Draft: OpenMoko Production Boot Environment (PBE)

Milosch Meriac

September 7, 2007

https://wiki.internal.openmoko.org/wiki/PBE

Content

Room for improvements

- vital SDRAM test and other peripheral tests are missing
- vital high/low voltage tests at board level are missing
- missing in-depth stray tests

Room for improvements

- vital SDRAM test and other peripheral tests are missing
- vital high/low voltage tests at board level are missing
- missing in-depth stray tests
- very high training effort needed for operators because of non-optimized Windows software
- unstable test software (very often exceptions occur, board level test need to be started multiple times)

Room for improvements

- vital SDRAM test and other peripheral tests are missing
- vital high/low voltage tests at board level are missing
- missing in-depth stray tests
- very high training effort needed for operators because of non-optimized Windows software
- unstable test software (very often exceptions occur, board level test need to be started multiple times)
- no tools available for automated quality/progress control during production

Room for improvements

- vital SDRAM test and other peripheral tests are missing
- vital high/low voltage tests at board level are missing
- missing in-depth stray tests
- very high training effort needed for operators because of non-optimized Windows software
- unstable test software (very often exceptions occur, board level test need to be started multiple times)
- no tools available for automated quality/progress control during production
- most peripheral tests are not automated yet - test quality depends mostly on operator and takes too long
- tests need to be automated to allow automatic quality tests - vitally important to enable process statistics which in turn allow to find component and design problems quickly

Room for improvements

- vital SDRAM test and other peripheral tests are missing
- vital high/low voltage tests at board level are missing
- missing in-depth stray tests
- very high training effort needed for operators because of non-optimized Windows software
- unstable test software (very often exceptions occur, board level test need to be started multiple times)
- no tools available for automated quality/progress control during production
- most peripheral tests are not automated yet - test quality depends mostly on operator and takes too long
- tests need to be automated to allow automatic quality tests - vitally important to enable process statistics which in turn allow to find component and design problems quickly
- important to log product serial number (PSN) at each individual station electronically to verify the process timing in real time

Room for improvements

- vital SDRAM test and other peripheral tests are missing
- vital high/low voltage tests at board level are missing
- missing in-depth stray tests
- very high training effort needed for operators because of non-optimized Windows software
- unstable test software (very often exceptions occur, board level test need to be started multiple times)
- no tools available for automated quality/progress control during production
- most peripheral tests are not automated yet - test quality depends mostly on operator and takes too long
- tests need to be automated to allow automatic quality tests - vitally important to enable process statistics which in turn allow to find component and design problems quickly
- important to log product serial number (PSN) at each individual station electronically to verify the process timing in real time
- very complicated fixture setup - needs to be simplified

Room for improvements

- vital SDRAM test and other peripheral tests are missing
- vital high/low voltage tests at board level are missing
- missing in-depth stray tests
- very high training effort needed for operators because of non-optimized Windows software
- unstable test software (very often exceptions occur, board level test need to be started multiple times)
- no tools available for automated quality/progress control during production
- most peripheral tests are not automated yet - test quality depends mostly on operator and takes too long
- tests need to be automated to allow automatic quality tests - vitally important to enable process statistics which in turn allow to find component and design problems quickly
- important to log product serial number (PSN) at each individual station electronically to verify the process timing in real time
- very complicated fixture setup - needs to be simplified

Parallel tests automatically assigned to fixtures

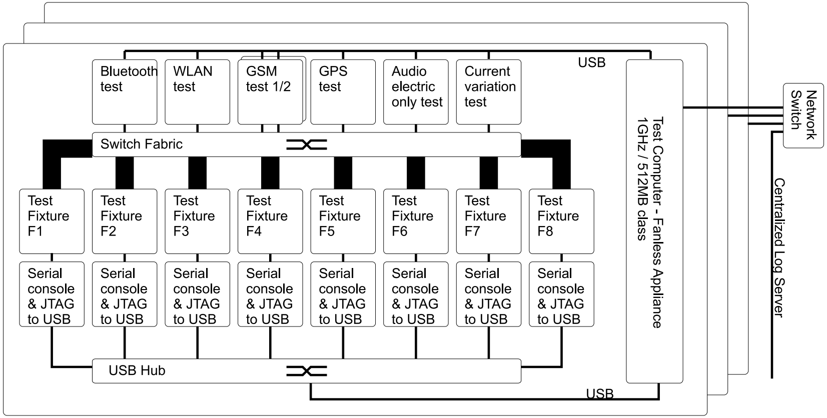

Figure: All switch fabric fixtures are identical - the same broadband antenna is used for tests like WLAN, Bluetooth, GSM and GPS

Parallel tests automatically assigned to fixtures

Example switch fabric setup for board level tests

- all fixtures are identical
- switch fabric can route test equipment to individual fixtures depending on their test progress
- all measurement equipment is connected to crossbar switch
- operator inserts untested PCB into HF shielded fixture compartment
- as soon as the compartment is closed, tests will start automatically
- while external tests are running (GSM etc.) internal tests can run in parallel (bad block scan etc.)
- orange LED (red+green) indicates an ongoing test process
- as soon as tests are over, a blinking green LED indicates success - a failure is indicated by a blinking red LED

Parallel tests automatically assigned to fixtures

Advantages

- higher scaling & availability
- better utilization of test equipment
- time saved, PCB doesn't need to be removed & replugged into different fixtures
- no time is lost for reboots because of fixture change
- while external tests run, internal test can be executed
- inside each device multiple test can run in parallel
- example: while one compartment runs the GSM test, the other compartment can run a Bluetooth test
- the smaller ammount of different fixture types results in higher availability because they can be stocked easier
- provides spare fixtures. In case a fixture breaks it will decrease downtime

Fixture improvements

Issue

Fixture designs are too complicated and include plenty of manual work. Duplication is time consuming and expensive. Desing are not modular - they must be basically thrown away if design changes.

Solution

Fixture design can be massively simplified by creating a modular fixture PCB (two standardized PCBs - the bottom PCB contains traces for the fixtures and terminates on a station-specific connector. The top PCB contains only drill for maintaining the position of the pogo test pins and is affixed by distance bolts to the bottom PCB. If pogo pins break one can easily swap the PCBs stack quickly. Multiple design can be supported just by swapping the PCB stack - if a fixture breaks, the fixture PCB can be replaced quickly.

Fixture improvements

Issue

Fixture designs are too complicated and include plenty of manual work. Duplication is time consuming and expensive. Desing are not modular - they must be basically thrown away if design changes.

Solution

Fixture design can be massively simplified by creating a modular fixture PCB (two standardized PCBs - the bottom PCB contains traces for the fixtures and terminates on a station-specific connector. The top PCB contains only drill for maintaining the position of the pogo test pins and is affixed by distance bolts to the bottom PCB. If pogo pins break one can easily swap the PCBs stack quickly. Multiple design can be supported just by swapping the PCB stack - if a fixture breaks, the fixture PCB can be replaced quickly.

1st. stage: Boot Loader

- 2-3kByte size
- copied into internal SRAM of ARM processor over JTAG
- massively stripped down u-boot code
- verifies serial debug port
- minimal setup of processor, PLL, SDRAM and SPI port (optional)

1st. stage: Boot Loader

- 2-3kByte size
- copied into internal SRAM of ARM processor over JTAG
- massively stripped down u-boot code
- verifies serial debug port
- minimal setup of processor, PLL, SDRAM and SPI port (optional)
- executes quick SDRAM test on demand
- executes full SDRAM test on demand

1st. stage: Boot Loader

- 2-3kByte size
- copied into internal SRAM of ARM processor over JTAG
- massively stripped down u-boot code
- verifies serial debug port
- minimal setup of processor, PLL, SDRAM and SPI port (optional)
- executes quick SDRAM test on demand
- executes full SDRAM test on demand
- loads minimal Linux based test environment (<3MB including initial ramdisk) over SPI (>=25MHz clock) or serial debug port
- the serial flash is connected to the SPI socket on the debug board

1st. stage: Boot Loader

- 2-3kByte size
- copied into internal SRAM of ARM processor over JTAG
- massively stripped down u-boot code
- verifies serial debug port
- minimal setup of processor, PLL, SDRAM and SPI port (optional)
- executes quick SDRAM test on demand
- executes full SDRAM test on demand
- loads minimal Linux based test environment (<3MB including initial ramdisk) over SPI (>=25MHz clock) or serial debug port
- the serial flash is connected to the SPI socket on the debug board
- executes Linux based test environment

1st. stage: Boot Loader

- 2-3kByte size
- copied into internal SRAM of ARM processor over JTAG
- massively stripped down u-boot code
- verifies serial debug port
- minimal setup of processor, PLL, SDRAM and SPI port (optional)
- executes quick SDRAM test on demand
- executes full SDRAM test on demand
- loads minimal Linux based test environment (<3MB including initial ramdisk) over SPI (>=25MHz clock) or serial debug port
- the serial flash is connected to the SPI socket on the debug board
- executes Linux based test environment

2nd. Stage Linux based Production Boot Environment (1/3)

- minimal kernel, total boot time around 7 seconds
- if persistent memory is not initialized yet, an unique production serial number (PSN) needs to be entered over serial console
- after persistent memory is initialized (kernel module), the PSN is stored
- single user mode based on busybox/uClibc
- all control and output over serial debug port

2nd. Stage Linux based Production Boot Environment (1/3)

- minimal kernel, total boot time around 7 seconds
- if persistent memory is not initialized yet, an unique production serial number (PSN) needs to be entered over serial console
- after persistent memory is initialized (kernel module), the PSN is stored
- single user mode based on busybox/uClibc
- all control and output over serial debug port
- "upstart" init environment will be used to to execute parallel background tests based on signals received over debug port or network
- no static drivers included - all drivers as modules and started manually via upstart signals

2nd. Stage Linux based Production Boot Environment (1/3)

- minimal kernel, total boot time around 7 seconds
- if persistent memory is not initialized yet, an unique production serial number (PSN) needs to be entered over serial console
- after persistent memory is initialized (kernel module), the PSN is stored
- single user mode based on busybox/uClibc
- all control and output over serial debug port
- "upstart" init environment will be used to to execute parallel background tests based on signals received over debug port or network
- no static drivers included - all drivers as modules and started manually via upstart signals
- selected kernel drivers will be extended with and debug-only messages and sanity assertions

2nd. Stage Linux based Production Boot Environment (1/3)

- minimal kernel, total boot time around 7 seconds
- if persistent memory is not initialized yet, an unique production serial number (PSN) needs to be entered over serial console
- after persistent memory is initialized (kernel module), the PSN is stored
- single user mode based on busybox/uClibc
- all control and output over serial debug port
- "upstart" init environment will be used to to execute parallel background tests based on signals received over debug port or network
- no static drivers included - all drivers as modules and started manually via upstart signals
- selected kernel drivers will be extended with and debug-only messages and sanity assertions

2nd. Stage Linux based Production Boot Environment (2/3)

- logging based on kernel messages - also utilized by user processes to unify both worlds
- logging via kernel messages allows interleaved messages by user code that executes kernel functions
- kernel logs are reflected to serial debug port and stored locally
- as soon as USB is verified, port is switched to USB host mode to connect to an external USB-to-Ethernet network interface

2nd. Stage Linux based Production Boot Environment (2/3)

- logging based on kernel messages - also utilized by user processes to unify both worlds
- logging via kernel messages allows interleaved messages by user code that executes kernel functions
- kernel logs are reflected to serial debug port and stored locally
- as soon as USB is verified, port is switched to USB host mode to connect to an external USB-to-Ethernet network interface
- a station is defined by the MAC/IP address of the USB-to-ethernet converter. A new converter can be easily assigned to a station via network interface.
- individual steps are executed by the device based on the individual station assignment (via IP)
- log level and special prefix is used to issue standardized test results - easy to filter for automated processing

2nd. Stage Linux based Production Boot Environment (2/3)

- logging based on kernel messages - also utilized by user processes to unify both worlds
- logging via kernel messages allows interleaved messages by user code that executes kernel functions
- kernel logs are reflected to serial debug port and stored locally
- as soon as USB is verified, port is switched to USB host mode to connect to an external USB-to-Ethernet network interface
- a station is defined by the MAC/IP address of the USB-to-ethernet converter. A new converter can be easily assigned to a station via network interface.
- individual steps are executed by the device based on the individual station assignment (via IP)
- log level and special prefix is used to issue standardized test results - easy to filter for automated processing

2nd. Stage Linux based Production Boot Environment (3/3)

- by design multiple test can run at the same time. The process name/ID can be used to check which message belongs to which process, because test like bad block ckeck, root file system download, display test and GSM test can be run in parallel easily
- isolated network from FIC network - only the database server sits in both networks and is accessible from outside via VPN from outside the factory by OpenMoko Inc
- single phones can be initialized with an updated PBE over JTAG automatically. Because such newer PBE version will be connected with the PSN, also different server processes can be spawned for testing. This allows to slipstream new PBEs before switching the whole production to the updated PBE. It also allows to have branded versions of phones produced easily.
- after an updated PBE is verified, the serial flash on the debug board can be flashed to the latest version by the current PBE. This allows full control and high speed downloads of PBEs.

High Level Design Rules (1/2)

- ensure that test software can be excecuted fully automatically and all test results are stored to a central server instantly
- ensure scalability and high availability

High Level Design Rules (1/2)

- ensure that test software can be excecuted fully automatically and all test results are stored to a central server instantly
- ensure scalability and high availability
- ensure flexibility & modularity to allow painless design changes

High Level Design Rules (1/2)

- ensure that test software can be excecuted fully automatically and all test results are stored to a central server instantly
- ensure scalability and high availability
- ensure flexibility & modularity to allow painless design changes
- ensure automated processing of production logs file to be able to compute a real time quality index

High Level Design Rules (1/2)

- ensure that test software can be excecuted fully automatically and all test results are stored to a central server instantly
- ensure scalability and high availability
- ensure flexibility & modularity to allow painless design changes
- ensure automated processing of production logs file to be able to compute a real time quality index
- solely use resources of third parties as long as the results fully belong to OpenMoko Inc (for examlpe documentation, hardware, firmware or test software)

High Level Design Rules (1/2)

- ensure that test software can be excecuted fully automatically and all test results are stored to a central server instantly
- ensure scalability and high availability
- ensure flexibility & modularity to allow painless design changes
- ensure automated processing of production logs file to be able to compute a real time quality index
- solely use resources of third parties as long as the results fully belong to OpenMoko Inc (for examlpe documentation, hardware, firmware or test software)

High Level Design Rules (2/2)

- ensure to have full control, knowledge and documentation about the production process and especially from all software tools involved. The only exception are standardized tool(kits)s that can be openly obtained
- ensure that all test steps are independent of human estimations

High Level Design Rules (2/2)

- ensure to have full control, knowledge and documentation about the production process and especially from all software tools involved. The only exception are standardized tool(kits)s that can be openly obtained
- ensure that all test steps are independent of human estimations
- making sure that whole process step can be verified and logged over network. Regard involved parties as untrusted and make test process bullet-proof by tracking devices electronically at every station.

High Level Design Rules (2/2)

- ensure to have full control, knowledge and documentation about the production process and especially from all software tools involved. The only exception are standardized tool(kits)s that can be openly obtained
- ensure that all test steps are independent of human estimations
- making sure that whole process step can be verified and logged over network. Regard involved parties as untrusted and make test process bullet-proof by tracking devices electronically at every station.
- ensure that equipment and human resources are utilized efficiently

High Level Design Rules (2/2)

- ensure to have full control, knowledge and documentation about the production process and especially from all software tools involved. The only exception are standardized tool(kits)s that can be openly obtained
- ensure that all test steps are independent of human estimations
- making sure that whole process step can be verified and logged over network. Regard involved parties as untrusted and make test process bullet-proof by tracking devices electronically at every station.
- ensure that equipment and human resources are utilized efficiently

Advantages

- near-zero peripheral dependencies
- clean evironment - programmers with low experience can write test routines in Linux User mode
- full network support
- cleanly separated processed to ease debugging
- multitasking to allow parallel execution of multiple tests
- clean log system
- full USB/network support at a very early stage
- existing drivers can be easily extended to sophisticated test routines
- automatic resolving of test dependencies
- USB host mode allows external test peripherals like USB cameras and sound cards for fully automated tests additionally to the USB-Ethernet converter