Emulation Tech Note 1
Getting Connected with OMAP

Document Revision 0.02

May 20, 2004

1. Introduction

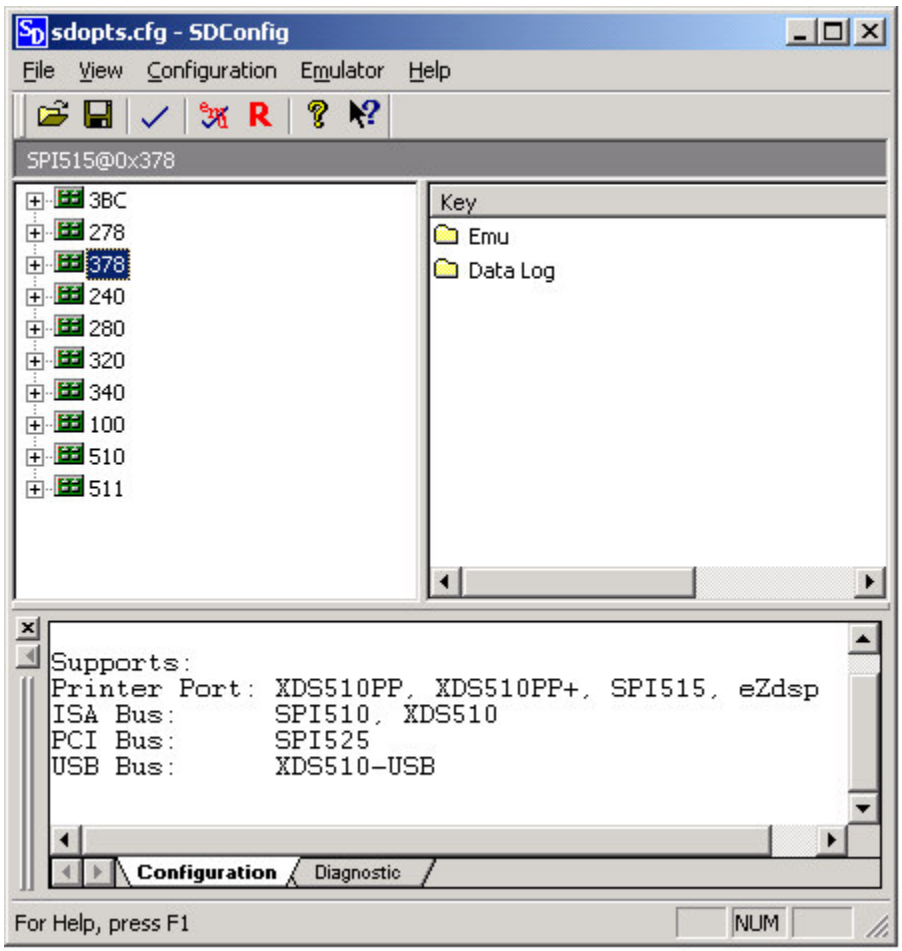
The TI OMAP16XX is based on an ARM926 core device, which implements a non-standard method of JTAG clocking. OMAP16XX devices require a JTAG clock input and return a JTAG clock output known as RTCK. RTCK is then fed back to the emulator on the TCK_RET signal or pin 9 of the standard TI 14 pin header. The emulator and OMAP16XX execute from the RTCK signal and not the TCK input signal. For details on ARM926 JTAG port visit the ARM web site. For details specific on OMAP devices visit the TI web site. This document is centered on the OMAP16xx devices. However the RTCK issues show up on ALL TI OMAP devices, which use the ARM 926 core. At present that includes OMAP5905, OMAP5912, OMAP16xx, OMAP17xx, DM320, and DM730.

The key points related to TCK and RTCK with respect to TI OMAP16xx devices can be summarized as follows:

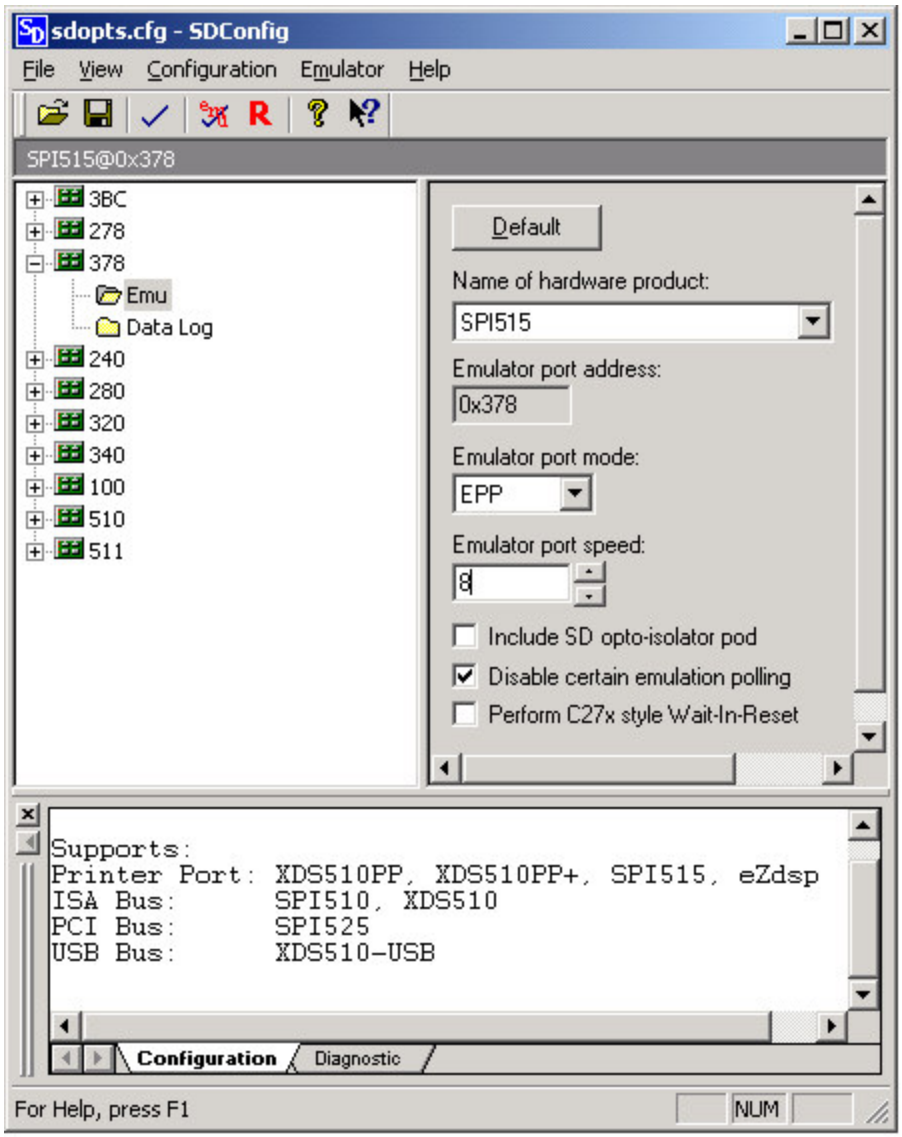
- RTCK is a derived from TCK input and synchronized to the ARM core clock. The net result is that on power up RTCK will be running at a low frequency. Generally RTCK will be 2 MHz with 10 MHz TCK input and 12 MHz OMAP16xx input system clock. This assumes the OMAP16xx PLL is in bypass.
- When an OMAP16xx device is reset then RTCK will turn off as the ARM core clock is turned off during reset. This prevents starting the emulator and performing any scan validation while the OMAP16xx device is held in reset.
- An OMAP16xx watchdog reset, or idle mode and other sources may also cause a device reset, which result in RTCK turning off if only for a short period.
- The OMAP16xx device reset is asynchronous and may cause glitches on RTCK during turn off. This can have adverse affects on an attached emulation controller, which may require an emulator reset to recover.

2. Setting up SD Emulators via SdConfig

Using an SD emulator with the OMAP16xx device generally requires additional emulator setup through SdConfig. This is due to fact that the default RTCK from an OMAP16xx device is generally in the low mega-hertz range. There are two possible ways to set up your emulator for slow RTCK. To get to the emulator settings start SdConfig then select the port address that corresponds to your emulator then the "Emu" folder.



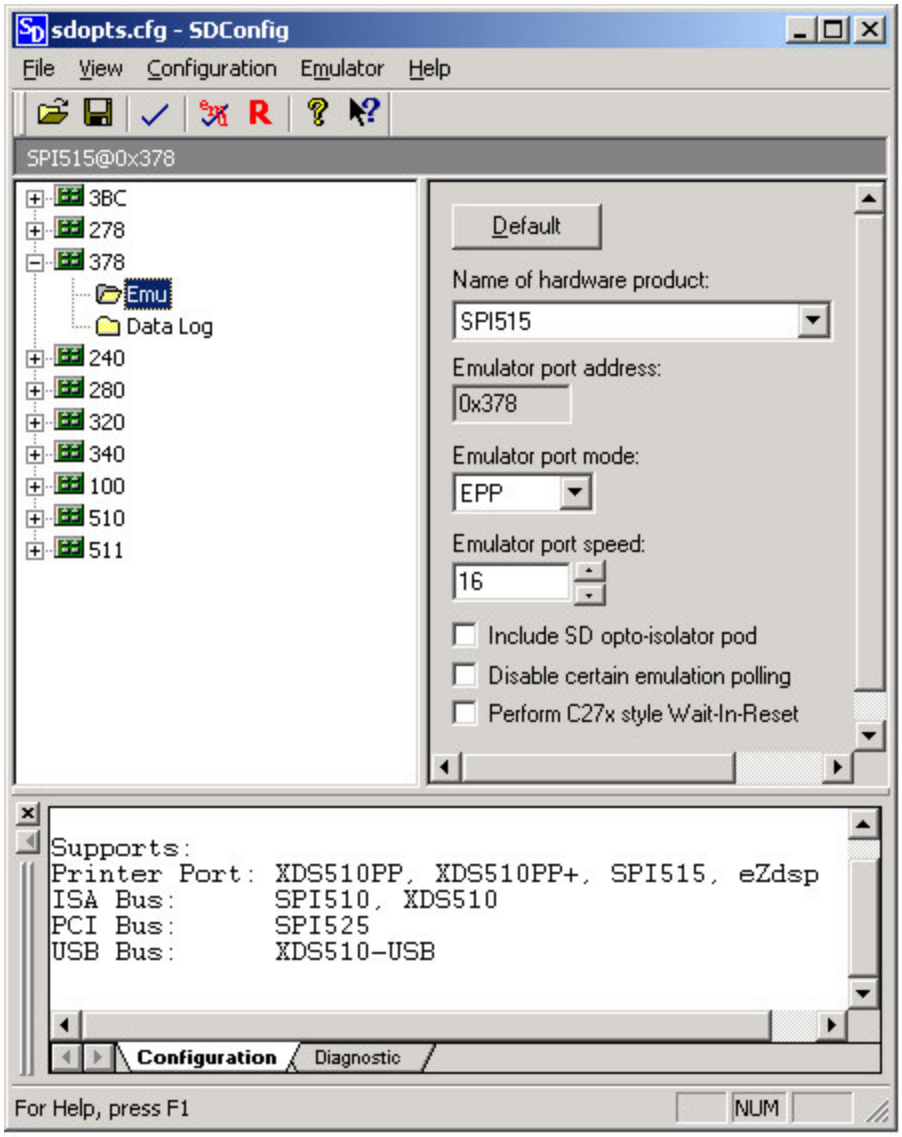
For PP emulator typical port is 378 then Emu. After the selection you should get a screen similar to the following.



This is an example setup for OMAP16XX using SPI515 emulator and EPP mode. The "Emulator port speed" setting of 8 was sufficient to support a 2 MHz RTCK frequency. Increasing the speed number adds in more delay for emulation accesses. Reducing the speed number reduces delay. This configuration has "Disable certain emulation polling" checked. In this setup the emulation drivers will insert at least 8 emulator wait-states on emulation cycles. But it will not verify the emulation controller is ready for the next cycle. This is the preferred mode of operation when the target system TCK and RTCK is above 5 MHz. In this mode the emulation driver assumes all accesses can complete in a timely fashion, which yields highest overall emulation and debugger performance.

An alternative approach, which is **recommended** for OMAP16xx is to **uncheck** "Disable certain emulation polling" and then increase the speed number to 16 (see below). In this configuration the emulation driver will poll the emulation controller ready signal to ensure that the previous emulation access is complete. In this configuration the speed setting becomes a timeout. So for the settings below the emulation controller will poll up to 16 times before timing out. The benefit to this method is that it is adaptive. When the OMAP16XX slows down then the overall emulation performance will slow but still function. As the RTCK frequency so will overall debug performance. As a rule of thumb the speed number for PP class emulators can be estimated as (RTCK clock period ns * 16)/500ns. In a typical OMAP16XX setup RTCK starts up

at 2 MHz or 500ns, which results in a speed setting of 16. The 500ns divisor is based on PP class emulator architecture and is not related to the RTCK or the target.



With either setup you can verify your settings by doing the following:

1. Emulator->Reset. The emulator should reset and both TCK and RTCK should be clocking at the JTAG header. On OMAP16xx expect RTCK to be slower then TCK. Per one test setup a TCK of 10MHz shows a RTCK of 2 MHz assuming the OMAP16xx PLL is in bypass.
2. Emulator->Test. This should return without error. For OMAP16XX device SdConfig should show a JTAG IR length of 50 with 3 devices in the scan chain. This represents an 8-bit bypass device, a 4-bit ARM926 and 38-bit C55xx device.
3. Emulator->Diagnostics. Set the "Test Loop" to 50 and select start. This will do a Test Loop * 10,000 bit scan test. This may take on the order of 20 seconds at 2 MHz with suggested settings.

If any of these tests fail then it is not likely you will be able to get reliable CCS operation. You should adjust the speed setting until you get reliable operation for your RTCK frequency.

2.1 Emulator Settings Summary

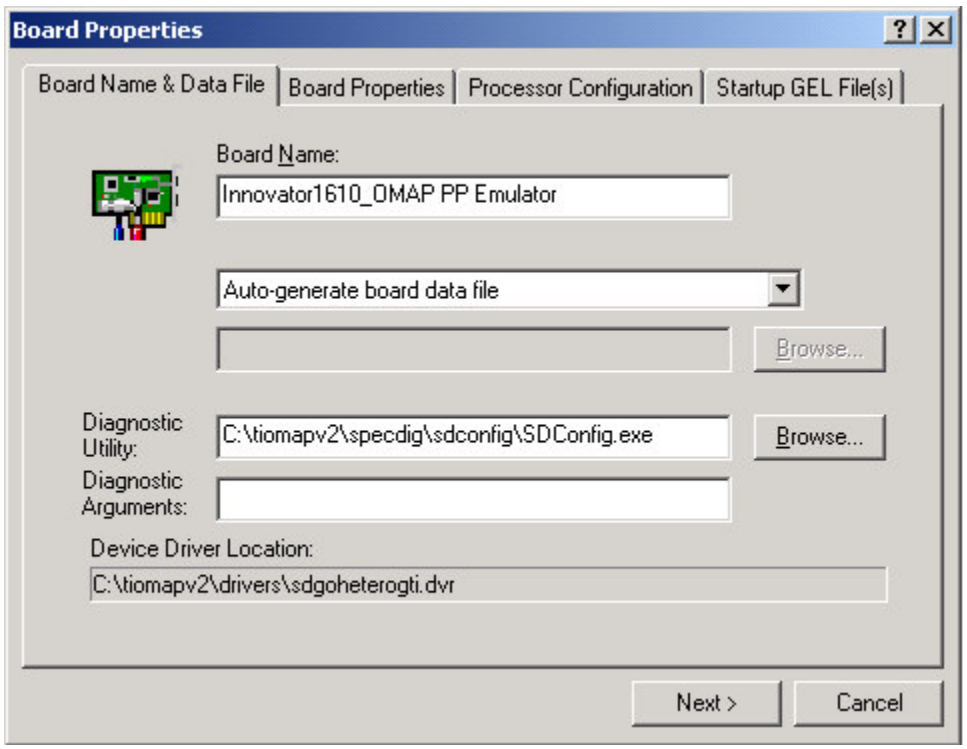
The following table summarizes the critical emulator settings for various emulators and port modes. These tests were conducted on OMAP16XX with TCK input clock of 10 MHz and 12 MHz OMAP16XX input clock. The resulting RTCK after power up was 2 MHz. Keep in mind that if your target powers up and sets the OMAP16XX PLL on power up your RTCK may be at a higher or lower frequency. However at no time will the frequency of RTCK exceed that of TCK.

Emulator	**Port Mode**	**Speed**	**Emulation Polling**
XDS510PP+ SPI515	SPP8	4	Checked
XDS510PP+ SPI515	EPP	8	Checked
XDS510PP+ SPI515	SPP8	16	Unchecked
XDS510PP+ SPI515	SPP8	16	Unchecked
SPI525	PCI	16	Unchecked

NOTE: The obsolete XDS510 ISA and XDS510PP were not included in the test.

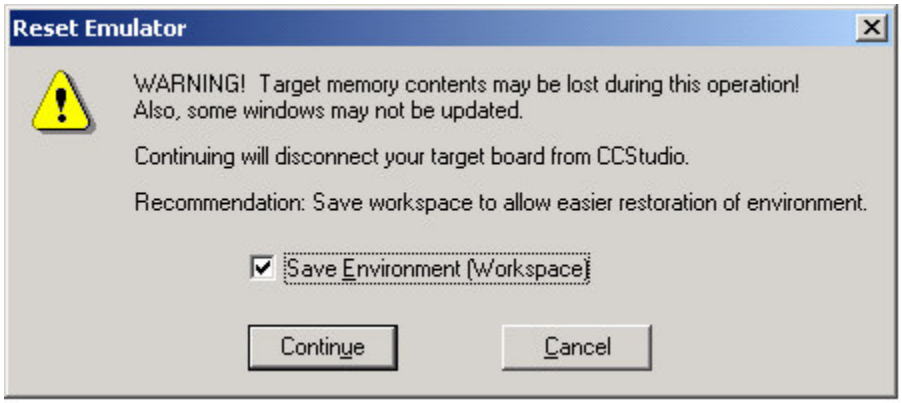
3. Setting up CCS

Follow the general TI instructions for running and setting up CCS. One additional setting you may want to make to your CC setup if not already done is to include a link to the SD configuration utility. This utility will be located in your CCS install directory under the subdirectory "specdig\sdconfig". By including this link you can launch SdConfig from CCS through the Debug->Reset Emulator sequence.

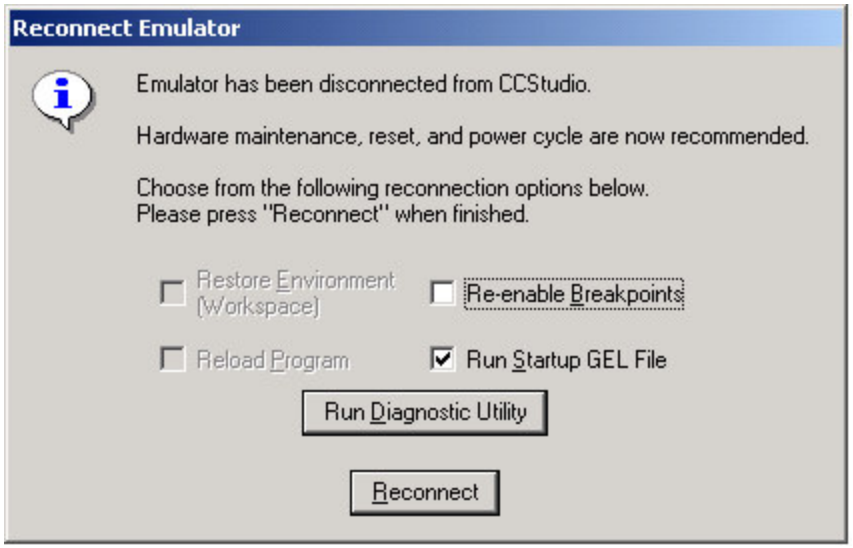


3.1 Recovering from Target Errors

CCS 2.20 and later provides a means to recover from many target errors without having to leave CCS. If you encounter a fatal emulation error then CCS will prompt you through a series of dialog boxes that will let you disconnect from your target, reset the emulator and return to CCS. You can also perform a Debug->Reset Emulator from inside CCS which will result in the following dialog:



If you select continue then you will have the option to run SdConfig via the "Run Diagnostic Utility". When you are finished testing your target via SdConfig you should exit then select the reconnect button.



Although the CCS dialogs allow you to save and restore environment testing has shown this to be problematic in a heterogeneous debug environment. Creating gel scripts to setup as much of your environment as possible seems to work much better then workspaces. The other problem with workspaces is that if you attempt to save a workspace and get CCS or target errors then the state of your workspace may be in an indeterminate state. This makes clean recovery next to impossible.

3.1.1 Device Reset and RTCK

Per the introduction, OMAP16xx devices have a specific problem area related to the ARM926 implementation of RTCK. Although RTCK is not constant it does not create a significant issue to emulation or debug if you have used SdConfig to properly setup your emulator. The significant problem is the effects of device reset on RTCK. If the OMAP16xx receives a system reset the RTCK is asynchronously disabled which may cause glitches on the RTCK signal. If this occurs the state of emulation controller is indeterminate. This will generally show up in CCS as a fatal emulator error either on ARM9 or the C55xx. At this point about the only solution is to go through the Reset Emulator path to get the device and emulator back to a good state. In general a power cycle is not required to recover but suggested per the CCS dialogs. At a minimum you should do a target reset. If you cannot reconnect and get into a good state then a power cycle is recommended.

Once you have CCS started, preventing a device level reset is extremely important to successful debug.

3.1.2 Booting from Uninitialized Memory

If an ARM9x device powers up and begins executing from uninitialized memory then behavior is indeterminate. Worse case the ARM9 could get itself into a state such that the emulator cannot connect. When this occurs the emulation driver will install a reset trap and then return failure. At this point CCS should fail and pop up a dialog box with an option to Ignore, Retry or Abort. You can then perform a hardware reset on your target and then select Retry in the CCS dialog. In theory the ARM9x device will have seen the reset and trapped before running random code. The caveat is that if the hardware reset glitches RTCK the emulator may not be in a deterministic state. At this point your only choice is to reset the emulator and retry. Unfortunately when you reset the emulator this sends the ARM9x JTAG tap to the

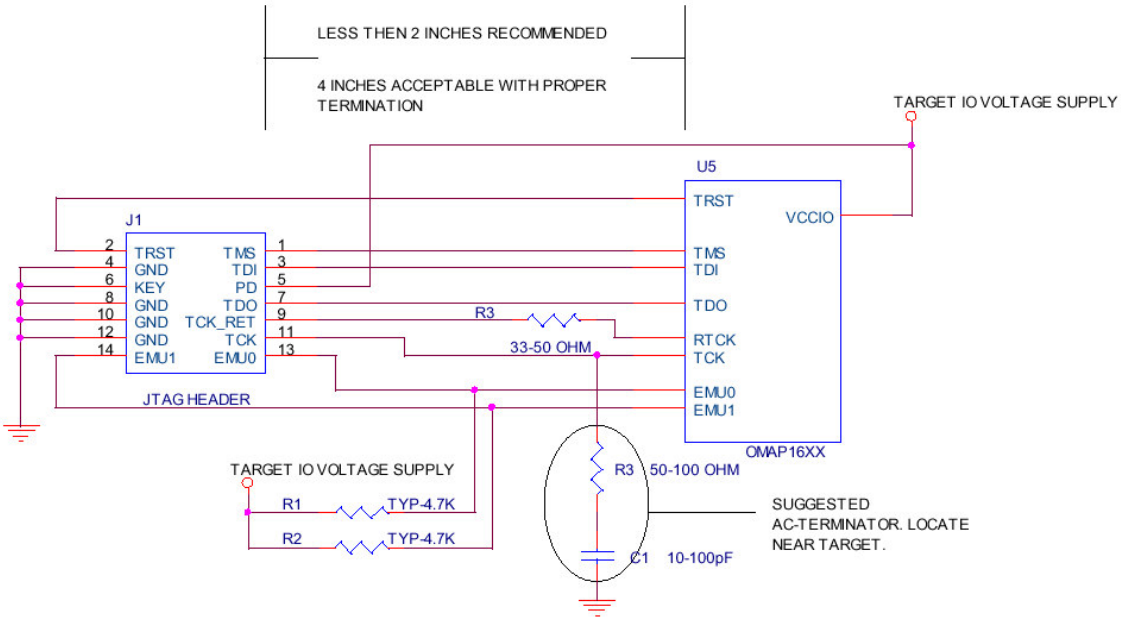
test-logic-reset state and clears the reset trap. So the end result is you can get into an endless retry loop trying to get a good connection to the ARM926. The recommended solution is to program a spin loop into FLASH so that ARM926 does not run off and execute garbage. In a worse case scenario you may never get connected to ARM926 via JTAG to program the FLASH through a JTAG based tool. Fortunately testing has shown that OMAP16XX connection success is actually much higher then on previous OMAP devices once you have your emulator configured properly.

3.1.3 PLL and RTCK

For all OMAP devices RTCK is synchronized to the ARM92x core clock, which is generally based off of the PLL. So changes in the PLL frequency can impact RTCK. This can have adverse effects on emulation if the device clock becomes very slow. Testing has shown that this generally becomes a problem when the PLL is divided down or goes into low power sleep modes. In these cases RTCK may become very slow, in the low KHz range or go to a DC level. When PLL goes to low KHz range then your emulator Speed setting may need to be adjusted. For the XDS510PP PLUS, SPI515 or SPI525 class emulators. For these emulators uncheck "Emulation Polling" and set the "Speed" to 3000. With these settings low level emulation drivers will use the "Speed" as a timeout vs. a pure delay.

4. Emulation Connection to OMAP16xx

The OMAP16xx device supplies the emulation clock via RTCK. This requires a slight modification of the typical emulator to device connection. As shown in the schematic example below the emulator TCK signal (pin 11 of header) is connected to OMAP16xx TCK pin. The OMAP16xx RTCK pin is then connected to the emulator TCK_RET signal (pin 9 of header). Both signals are terminated to reduce transmission line effects. *Noise on the TCK/TCK_RET signals is the number 1 cause of emulation failures to date. So adding the extra components is highly recommended on development boards.*



4.1.1 Other Devices in OMAP16xx Scan Chain

Due to RTCK implementation the OMAP16xx devices are not well suited to having other JTAG devices in the scan chain. This is especially true if multiple OMAP16xx devices are needed in the same system. In this case each OMAP16xx device wants to supply the RTCK signal back to the emulator, which is not possible. The only plausible solution is multiple JTAG headers and multiple emulators if necessary.

It is possible to include other JTAG devices in an OM16xx scan chain as long as they meet the following criteria:

1. They do not need to supply the TCK_RET to the emulator.
2. They can tolerate a modulating TCK, i.e. RTCK.
3. ***They can recover from glitches in TCK. This is the criterion, which will prevent most all JTAG devices from being included in an OMAP16xx scan chain.***

If these criterions are met then it is possible to include other JTAG devices in the OMAP16xx scan chain.