includes some corrections from the original that I used.

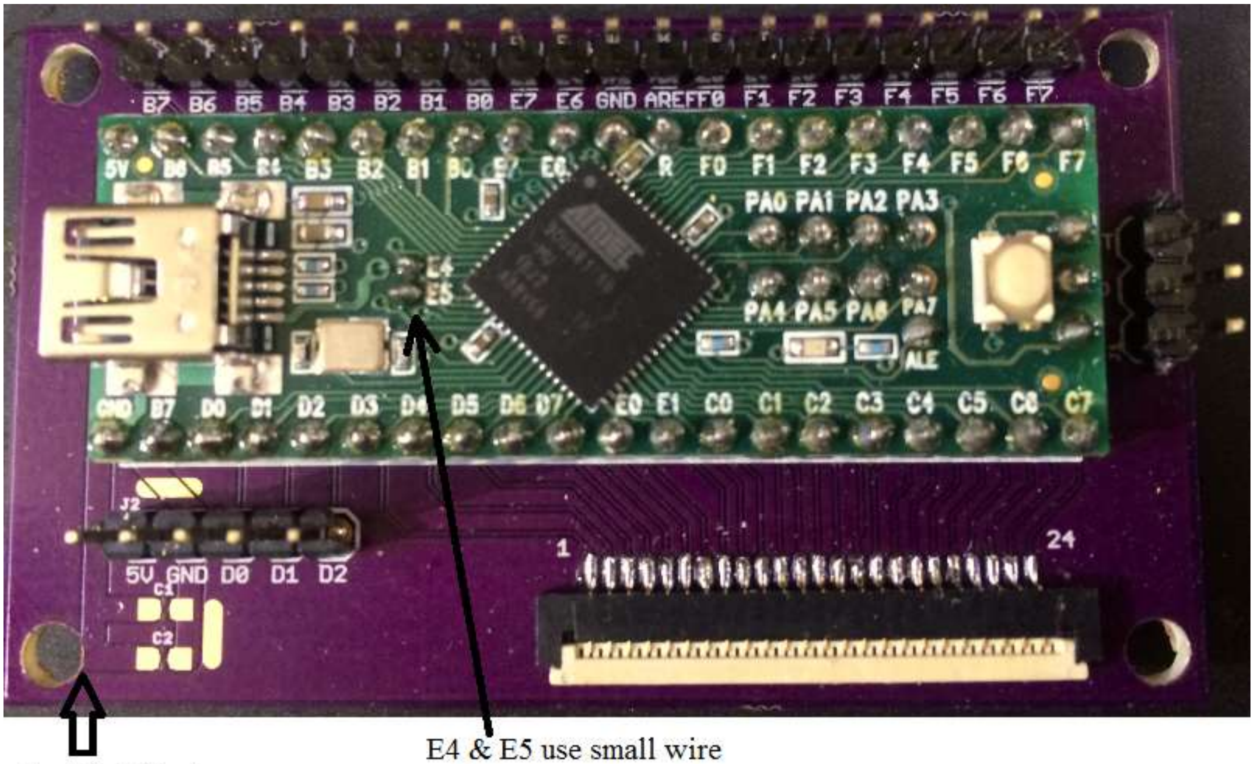


Size: 2506 x 1456 mil

Three boards were fabricated by Oshpark in under 2 weeks. Oshpark accepts the Eagle layout file directly so you don’t have to create Gerber files or even know how to use Eagle to order the boards. I put a drop of super glue on the FPC connector and attached it to the board (just to hold it in place). Then I used my Hakko soldering iron and a T18-C05 tip to solder the 24 connector pins to the board. I definitely needed geezer goggles to see the small solder joints. Flow soldering with the toaster oven would have worked much better but my wife vetoed that plan.

I bought the Teensy without pins so I could solder my own header pins around the edge and in the interior. A standard pin won’t fit in the E4 and E5 locations but a small wire from a 1/8 watt resistor will work. The finished board with Teensy and FPC connector is shown below. I included pins so I could wire a reset switch to the back of the laptop case to initiate the Teensy loader but the reset button is only needed the very first time the loader is used. After that, the loader can initiate everything over USB so I didn’t bother adding a button. This worked fine until I updated the Arduino/Teensyduino software to the latest version, then I had to open up the case and push the button for the first load.

My original layout had mounting holes that were way too small. I had to drill larger holes but ended up with a power to ground short when I installed the lower left mounting screw. I've enlarged the hole size and moved the 5 volt trace to fix these problems. The schematic and revised layout files are posted on Github.



Pwr/Gnd Short

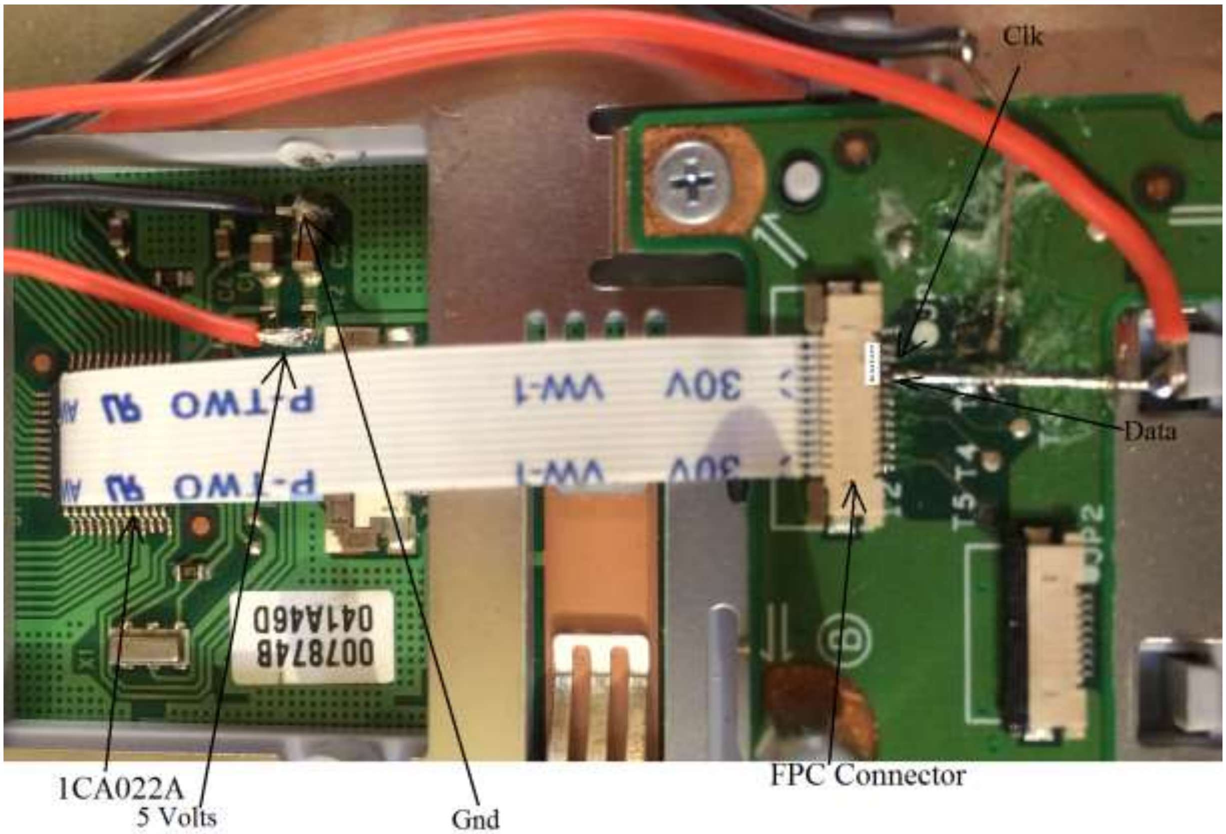
E4 & E5 use small wire

Table of FPC pins to Teensy inputs

FPC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Teensy	D3	D4	D5	E5	D7	E0	E4	E1	C0	A4	C1	A0	C2	A5	C3	A1	C4	A6	C5	A2	C6	A7	C7	A3

I used a Dremel tool to cut the air vent bars on the back of the laptop so that the Pi's network and USB connectors are accessible. The USB cables from the Teensy and from the SSD come out the back of the laptop and loop back to the connectors on the Pi as shown below. I may change to shorter cables with right angle connectors but for now this length makes it very easy to move the cables over to my PC when I want to download new Teensy code or transfer a large file to the SSD. I thought I'd finish the keyboard routine and never do any more Teensy coding but I keep coming up with more things it can do.

Touchpad – The following picture of the Touchpad circuit boards show the connections to Clock, Data, +5 volts, and ground. The +5 volts and ground were connected on the circuit board that contains the 1CA022A touchpad chip. The Clock and Data were connected on the button board. The Clock is on pad T2 and FPC pins 1 & 2, Data is on pad T3 and FPC pins 3 & 4. It was difficult to attach the Clock and Data wires to the small pads. To give the solder joint some strain relief, I tinned a wire with solder and glued it to the board. After the glue dried, I soldered it to the connection point and to the jumper wire. I used super glue but I think JB Weld would work better because it's not so runny.



I started with ps/2 touchpad code from [playground.arduino] and modified it for USB. The touchpad uses the [ps/2 mouse protocol] which sends X and Y movement values as signed 9 bit values. My Teensy code converts the movement values to signed 8 bits and inverts the sign of the Y data to match the polarity of the PJRC [Mouse.move] USB function. The left and right buttons on the touchpad are converted from ps/2 to the PJRC Mouse.set_buttons USB function. Normally a touchpad sends position and button data in stream mode whenever a change is detected and the host runs an interrupt service routine. I needed the touchpad to send position and button data when queried by the Teensy (remote mode). In my code, after completing a keyboard scan, the Teensy requests data from the touchpad and if any movement or buttons are pressed, the information is sent to the Pi over the same USB cable as the keyboard data (known as USB Composite). The touchpad USB data can be disabled by pressing Fn and F12 on the keyboard. It can be turned back on with the same sequence. This allows a USB or Bluetooth mouse to be used instead of the touchpad.

Display - The LCD from the Sony laptop is a Quanta Display Inc. part number QD15TL03. It needs LVDS video signals so the HDMI out of the Pi must be converted by the M.NT68676 board. Several eBay companies sell the M.NT68676 board. Verify with an eBay message prior to purchase that the board is compatible with your LCD model number. The Monitor Control Board Specification gives the interface definition of the M.NT68676 board.

The board contains a 4 Megabit serial flash memory (part number Winbond 25X40CLNIG) that is programmed by the vendor for the LCD parameters. The main chip on the board is marked NOVATEK NT68676UFG but I could find no data sheet online. The board takes in 12 VDC and a 54329E buck regulator outputs 5 volts. There are two 3.3 volt linear regulators, part number 1117B 33 that feed the LCD and the NT68676UFG chip. The NT68676UFG is also fed 1.8 volts from an 1117B 18 regulator. A two watt stereo audio amplifier, part number TDA7496LG, is fed from the HDMI audio and outputs to a 3.5mm jack which I have cabled to the laptop speakers. The volume is pretty low so I'm looking for replacement speakers. Linux commands from a terminal window are used to switch the audio to the speakers or to the headphone jack.

Speakers:

amixer cset numid=3 2

Headphone jack:

amixer cset numid=3 1

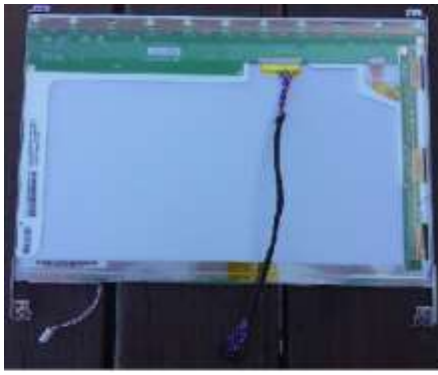
I've made aliases called "speakers" and "headphones" in .bashrc that execute the amixer commands so I don't need to remember them.

Before making any video cable modifications, I connected the M.NT68676 video board to the Pi and confirmed it correctly drove the LCD. The keypad menu items on the screen were in Chinese. Luckily I found an Alex Eames video which shows the key sequence to change the language from Chinese to English. The sequence is Menu, Right, Menu, Menu, then use the Left or Right buttons to highlight English, and then push Menu to select it.

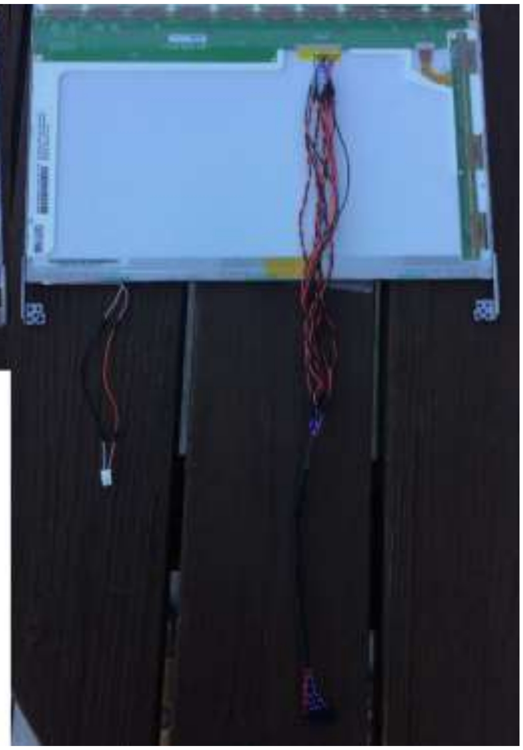
I needed to lengthen the LVDS cable so it would reach the board when mounted in the laptop. I added 10 inches to each wire including the 4 twisted pairs using 28 gauge stranded wire. I also added 4 inches to the 2 wires that feed the backlight as shown in these before – after pictures. The display was noisy and not syncing properly with the longer cable but once I pulled the wires tight and wrapped them with electrical tape, it worked fine.

In hindsight, I should have lengthened the original laptop video cable by about 4 inches instead of the cable that came with the kit. The original cable has a shielded twisted pair clock with grounded conductive tape over all wires and is flat for the section that fits behind the LCD. My only concern with using this cable was the wire gauge is really tiny so I thought it might be fragile and hard to solder.

Before



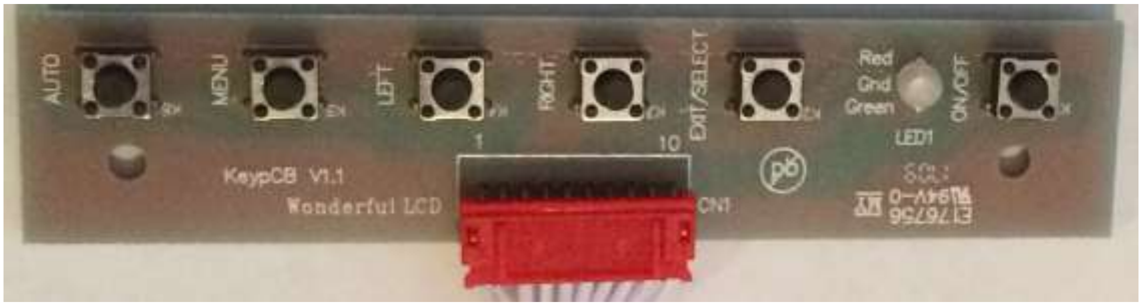
After



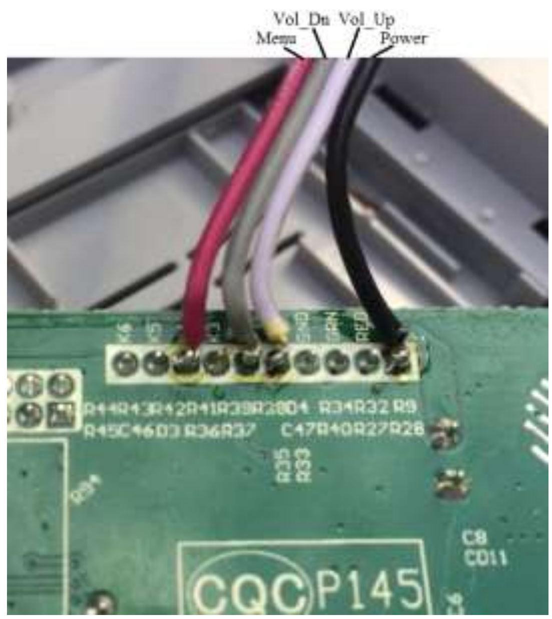
The pinout for the video cable is below:

LCD Connector	Signal Name	Card Connector
1	GND	4
2	3.3V	1
3	3.3V	2
8	LVDS ODD 0-	7
9	LVDS ODD 0+	8
10	GND	5
11	LVDS ODD 1-	9
12	LVDS ODD 1+	10
13	GND	6
14	LVDS ODD 2-	11
15	LVDS ODD 2+	12
16	GND	13
17	LVDS ODD CLK-	15
18	LVDS ODD CLK+	16
19	GND	14

I replaced the key pad switch board shown below with the Teensy and Fn-Function Keys.



The only switches I wired were Left (volume down), Right (volume up), Menu, and On/Off (Power). Auto didn't seem to do anything and Exit/Select wasn't needed because the controller will exit if no keys are pressed for a couple seconds. The switches are pulled up on the video card to 3.3 volts and go to ground when pushed. The Teensy must not drive these signals to 5 volts or it could damage the control chip. The following picture shows where 4 wires were soldered to the keypad connector on the backside of the video card. The connections to the Teensy are given in the table.



Key Function	Board	Teensy
Volume Up	K1	B7
Volume Down	K2	B6
Menu	K4	B5
Power	K0	B4

The Teensy code floats the 4 control signals but sends a low when the Fn & Function keys are pressed per the following table. Menu, Volume Up, and Volume down can be sent low with manual keypresses to navigate thru the video card selections. This works for commands that are seldom changed but would get old fast if I had to do this for Mute or Brightness. For these commands, the Teensy pulses the Menu, Volume Up, and Volume Down signals in a sequence to reach the desired command. I tried speeding up the pulsing sequence but the video controller chip expects human speed or it doesn't work properly.

Key Sequence	**Result**
Fn and F1	Send the Menu signal low
Fn and F2	Mute/Unmute the HDMI Audio [1]
Fn and F3	Send the Volume Down signal low
Fn and F4	Send the Volume Up signal low
Fn and F5	Lower the Display Brightness [2]
Fn and F6	Raise the Display Brightness [3]
Fn and F7	Send the Power signal low

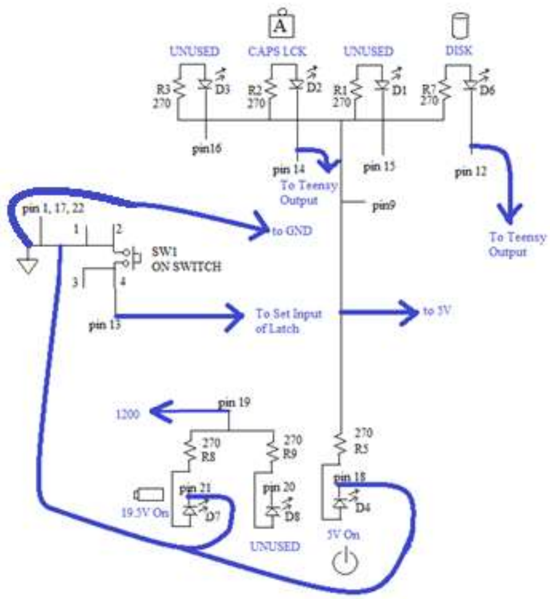
[1] Pulse Menu, Volume Up, Menu, Volume Down, Menu, Volume Down

[2] Pulse Menu, Menu, Menu, and then send Volume Down low until key is released

[3] Pulse Menu, Menu, Menu, and then send Volume Up low until key is released

I probably didn't need to wire the power signal but I wanted the ability to blink the display off and back on to get the operator's attention. This could come in handy if I ever add battery operation and want to let the user know the battery is almost dead. This could also be used if a sensor was added to detect if the laptop lid has been closed.

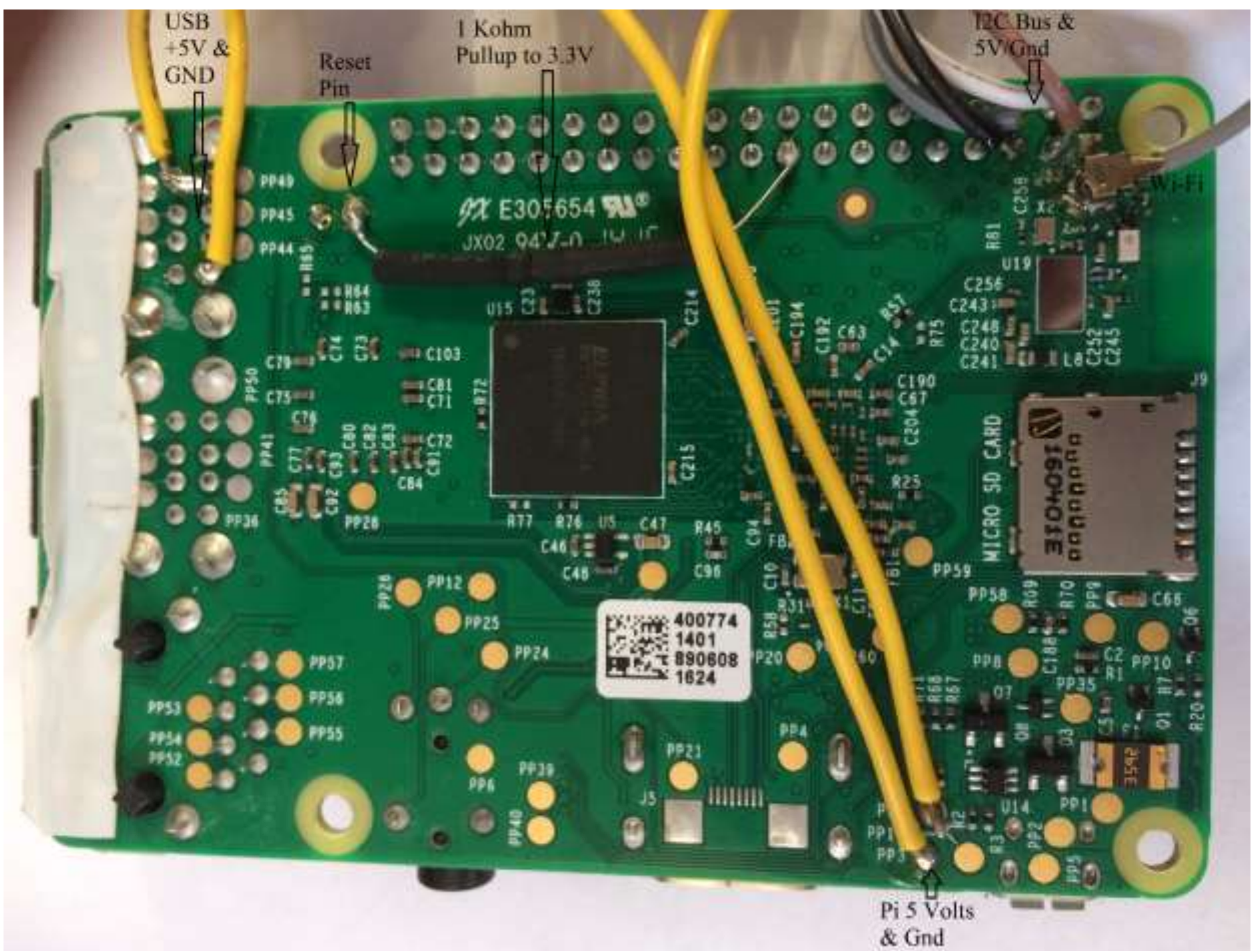
Power and LEDs – The following schematic of the Power Switch & LED Board shows blue connections where I added wires to the board. I chose to solder directly to the surface mount components instead of dealing with the FPC connector. The Battery LED turns on when the 19.5 volts is connected and the Power LED lights when the Pi is powered with 5 volts. The Caps Lock LED is driven by the Teensy and is based on the “keyboard_leds” variable controlled by the Pi. The Disk LED is also driven by the Teensy and is used for code debug. The power switch is connected to the “Set” input of the Latch circuit described later.



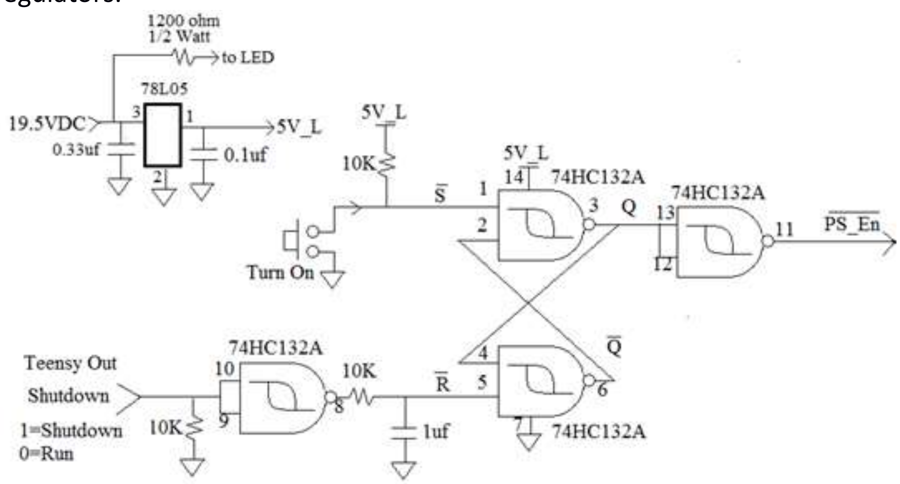
The four LEDs are all turned on in this picture.

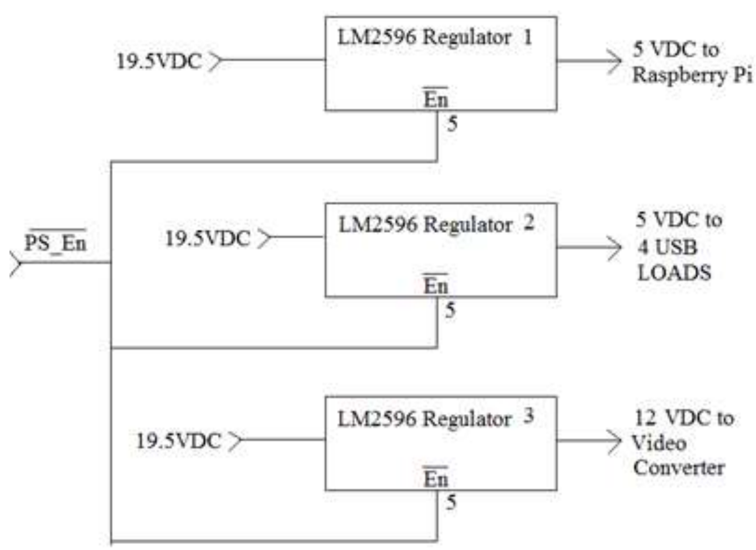
Power Supplies – The laptop is fed 19.5 volts DC from the original Sony AC adapter. 3 regulator boards based on the LM2596 drop the voltage down to 12V for the video card, 5V for the Pi, and 5V for the USB ports. Many versions of this board can be found on Amazon and EBay. I adjusted the potentiometer on each board to give the desired voltage, then connected them to their respective loads and readjusted the voltage. The unmodified LM2596 board is shown below.

The 5 volt power from regulator #1 is soldered to the Pi (back side) at the large pad labeled PP7 near the micro USB connector. For USB power, the AP2253 at location U13 on the Pi (front side) was removed in order to separate the Pi 5 volt bus from the USB 5 volt bus. The 5 volts from regulator #2 is soldered to the power pins of the USB Connector as shown in the picture below. The 12 volt power to the video card from regulator #3 uses a 2.5 x 5.5mm connector. This picture also shows the Wi-Fi connector, the reset pullup, and the 4 I2C wires for connecting the real time clock.

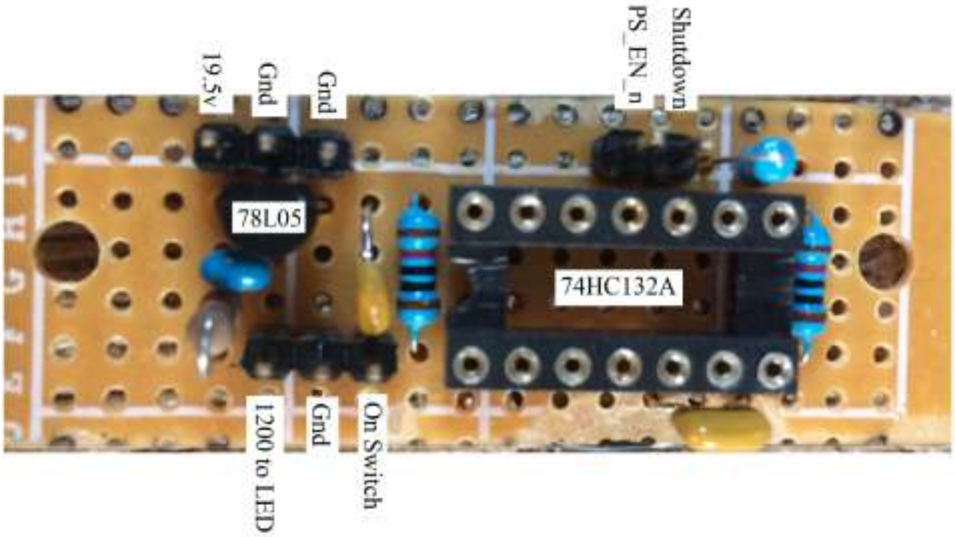


The enable pin of the LM2596 regulator is active low and tied to ground on the circuit board as described at raspberrypi.org forum. I cut the LM2596 enable pin (pin 5) on all 3 boards so I could feed an on/off signal from the push button latch circuit shown below. Plugging in the power from the AC adapter provides power to the 78L05 so it can output 5 volts to the 74HC132A NAND gates, two of which are wired as an SR NAND Latch. The Set and Reset inputs to the Latch are active low. The Reset input to the Latch is fed from a NAND gate tied as an inverter to buffer the Teensy control signal. The Q output of the Latch drives another NAND gate also tied as an inverter to buffer the Latch output. The power to the NAND gates and the Set input to the latch comes up quicker than the Reset input because of the 1uf cap. This causes the latch to be reset and the power supplies disabled when the AC adapter is plugged in. When the Turn-On button is pushed, it sends a low to the Set input of the latch which turns on the power supply enable signal on the regulators. The Latch will continue to enable the power supplies until the Teensy is given a command to shut down, at which time, the Teensy resets the Latch to disable the regulators.

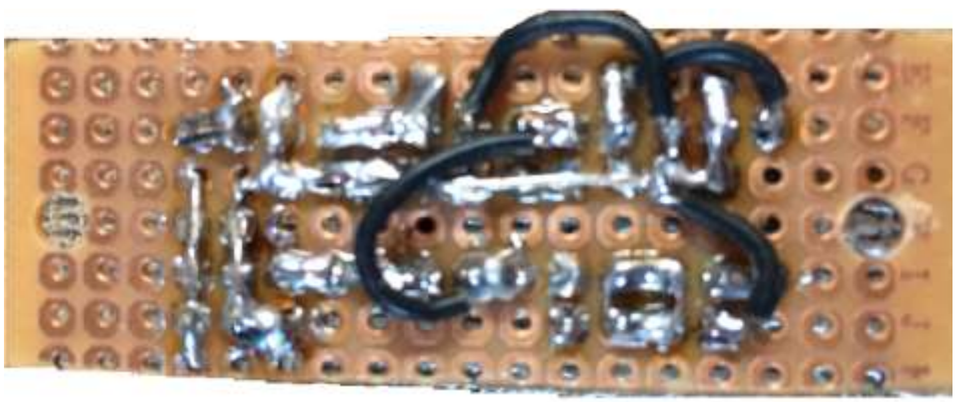




The Teensy will cause a shutdown if control-alt-s is typed on the keyboard or if a Linux shutdown is initiated with a “turnoff” alias described later. The latch circuit board shown below includes a 14 pin socket for the 74HC132A and header pins for easy jumper attachment.



The components are soldered with point to point “ugly” wires on the back side.



When turned off, the three LM2596 regulators, 78L05, 74HC132A, and the Power LED measure 14ma from the 19.5 volt source. When turned on and running a You Tube video over Wi-Fi, the 19.5 volt source measures about 1.06 amps. Running the “Stress” program (described later) increases the current to 1.16 amps.

Occasionally I will lock up the Pi with a bad program but instead of cycling power, I reset the Pi with Control-Alt-r. I connected the Pi's reset pad to one of the Teensy's I/O pins. I also added a 1K pull up on the reset so I can plug and unplug the Teensy USB cable and not cause a glitch that resets the Pi. This picture also shows the USB power switch that was removed at location U13 so that the Pi's 5 volt bus is separated from the USB 5 volt bus.

Solid State Drive – I followed the ETA Prime procedure to boot from an external drive. I had a spare 240GB SSD which is a bit of an overkill but the price was right. Now I no longer need to worry about wearing out the micro SD card or running out of space. The micro SD card has been removed from the Pi which means I don't need to mount the Pi with the SD card connector accessible. The SSD fits in the laptop hard drive compartment (see below) but there was no mounting hardware available. To keep the drive from moving around, I used tightly rolled paper to take up the remaining space.

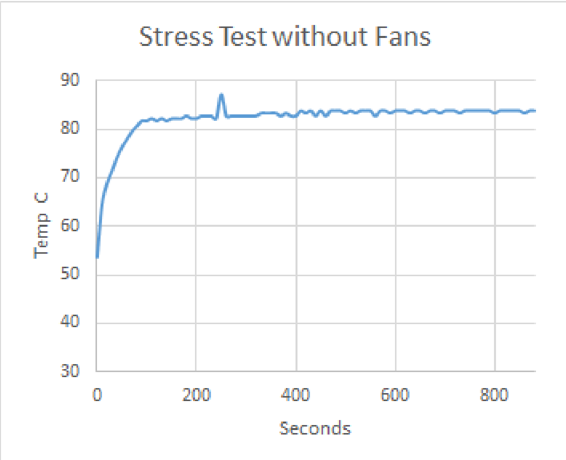
I used a standard USB to SATA cable and routed the cable out the back of the laptop so I can plug it into the Pi.

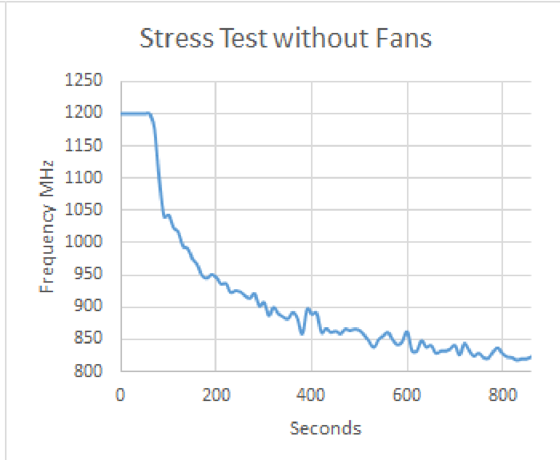
Wi-Fi Antenna – The laptop has a Wi-Fi antenna above the LCD fed by a coax with a U.FL connector. In order to use the laptop's Wi-Fi antenna, I had to add a matching U.FL connector to the Pi and move a zero ohm resistor. This mod was a bit difficult due to the small size of the components. The wardr instructions are very clear and include some magnified pictures. A microscope would really come in handy for this mod. I used the Wavemon program to take before and after signal strength measurements so I could tell if the new antenna was working. The following table shows a definite improvement in the Wi-Fi signal strength.

Location & distance to source	**Signal Level Before Mod**	**Signal Level After Mod**
Desk (next to Wi-Fi router)	-20dBm	-17dBm
Chair (10ft)	-40dBm	-37dBm
Bedroom (15ft)	-59dBm	-50dBm
Dining room (15ft & wall)	-68dBm	-54dBm
Kitchen (20ft & wall)	-69dBm	-62dBm

The laptop has a Bluetooth antenna with coax and a U.FL connector that I'm currently not using. I haven't found anything on the web yet for hooking this up to the Pi.

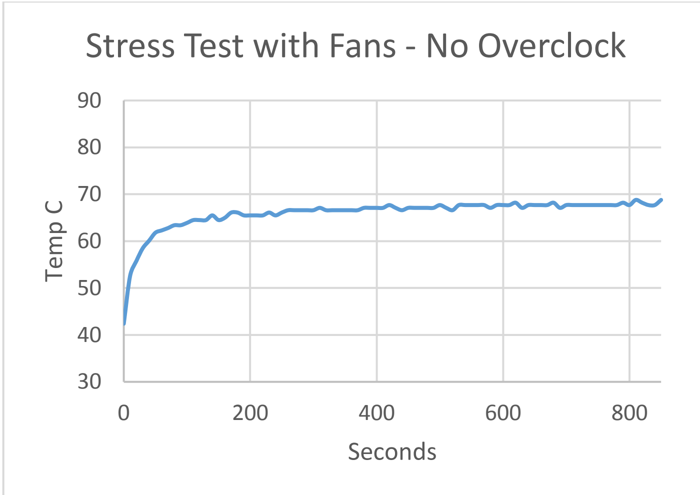
I ran the stress test from Core Electronics on the laptop to measure the temperature of the ARM SoC and its clock speed. Core Electronics says their stress test gives “real life, high-load usage across all of the Pi's resources”. My Pi has a small aluminum heat sink on the ARM SoC chip and I wanted to see what would happen if I didn’t have any fans. The curves below show that the temperature quickly rose to 83°C and the clock speed was heavily throttled to avoid overheating.





I placed a 5 volt fan below the Pi, near the video connector but I found it didn’t help very much because the height of the video connector blocked the airflow and the SoC heat sink fins needed to be rotated 90 degrees. I moved the fan to the right away from the video connector and angled it at the processor. I also added a 12 volt fan at the rear of the case to blow the hot air out the back and draw cool air up thru the floor vents. The fans are sandwiched between the case floor and the metal shield for the keyboard. Double back foam tape keeps them from moving and vibrating (they’re still pretty loud). The fan locations are shown below.

The curve below shows the temperature not exceeding 69°C so the 1200MHz clock frequency never throttled. This is the fan configuration that I have continued to use.



I changed the /boot/config.txt file to add the “usually-safe” overclock settings given on the overclocking wiki. So far I’ve had no issues with these values.

total_mem=1024

arm_freq=1300

gpu_freq=500

core_freq=500

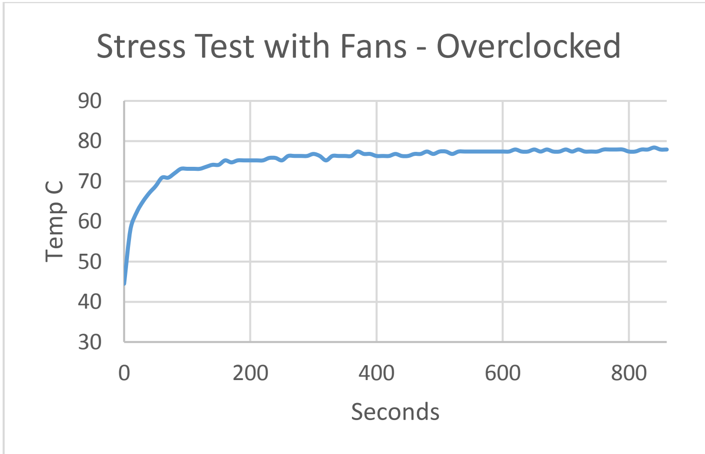
sdram_freq=500

sdram_schmoo=0x02000020

over_voltage=2

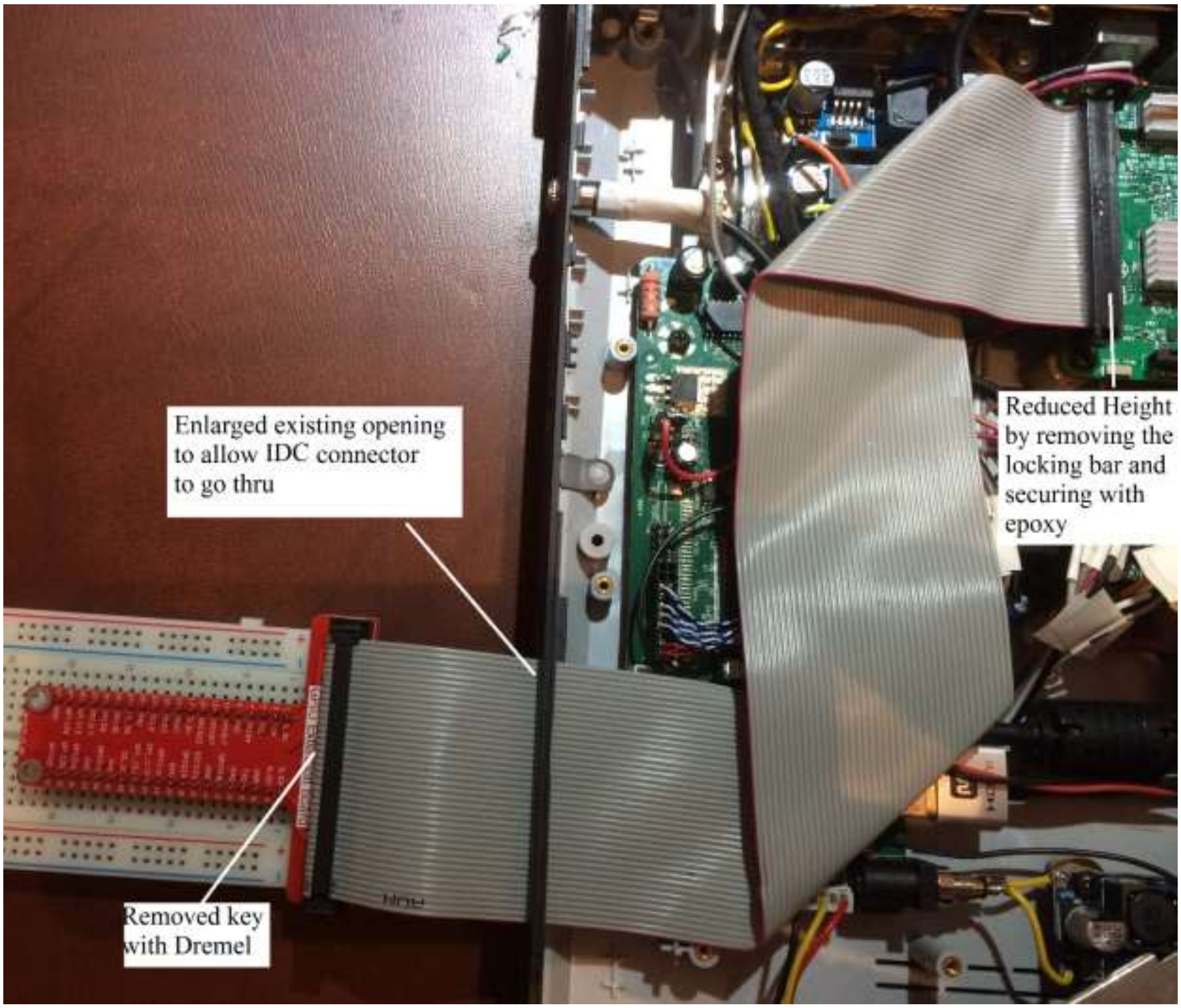
sdram_over_voltage=2

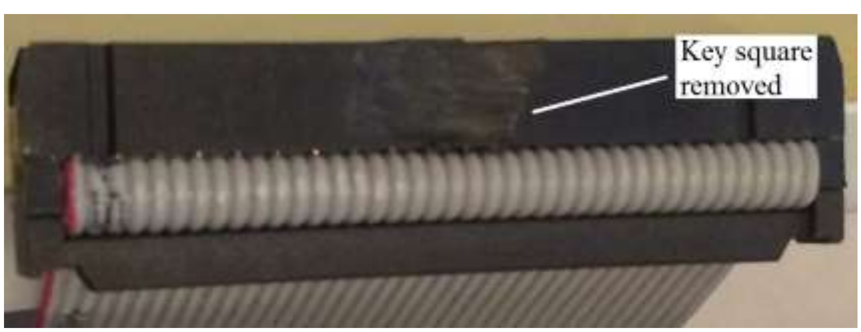
I repeated the stress test with these overclock settings. The 1300 MHz clock speed never throttled and the temperature maxed out at 78°C. I'll keep these settings unless I notice any problems.



For a final cooling test, I ran cpuburn-a53. Core Electronics says it “maxes out your Pi's CPU completely” and “is the most intense CPU stress test you can run and doesn't represent a realistic real-life scenario”. The temperature quickly maxed out at 83°C and the clock speed throttled down to 820MHz. I guess I should be happy that the Pi didn't lock up (or burn up).

A 40 pin ribbon cable is attached to the GPIO connector on the Pi and routed out the side of the laptop for bread-boarding. The IDC connector on the Pi end of the cable was too tall and touched the keyboard so I removed the locking piece from the top of the connector to reduce the height and glued it together with JB Weld. I used my Dremel tool to remove the key square on the breadboard end of the IDC connector so it can plug into the GPIO expansion board. I also used the Dremel to enlarge the existing opening in the laptop case so the IDC connector can pass thru. I'm not sure if this opening was for some kind of gigantic memory card or mini floppy drive.





I installed the Adafruit DS1307 based real time clock (RTC) per their instructions and connected it to the Pi's I2C bus by soldering jumpers per this table. I also connected the Teensy to the I2C bus.

I2C Name	Pi GPIO & Pin Number	RTC Pin Name	Teensy Pin
Clock	GPIO 3 Pin 5	SCL	D0
Data	GPIO 2 Pin 3	SDA	D1
+5 Volts	Pin 4	5V	
Ground	Pin 6	GND	

I cut the small traces between the large pads at the locations shown below in order to disconnect the clock and data pull up resistors. These resistors are pulled to 5 volts which is not compatible with the Pi.

The 1.8K pull up resistors on the Pi are the only pull ups on the I2C bus. The Pi pull ups go to 3.3 volts but the DS1307 is a 5 volt device. The DS1307 datasheet says the minimum voltage for a logic high is 2.2 volts and all testing so far shows the RTC and Teensy work fine with 3.3 volt signals. The RTC is at I2C address 0x68.

The Teensy I2C pins are connected to the Pi along with the RTC. The Teensy is at I2C address 0x08 and if it receives a value of 0x5a, it waits four seconds to let the Pi finish and then resets the power latch to disable the regulators. I created an alias in .bashrc called turnoff that executes the turnoff.exe script. The alias in .bashrc reads:

alias turnoff='/home/pi/turnoff.exe'

The turnoff.exe script sends the shutdown value 0x5a to the Teensy, waits a second to make sure the I2C transmission is complete, then issues a shutdown command to the Pi. The turnoff.exe file is executable and reads:

i2cset –y 1 0x08 0x00 0x5a

sleep 1s

sudo shutdown –h now

The i2cset command is part of the i2c-tools that were loaded per the Adafruit instructions.

Instead of using an alias, I should be able to automatically run an executable file at shutdown but so far I have been unsuccessful. Per several websites - When a Linux shutdown command is given, all executable files in /lib/systemd/system-shutdown are run and one argument is passed to them; either “halt”, “poweroff”, “reboot” or “kexec”.

I created the following executable file that checks for “poweroff” and placed it in the system-shutdown folder.

```
If [ “$1” == “poweroff” ]; then
  i2cset –y 1 0x08 0x00 0x5a
fi
```

Unfortunately this does not work. I even tried commenting out the if and fi lines but still no luck. If you have any suggestions, let me know.

The Teensy code is watching the I2C bus for the following commands:

5a = shutdown (as described above)

b7 = reset the Pi and the Teensy

10 = turn on the “Disk” LED

11 = turn off the “Disk” LED

The Pi can read the Teensy code version and date with a i2cdump command in a terminal window.

i2cdump –y 1 0x08 i

This results in 256 bytes of ASCII text with the version & date shown 8 times. The i2cdump command converts the ASCII to text on the right hand side of each line.

About the Author – I was an engineer at Boeing for 33 years, designing circuit boards, multi-chip modules, and ASICs for various systems. I built lots of Heathkits and Ham radios when I was a kid but didn't do much electronics at home during my Boeing career. Now that I'm retired, I'm getting back into it and discovering how great the "Maker" community is. "Thank you" to all the people that helped me with your forum posts, website articles, and You Tube videos. I started writing this document to keep track of the numerous sources of information and decided I should share it in case others want to do a similar project. I'm sure I've made mistakes so feel free to send me an email (thedalles77@gmail.com) if you have any comments, improvements or questions. I'll update this document and the Teensy code as I learn more.

Cheers

Appendix A – Parts List

Purchased Items:

Raspberry Pi 3 model B from Amazon for $35

Five FPC connectors from Ali express for $4.51 + shipping

Three boards from Oshpark for $18 (includes shipping)

Teensy ++2.0 from PJRC for $24 + shipping

M.NT68676 board from ebay for $25 + shipping

Short HDMI cable from Amazon for $8.13

Four LM2596 regulator boards from Amazon for $10

Latch components from Digikey for $15 + shipping

USB to SATA Cable from Amazon for $8

Two U.FL connectors from Amazon for $5.50

RTC & battery from Adafruit for $8.50 + shipping

Two 40 pin ribbon cables with IDC Connectors from Amazon for $10

5 volt fan from Amazon for $9

Two 12 volt fans from Amazon for $13

Total is over $225 with shipping

Parts from the junk drawer:

Donor Laptop with AC Power Supply

Solid State Drive

USB Cable – Teensy to Pi

Various gauge wire, shrink tubing, solder, mounting screws, wood dowels, and electrical tape

Header pins for Teensy and RTC

Jumpers to attach to header pins

Perf board for Latch circuit

JB Weld Epoxy

Appendix B – Notes on Battery Operation

My Sony Laptop came with a 4 cell Li-ion battery that wouldn't hold a charge. There are many web sites that sell a replacement battery but I'm not ready to buy one if I can't figure out how to safely use it. The Sony circuit board that the battery plugs into uses a [Max 1873] IC that takes the 19.5 volts from the wall charger and reduces it to the battery voltage. The top and back side of the board is shown below.

The Max 1873 controls a PFET which feeds an inductor and diode to make a Buck regulator. The Maxim datasheet has a typical application schematic that looks similar to how the Sony board is wired but there are extra components and I can't find a schematic online. The 19.5 volts is switched by a PFET as soon as it comes on the board but I don't know what controls this FET. There is a PFET connected to the battery that I believe acts as a transfer switch to tie the battery to the load but I don't know what controls this FET. I spent hours with an ohm meter trying to figure out how the multi-layer board is wired but in the end I gave up. If I do proceed with an 1873 based battery charger board, I'll need a transfer switch circuit so that the laptop doesn't momentarily loose power when the 19.5 volt charger is unplugged. A separate circuit will need to shut down the laptop if the battery voltage is too low. Perhaps the A to D converter in the Teensy can monitor the battery voltage and issue a shutdown. Determining the shutdown voltage is another subject. One web site claims shutdown should be at 3.0 V/cell and another claims 3.3 V/cell to give a longer battery life.

I think I'll enjoy my Pi laptop for now and wait to see if I really need to add battery operation. Besides adding more complexity to the laptop, the Lithium-Ion battery is a fire hazard that I'm not ready to deal with (I remember the Boeing 787 battery problems). I am able to use my laptop in the car thanks to a 12 volt to 19.5 volt adapter that I had in my junk drawer from an old Dell laptop.

Appendix C – Alternative Power Connection

I tested another method of powering the Pi and Teensy that involved feeding the video card directly from a 12 volt 3 amp AC adapter and then powering the Pi and USB from the 5 volt buck regulator on the video card. The 5 volts measured 650ma to 950ma feeding into the Pi with USB loads of an SSD and a Teensy. I surfed the web, played YouTube video's and played an mp4 from the SSD with no issues

Moving the amp meter to the 12 volts feeding into the video converter card (which is powering the entire system) shows 1.2 to 1.55 amps at 12 volts, (1.55 x 12 = 18.6 watts).

The 12 volts feeding the backlight measures 450ma. To find the total load on the 5 volt buck regulator, the backlight power (12 x .45 = 5.4 watts) is subtracted from the 18.6 watt input power.

18.6W total input power - 5.4W to the backlight = 13.2W into the 5 volt buck regulator

If you assume the buck regulator is 90% efficient, then 13.2W in but only (13.2 x .9) = 11.9W out. The load current on the buck regulator is approximately 11.9W/5V=2.38 amps. The regulator is rated for 3 amps so the buck regulator on the video card seems to have enough power for the Pi, SSD, and Teensy but not much extra. After making these measurements, I learned about the "stress" test program which adds about 300ma to the 5 volt current. This puts the total 5 volt current up near 2.7 amps.

I chose to not use this method of powering the system because I wanted enough current for powering projects on the GPIO bus, plugging in one or two Passport external USB drives and still have plenty to spare. Another factor is that starting with 12 volts instead of 19.5 volts would not allow for battery operation in the future.

Revision History

November 5, 2017 – Original release
November 10, 2017 – Added description of i2cdump to read version & date of teensy code