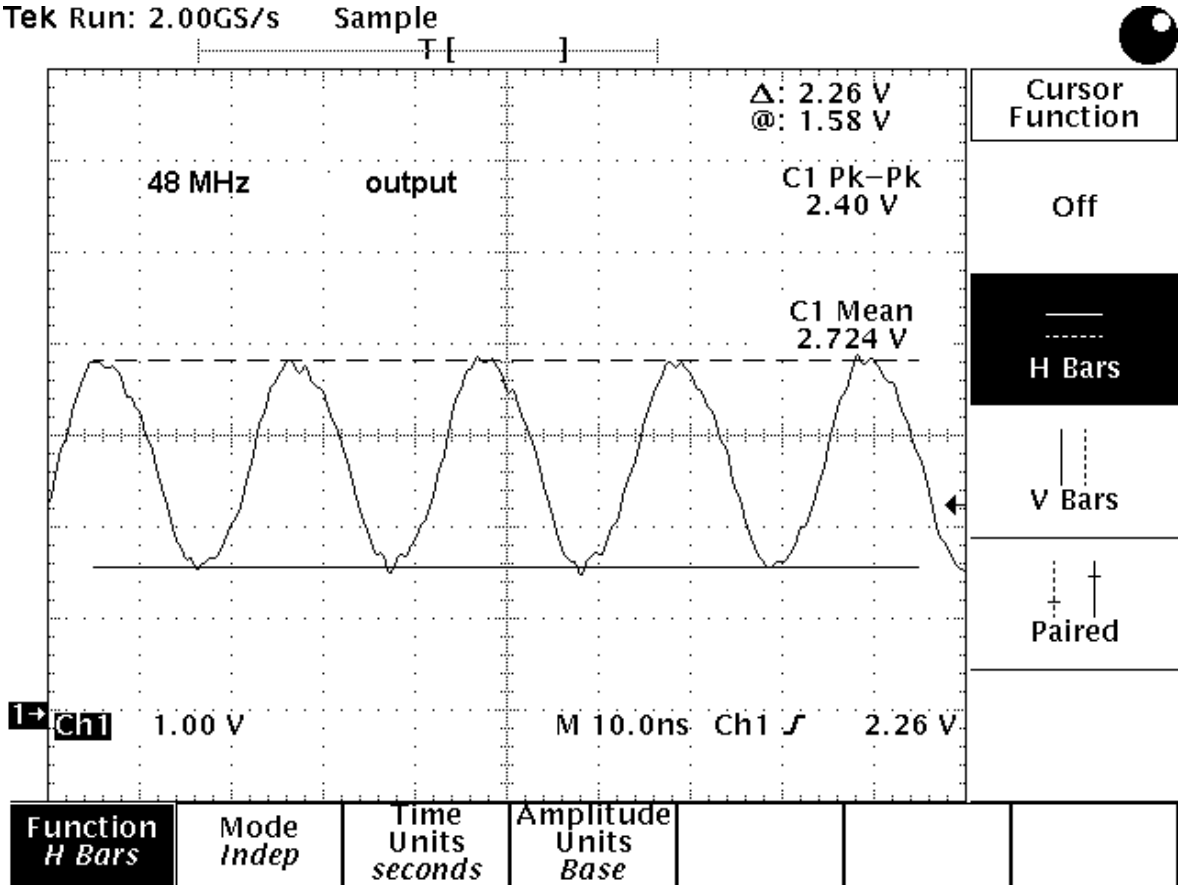
Debug Information for FT8U232/245 devices

This information is provided to help debug a design using the FT8U232 / FT8U245 devices.

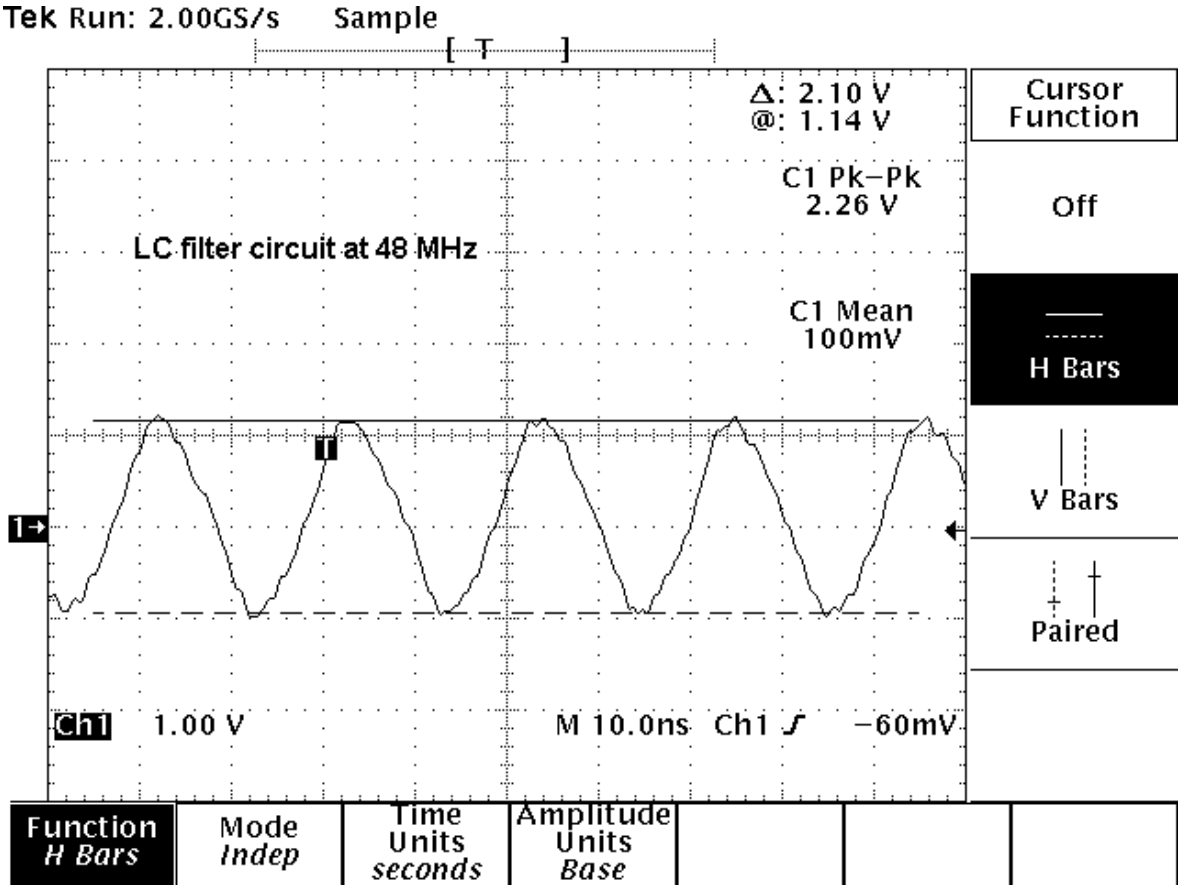
1.(A) Clock circuit (48 MHz crytsal)

Here is what the clock output (pin 28) should look like when oscillating normally. In this mode the pin EECS (pin 32) should be pulled low via a 10K resistor.



As seen here it has a mean of around 2.7 volts and a Peak to Peak voltage of around 2.4 volts.

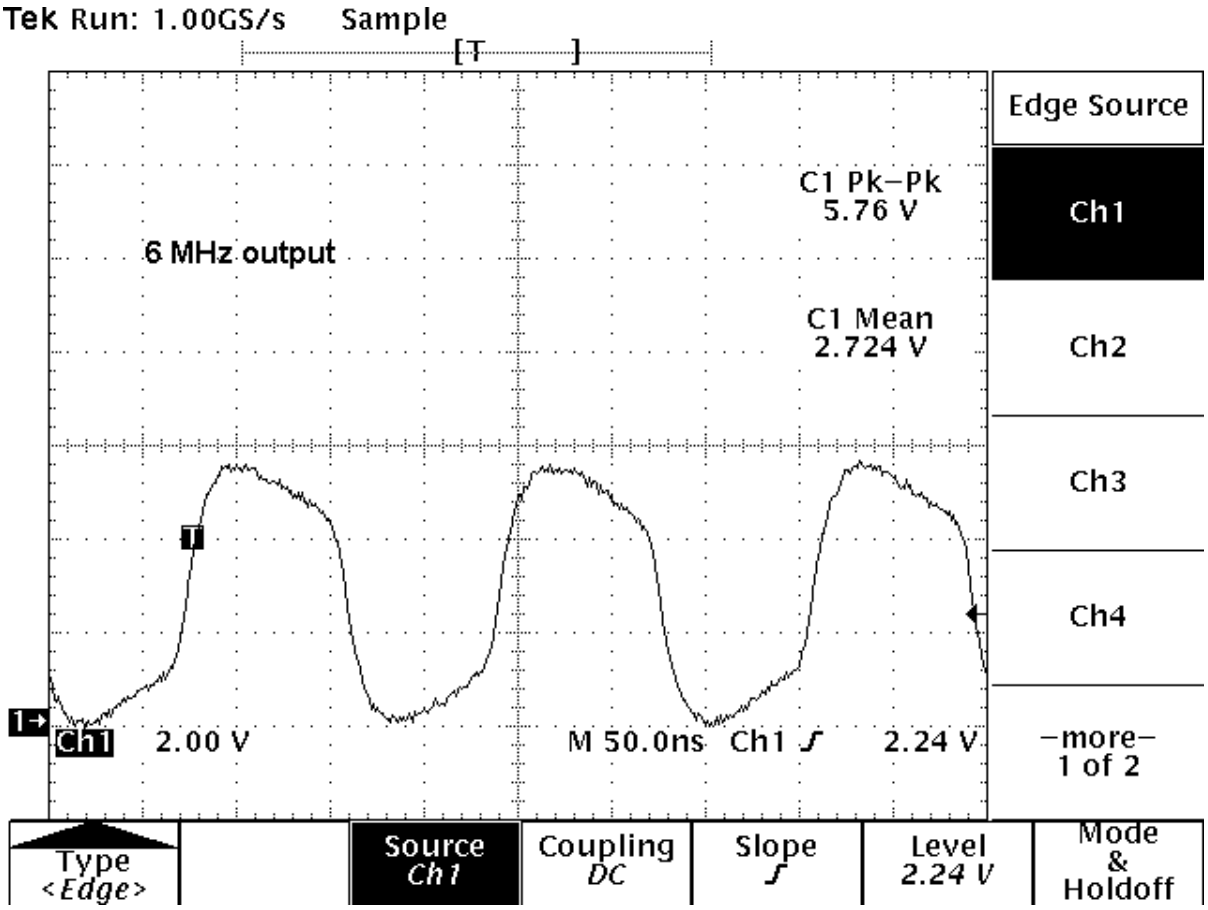
The 1.0uH inductor and 8.2pF capacitor in parallel are used as a tuned filter. The filter is necessary to avoid the crystal running at its base frequency of 16MHz. The waveform at the end of the inductor connected to the 10nF cap should look like this :



As seen here it has a mean of close to 0V (100mV shown) and a Peak to Peak voltage of around 2.3 volts. This is because it is capacitively coupled to pin 28 via the 10nF cap.

1.(B) Clock circuit (6 MHz crystal)

Here is what the clock output (pin 28) should look like when oscillating normally. In this mode the pin EECS (pin 32) should be pulled high via a 100K resistor.



Reasons for not oscillating :

1. Lack of voltage - Ensure USB cable is plugged in and 5volts is seen at the chip.

2. Bad Communication / Enumeration - The chip will stop the oscillator if the USB bus puts it into suspend. The system does this by stopping the Start Of Frame (SOF) packets which are normally every1 millisecond. To check that the oscillator circuit is OK, the RESET# pin (pin 4) can be held to GND (0V). This will stop the chip going into suspend so that the circuit can be looked at with an oscilloscope.

Reasons for oscillating but not working:

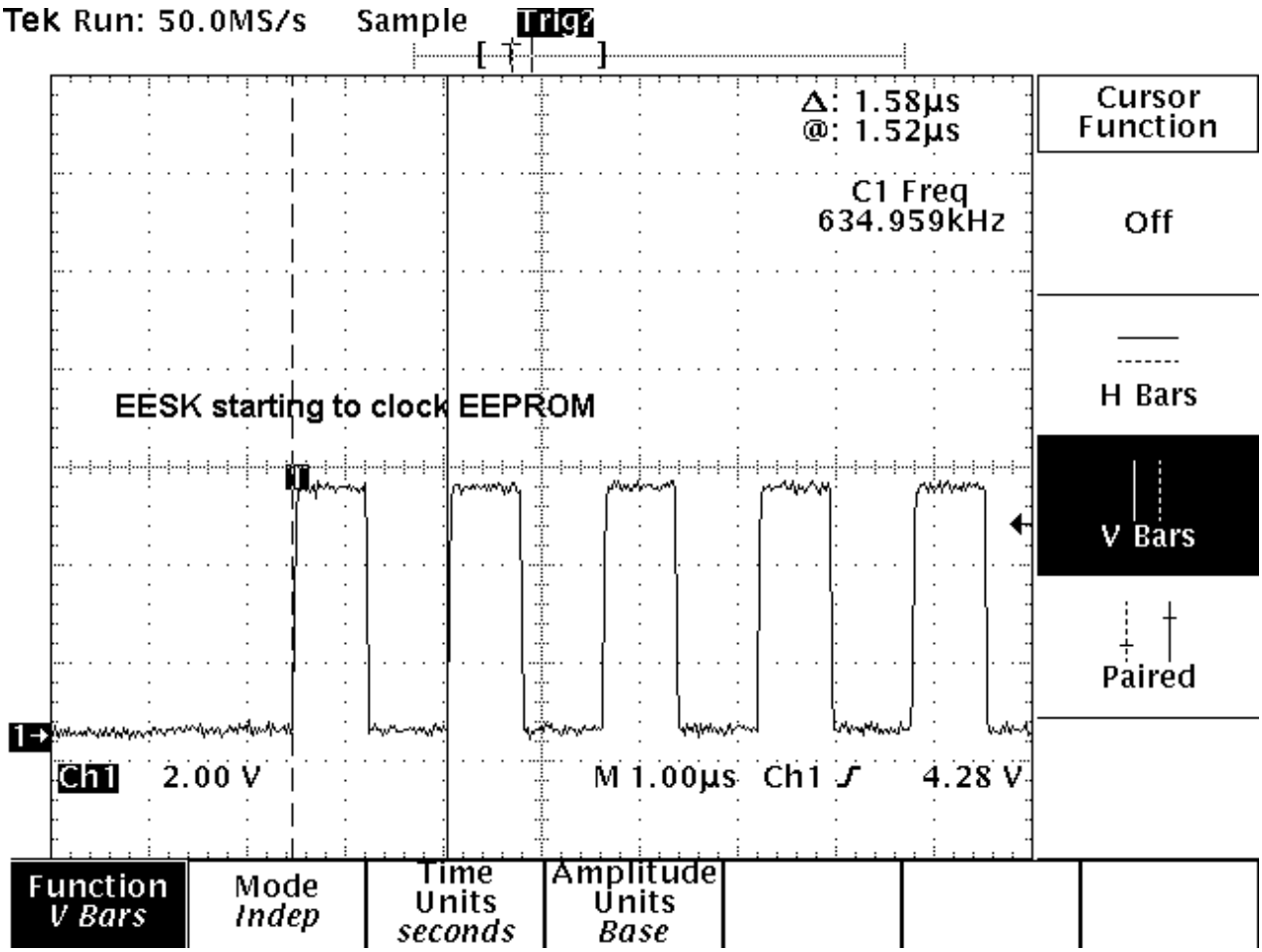
1. RCCLK pin 31 needs to be pulled high to start the chip. It is used when the chip is put in suspend. When suspended, the chip discharges the cap on RCCLK and stops the oscillator. When it starts, the RC time constant is used to delay allowing the oscillator to clock the logic inside the chip, until the oscillator is stable. If it does not go high then the chip will not start. In addition if it goes high too quickly then it may crash coming out of suspend. The time constant should be around 5 ms. It has to be greater than the reset time for correct 6MHz operation. NOTE : some systems put the device into suspend even before it has been enumerated by USB.

2. RESET# pin 4 stuck low or taking too long. The RESET# pin should start low and then rise to take reset off. The time constant should be around 10 millisecs. A good indication that reset has finished is to look at the EESK pin (pin 1). This will clock when the chip starts to check if the EEROM is valid.

3. Chip responds to first SETUP packet but always NACKs the response. - This is generally because there is a fault in the EEROM interface. Usually the pullup on EEDATA (pin 2) is missing or EEDATA is stuck at GND.

4. A PLL bypass set to wrong value for speed :- at 6 MHZ pin EECS (pin 32) should be pulled high, at 48 MHz pin EECS should be pulled low. To check for the correct speed, the EESK line can be checked for the clock period it uses to interrogate the EEPROM when the device comes out of reset. It should have a period of just less than 1.6 microSeconds.

Checking Internal Chip frequency

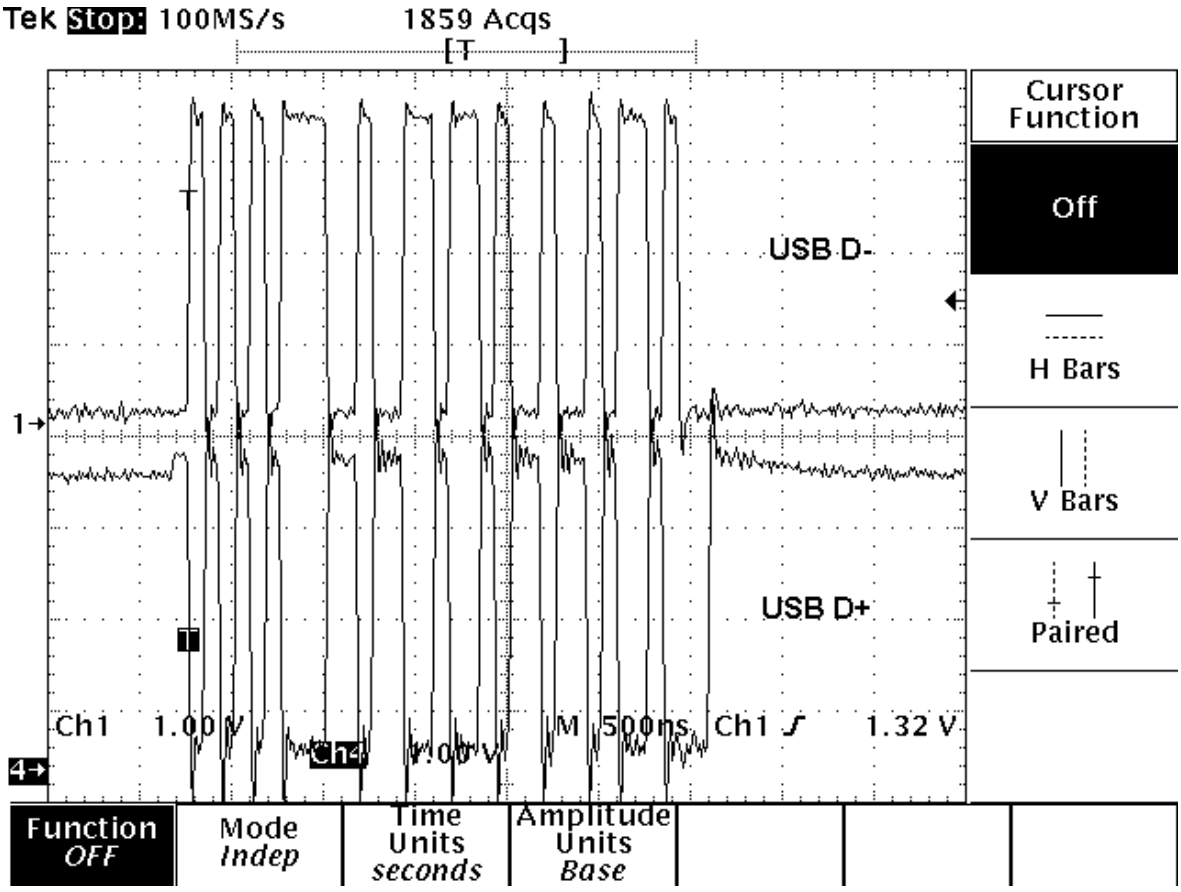


A combination of faults has been seen where an FT8U232 device appeared to work properly, but the baud rate selection was out by a factor of 8. This came about by the wrong frequency mode being selected by EECS, and the D+ and D- pins swapped around. The board was fitted with a 6MHz crystal and set to bypass the 8x APLL. This made it run internally at 6 instead of 48 MHz. The D+ and D- lines were swapped at the connector which caused the host PC's D- line to be pulled up instead of the D+ line. This made the host PC think that a low speed device was present. The 6 MHz frequency is the 4x over-sampling frequency for the internal DPLL that would be used for a normal Low Speed device. Hence everything appeared to be correct except that we ran 8x slower than we should of done. Either fault on its own would have caused the device not to work. When debugging, a good sanity check is to look for the above waveform on the EESK pin and check the period.

2. USB signals

The two signals, D+ and D-, should not have any ferrite beads or inductors fitted in series, because this will make the bus ring and oscillate. They should only have the series resistors in their path.

SOF packet as seen at the USB connector. The top signal is D- and the bottom signal is D+.



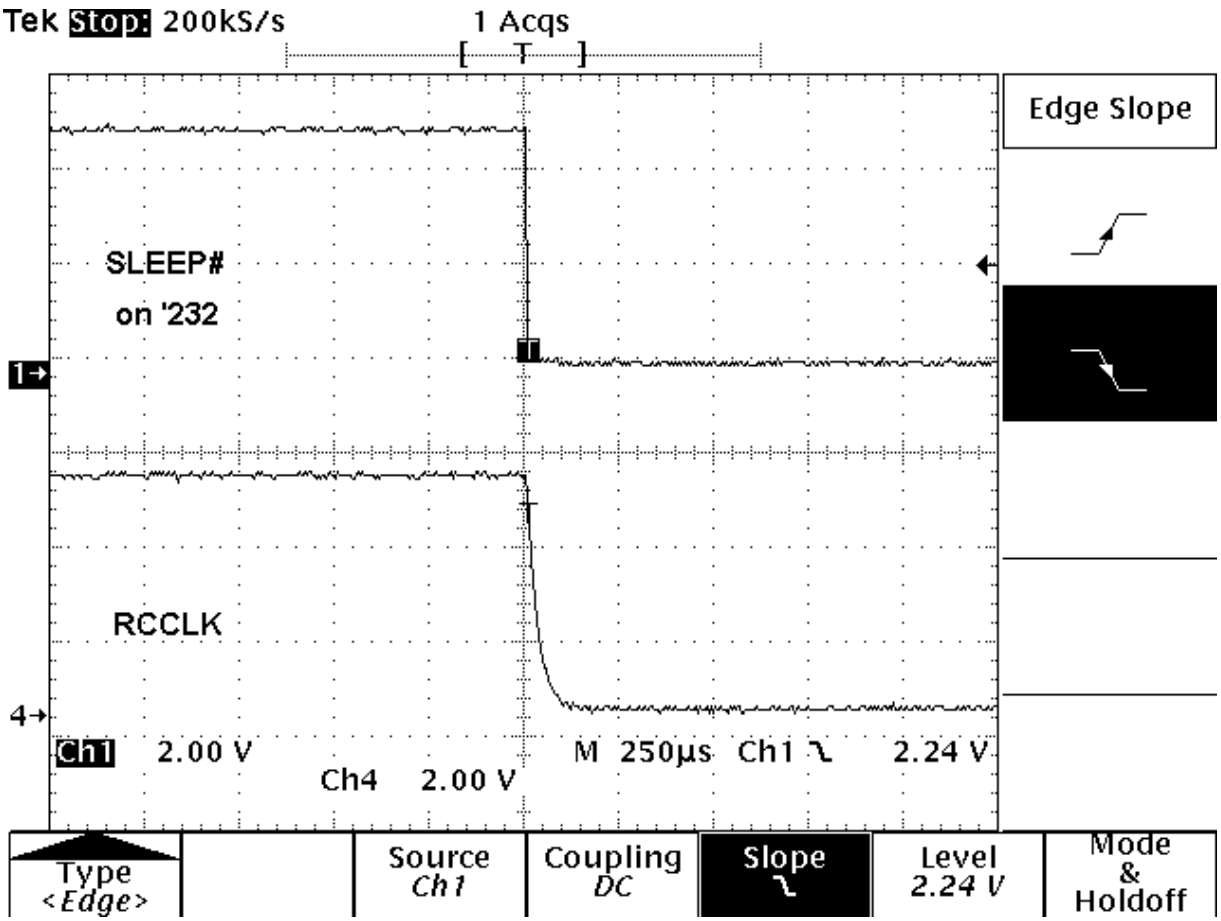
3. Reset circuit

It has been seen, on some more modern computer desktops system, that the system will supply a voltage of around 1.0 volts on USB even when the power is supposed to be off. Because of this, a simple R/C circuit connected directly to the RESET# pin (pin 4) will not work as the Cap is charged to 1 volts already. The use of a 74LS' logic device or the use of a chip such as a Dallas DS1810 will get around this. If the DS1810 is used then a transistor has to be fitted on the D+ line to only pull the line high after reset is over. The reason for this is that the DS1810 takes about 200millisecs before releasing RESET#. If we pull up the D+ line immediately that we are powered, the system will try to talk to us before reset is over. It will get no response and mark us as not working.

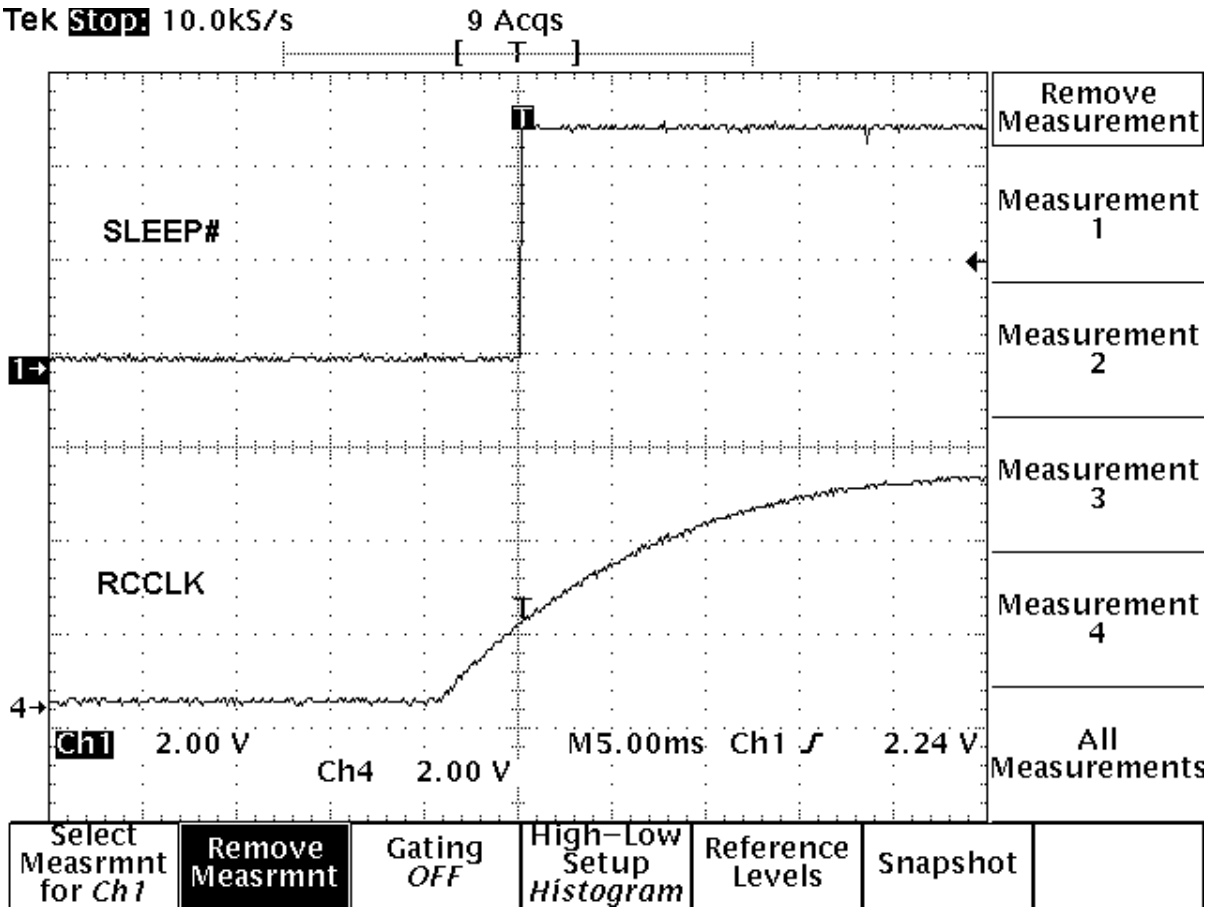
4. Suspend and Resume

The device will go to sleep if USB traffic stops for more than 3 milliSeconds and the USB bus is idle. For the FT8U232 device, the SLEEP# pin will go low to indicate the device is suspended.

When either the FT8U232 or '245 go to sleep, the RCCLK pin is driven and held low as seen below :

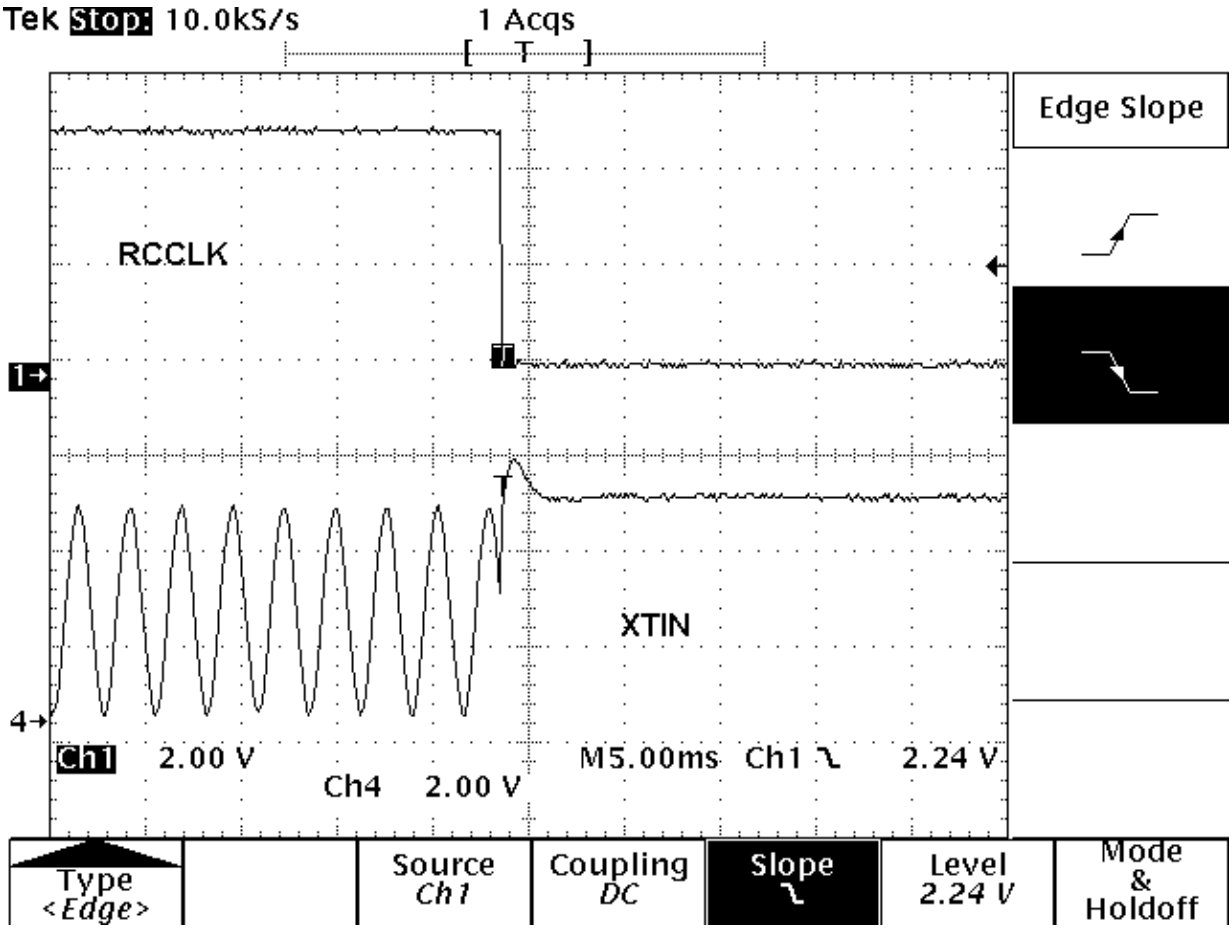


When the device wakes up to an external event such as USB resume or reset, it will release the RCCLK pin and use it to wait until the oscillator is stable.

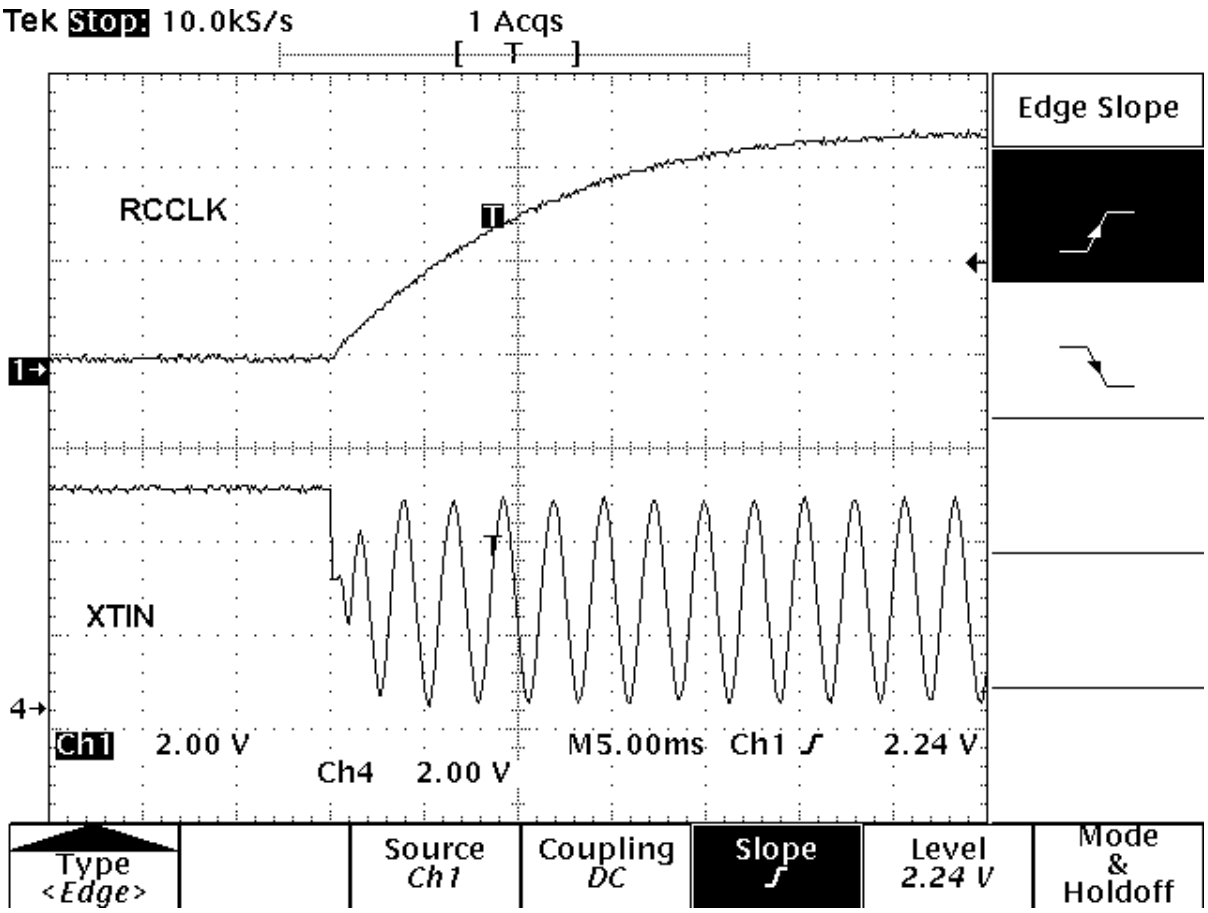


These next two diagrams show the relationship between RCCLK and the crystal oscillator. The frequency of the clock signal appears low because the digital oscilloscope is sampling too low to accurately reproduce it. It is a reasonable assumption that if the wave looks good at this sample rate then it will be good at its real frequency due to the oscilloscopes under sampling (i.e. if it appears to be oscillating, then it is).

This one shows it going into suspend :



This one shows it waking up out of suspend :



5. Signals unique to FT8U232

SLEEP# - This goes low when the device is in USB suspend.

RXLED# - This is pulsed low for a maximum of 1 millsecond when a character is Received from RS232. It has a recovery time of approximately 1 millisecond before it can be pulsed low again.

TXLED# - This is pulsed low for a maximum of 1 millsecond when a character is Transmitted to RS232. It has a recovery time of approximately 1 millisecond before it can be pulsed low again.

PWRCTL - This is used to show the present power source for a GET_STATUS command on USB. The device will use the values from the eeprom for the data returned in a CONFIG_DESCRIPTOR for Bus powered / Self powered / remote wakeup. If PWRCTL is low then a GET_STATUS command will see the device as bus powered. If PWRCTL is high then a GET_STATUS command will see the device as self powered. This is useful for a system where the device can be Self or Bus powered. The Config descriptor should say that is was self powered and the actual source can be read by the GET_STATUS command. If the PWRCTL pin is connected to the external power supply with a 10k pull down, then the present power source will be seen with the GET_STATUS command.

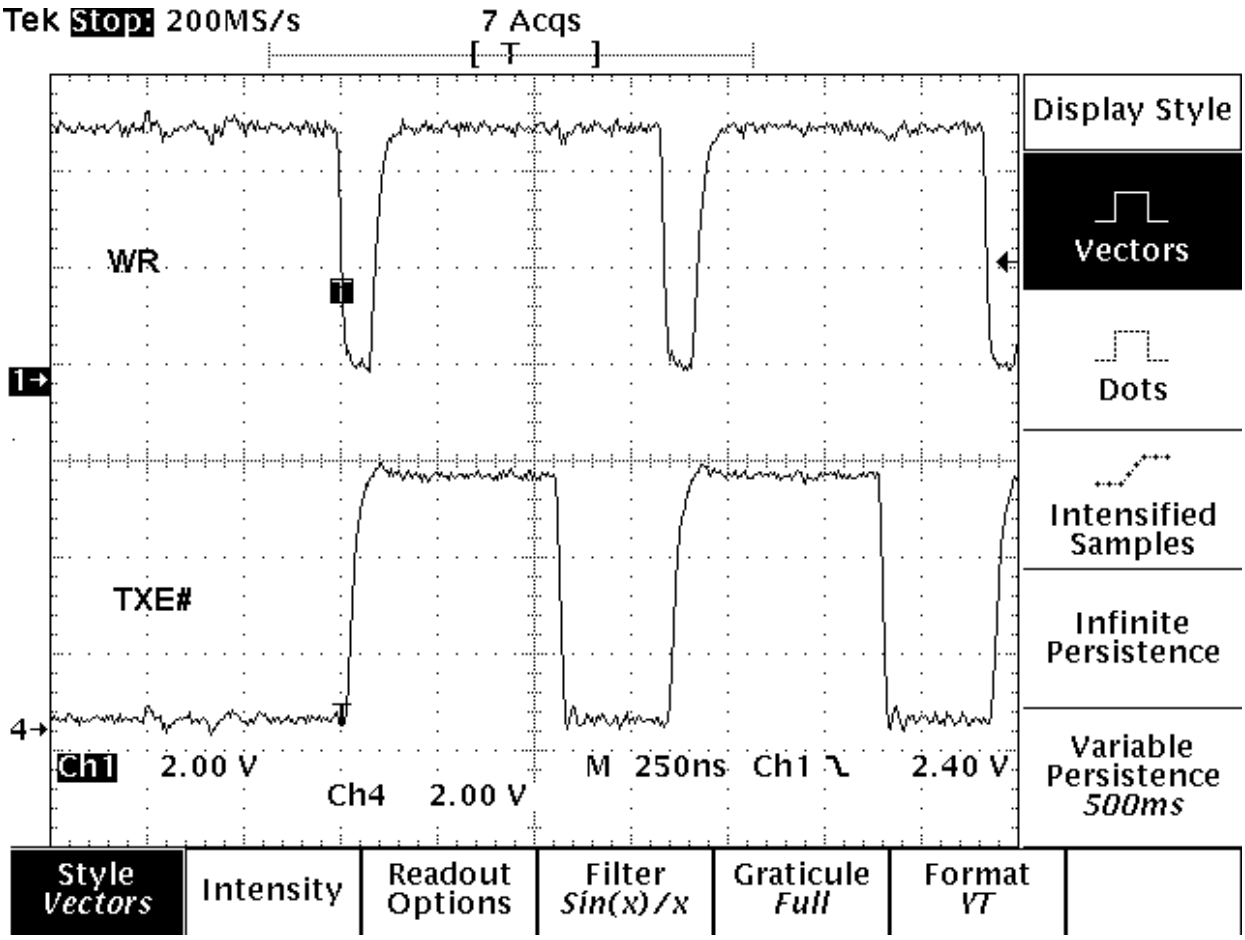
USBEN - This goes high when the device has been configured using a SET_CONFIGURATION command. It is useful in a system where there is a choice of RS232 source. In a modem, for example, there could be two connectors. One would be USB the other normal RS232. This gives the user a choice of which port to connect to. The RS232 lines could be buffered at the 5 volt side using a '244 device. The USBEN signal can then be used to disable the '244 when USB is connected.

TXDEN - This goes high when the device transmits a character. It is used for systems where multiple devices can be driving a cable. Its' purpose is to control the output enable of a level converter. It is turned off at the same time as the last STOP bit is sent. If 2 stop bits are not being used, then a small delay can be added using logic or an RC network to ensure the TXDEN drives the single STOP bit.

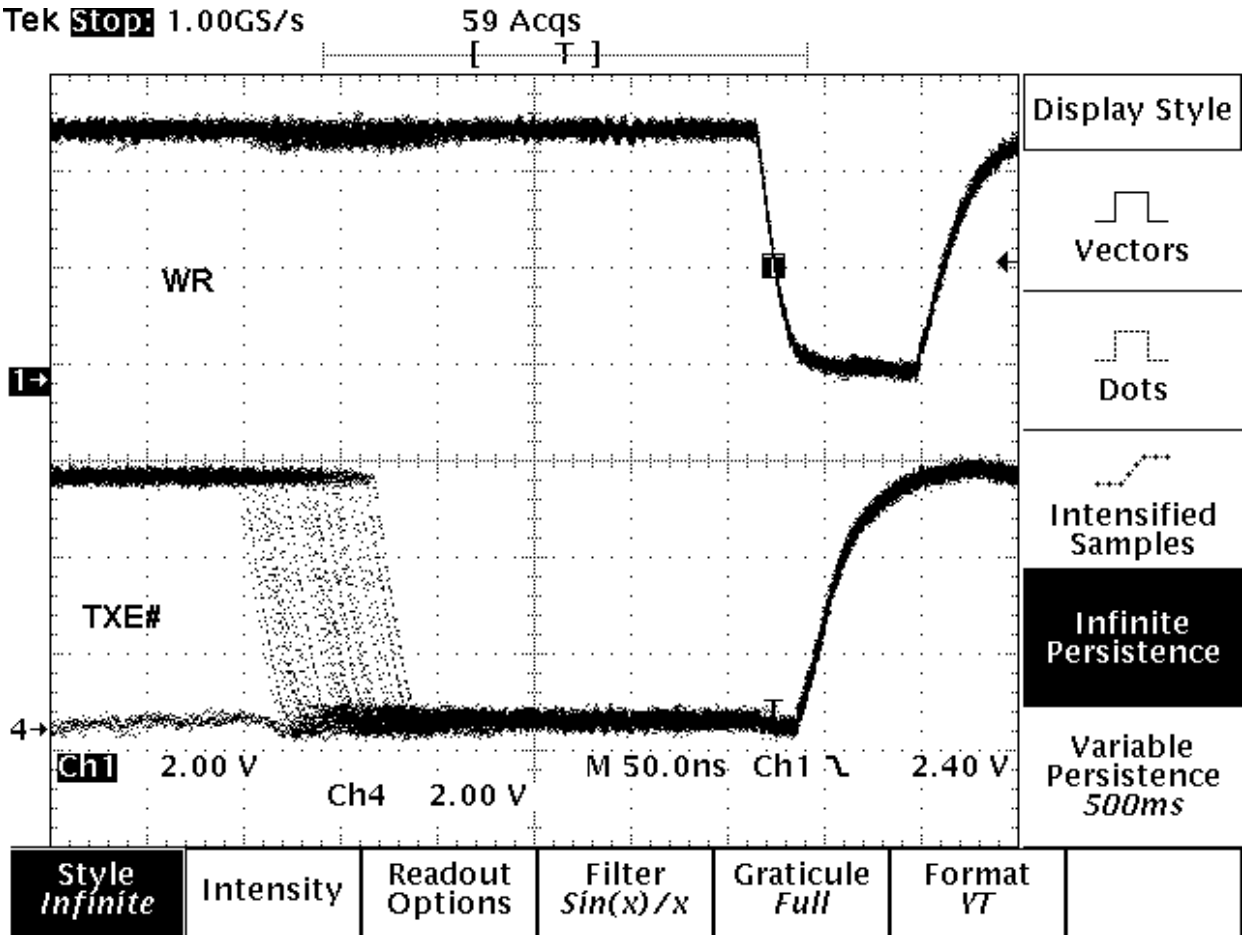
6. Signals Unique to FT8U245

TXE# - Transmitter Empty #
WR - Write to Buffer

This shows a typical burst of data to the device. Data is asynchronously written into an internal holding register in the FT8U245 on the negative edge of WR. The high and low time of WR are not critical – it is the negative edge that is important. On data being strobed into the device on the negative edge of WR, TXE# will always go high in response. If the transmit FIFO is not full, TXE# will go low after a few 12MHz clocks (internal synchronisation) , otherwise it will stay high until the FIFO is partially emptied by a USB read of the transmit FIFO endpoint. TXE# must be low before data can be strobed into the device and the MCU or other logic attached to the FT8U245AM must monitor TXE# to ensure correct operation.

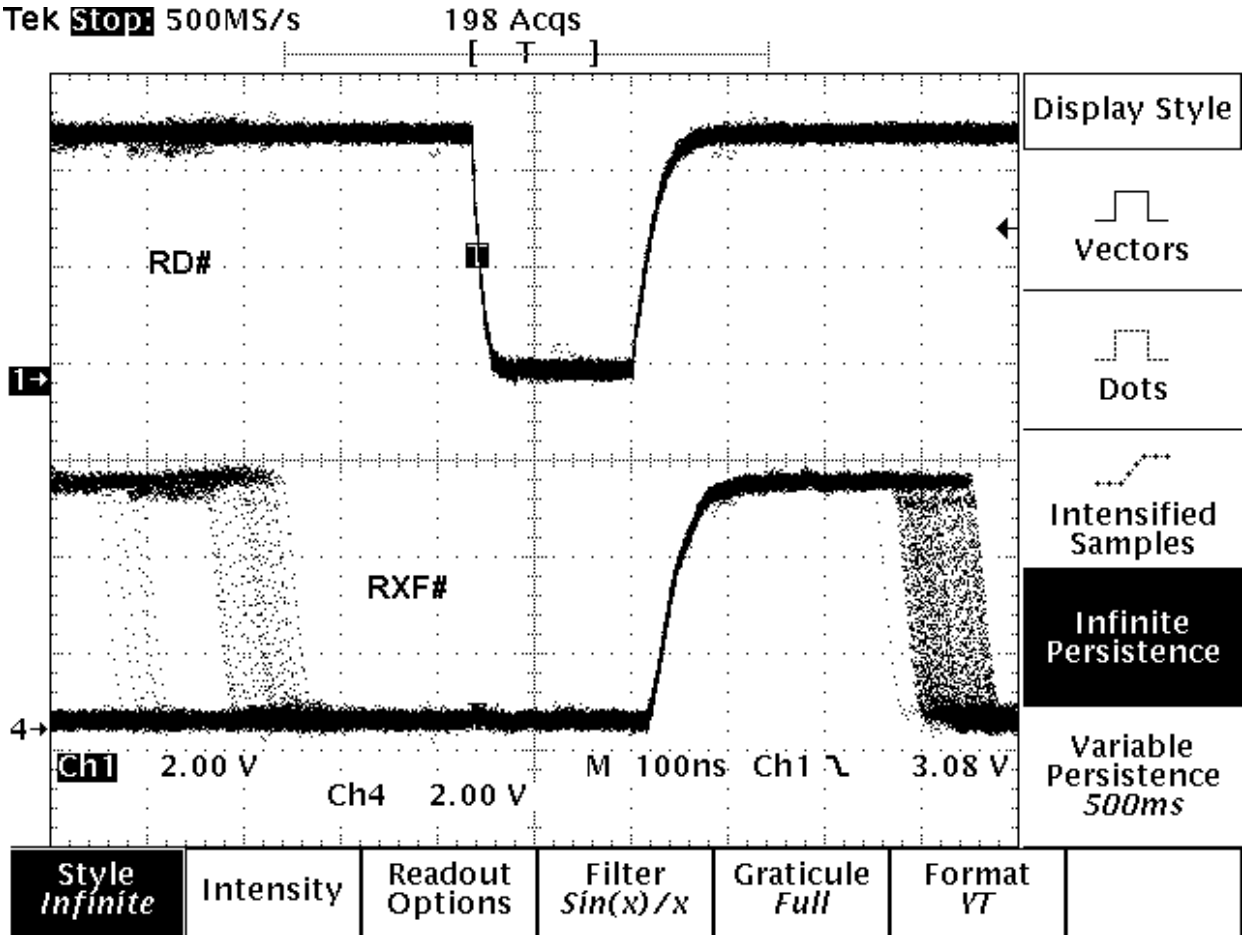


As can be clearly seen from the following trace, the TXE# signal becomes inactive (high) on the falling edge of WR. This example is using a single 12 MHz clock period for the WR inactive period, which gives a a negative pulse of approx. 80 nS.

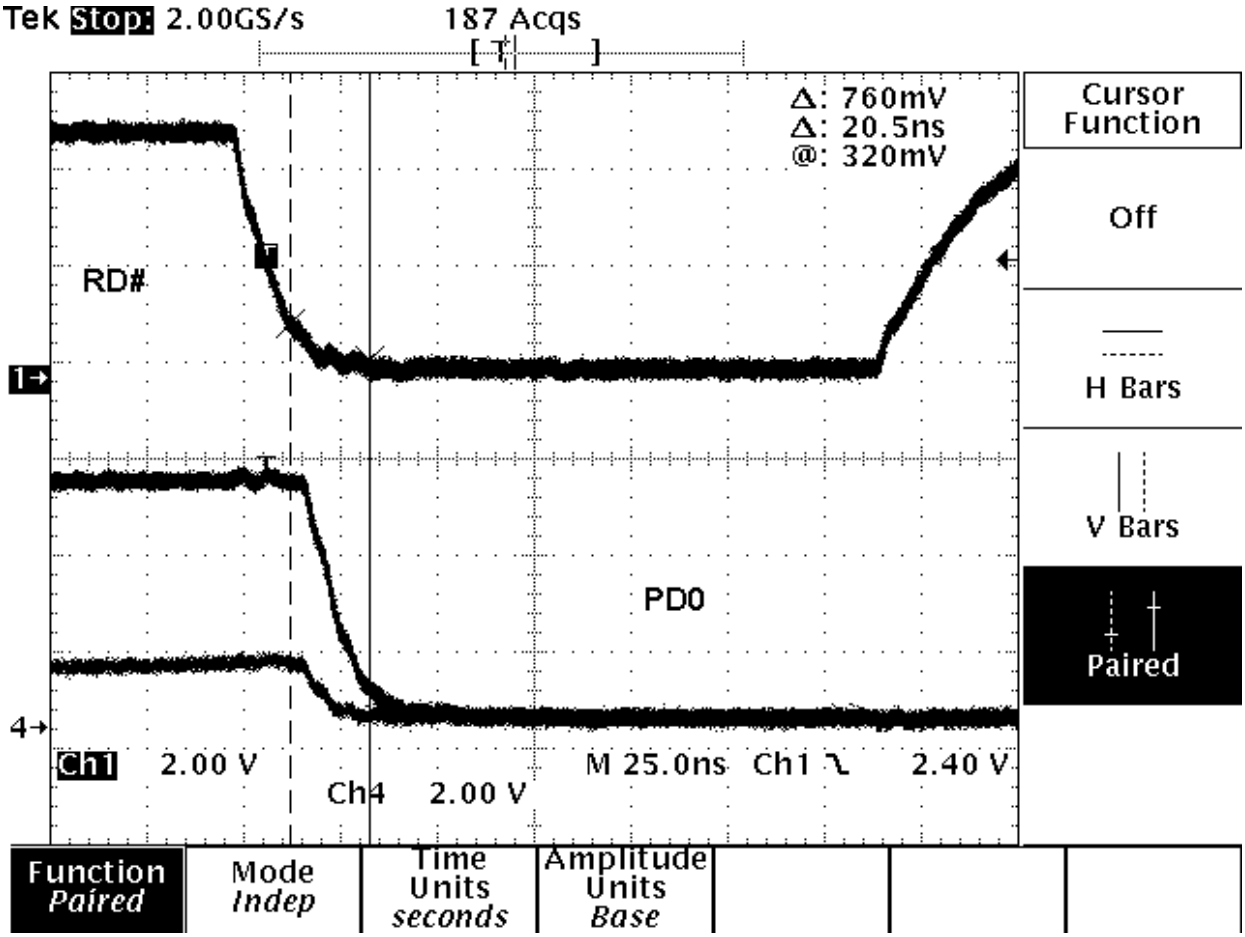


RXF# - Receiver full #
RD# - Read from buffer

This shows data being read from the device. RD# should be held inactive (high) when not reading data from the device. When data is available to be read, the device will drive RXF# inactive (low). To read the data, take RD# low. The device will then drive the data onto the D0..7 data bus. After the data has been read by the external MCU or logic, RD# must be returned to the inactive (high) state. On the rising edge of RD#, the FT8U245 will always drive RXF# inactive (high). If there is more data to be read, RXF# will go low again after a few internal 12Mhz clocks otherwise it will stay high until more data is sent to the device by a USB write to the receive FIFO endpoint.



The following trace shows the data being driven by the device when RD# goes low. The RD# signal enables the devices data buffer drivers. The data will be valid within a maximum of 50nS from the falling edge of RD#. This example is using two 12 MHz clock periods for the read pulse,.which gives a read time of approx 160 nS.



EEREQ# - EEPROM request (input)
EEGNT# - EEPROM grant (output)

These signals are provided for a special configuration of the FT8U245AM which allows two devices to be connected back to back in a master / slave configuration to make a USB data transfer cable. For normal operation these signals are not used and will not be supported in future generations of FT8U245.

EEREQ# must be pulled high or tied to VCC.
EEGNT# should be left open circuit.