Android Multilib Build Cheat Sheet
Android Multilib Build Cheat Sheet

● AOSP build configurations
  ○ 32-bit and 64-bit only builds
  ○ Multilib builds

● How to do a Multilib build?
  ○ Multilib platform configuration
  ○ Building Multilib modules

● Multilib examples from android-5.1.0_r1
  ○ Platform configuration example
  ○ Multilib module build example
AOSP Build Configurations

- **32-bit and 64-bit only builds**
  - Android build for a single target cpu arch i.e. either 32-bit or 64-bit.

- **Multilib builds**
  - Android build for two target cpu archs e.g. 64-bit primary and 32-bit secondary, or 32-bit primary and 64-bit secondary.
32-bit and 64-bit only builds

- **32-bit only build**
  - Target support 32-bit applications only
  - Build 32-bit Android binaries to run on 32-bit targets
  - Generate huge interest even on 64-bit targets

- **64-bit only build**
  - Target support 64-bit applications only
  - Build 64-bit Android binaries to run on 64-bit targets
  - Build not yet ready for a day to day use. Builds successfully but doesn’t boot up. Last tried booting on stock android-5.1.0_r1.
Multilib builds

- Multi-target build configuration for 64-bit targets
- Support building binaries for two target cpu archs in the same build, with a primary and a secondary arch configuration.
- Target can support both 32-bit and 64-bit applications
Multilib builds

- **64-bit Primary and 32-bit Secondary (aka 64_32)**
  - 64-bit arch is configured as the Primary arch and 32-bit as Secondary
  - 64-bit is the default target for modules if not configured otherwise locally
  - system_server will run as a 64-bit process

- **32-bit Primary and 64-bit Secondary (aka 32_64)**
  - Build configuration contrary to 64bit Primary and 32bit Secondary
  - Theoretically possible, traces still available in AOSP
    (system/core/rootdir/init.zygote32_64.rc)
  - Configuration might have been dropped somewhere in the development cycle. Build is broken for stock android-5.1.0_r1

    art/build/Android.common.mk:42: *** Do not know what to do with this multi-target configuration!. Stop.
Multilib builds

- **Zygote configuration**
  - Primary and Secondary zygotes
    - Multilib builds run two zygote processes
    - Primary zygote and Secondary zygote
    - To support both 64bit and 32bit applications
  - Starting Lollipop, zygote init config is not part of init.rc anymore.
    - init.rc include init.${ro.zygote}.rc at runtime which initialize zygotes
    - Enable/Select Multilib zygote in product config:
      - PRODUCT_DEFAULT_PROPERTY_OVERRIDES += ro.zygote=zygote64_32
      - PRODUCT_COPY_FILES += system/core/rootdir/init.zygote64_32.rc:root/init.zygote64_32.rc
Multilib builds

- Dissecting `/init.zygote64_32.rc`:

  ```
  service zygote /system/bin/app_process64 -Xzygote /system/bin --zygote --start-system-server --socket-name=zygote
class main
socket zygote stream 660 root system
onrestart write /sys/android_power/request_state wake
onrestart write /sys/power/state on
onrestart restart media
onrestart restart netd
  
  "service zygote" → `/system/bin/app_process64` → Primary Zygote
  "--start-system-server" → `system_server` → 64-bit process
  
  service zygote_secondary /system/bin/app_process32 -Xzygote /system/bin --zygote --socket-name=zygote_secondary
class main
socket zygote_secondary stream 660 root system
onrestart restart zygote
  
  "service zygote_secondary" → `/system/bin/app_process32` → Secondary Zygote
  ```
How to do a Multilib build?

- **Multilib platform configuration**
  - Configure target archs and abis
  - Application/Executables support
  - Custom toolchains

- **Building Multilib modules**
  - Local build flags
  - Building arch specific modules
  - Binary installation path
  - Handling pre-built modules
  - Dex-preopt and generated sources
Multilib Platform Configuration

Configure target CPU archs and ABIs in `BoardConfig.mk`

- **Primary arch:**
  - `TARGET_ARCH` and `TARGET_CPU_*` variables defined as usual
    - `TARGET_ARCH := arm64`
    - `TARGET_ARCH_VARIANT := armv8-a`
    - `TARGET_CPU_VARIANT := generic`
    - `TARGET_CPU_ABI := arm64-v8a`

- **Secondary arch:**
  - Android build system uses `TARGET_2ND_*` variables to set up an additional compilation environment for the secondary arch
    - `TARGET_2ND_ARCH := arm`
    - `TARGET_2ND_ARCH_VARIANT := armv7-a-neon`
    - `TARGET_2ND_CPU_VARIANT := cortex-a15`
    - `TARGET_2ND_CPU_ABI := armeabi-v7a`
    - `TARGET_2ND_CPU_ABI2 := armeabi`
Multilib Platform Configuration

- Application/Executables Support
  - To build 32-bit executables and apps by default, set `TARGET_PREFER_32_BIT := true`
  - Set `TARGET_SUPPORTS_32_BIT_APPS` and `TARGET_SUPPORTS_64_BIT_APPS` to choose which native libraries to build for an app.
    - If both are set, it will build 64-bit apps unless `TARGET_PREFER_32_BIT` is set or it is overridden by module-specific local variables in `Android.mk`
    - If only one is set, it will only build apps that work on that particular arch.
    - If neither is set it will fall back to only building 32bit apps unless overridden by `Android.mk` config.
Multilib Platform Configuration

- Set Custom Toolchains
  - Set `TARGET_GCC_VERSION_EXP`, if you are using a common GCC toolchain version for both the archs.
    - For example, to use custom 4.9-linaro toolchains to build both 32-bit and 64-bit binaries, set:
      ```
      TARGET_GCC_VERSION_EXP := 4.9-linaro
      ```

      The build system in this case will pick both 32-bit and 64-bit custom 4.9-linaro toolchains from default prebuilts toolchain path
      i.e. `prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9-linaro` and `prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9-linaro`.
Multilib Platform Configuration

- Set `TARGET_TOOLCHAIN_ROOT` and `2ND_TARGET_TOOLCHAIN_ROOT` to use different toolchain versions for 64-bit and 32-bit binaries.

  - For example, set custom 4.9-linaro toolchain for primary arch and stock 4.9 toolchain for secondary arch:

    ```
    TARGET_TOOLCHAIN_ROOT := prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9-linaro
    2ND_TARGET_TOOLCHAIN_ROOT := prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9
    ```
Building Multilib Modules

- Building an Android module with Multilib support
  - Module names in product configuration, \texttt{PRODUCT\_PACKAGES}, together with the dependency graph decides what binaries will be built and installed to the system image.
    - For libraries pulled in by dependency, a 32-bit library is only installed if it's required by a 32-bit library or executable. The same is true for 64-bit libraries.
    - For executables, by default the build system builds only the 64-bit version, but this build rule can be overridden by \texttt{TARGET\_PREFER\_32\_BIT} or \texttt{LOCAL\_32\_BIT\_ONLY} module-scoped local variable.

Note: Module names on the make command line cover only the 64-bit version build. For example, after running “lunch aosp\_arm64-eng”, “make libc” builds only the 64-bit libc. To build the 32-bit libc, you need to run “make libc\_32”.
Building Multilib Modules

- **Module definition in** Android.mk
  
  Set `LOCAL_MULTILIB` to build for 64-bit and/or 32-bit archs. It overrides the global `TARGET_PREFER_32_BIT`.

  - `LOCAL_MULTILIB := first`, build module for the first arch (64-bit on a 64-bit target, 32-bit on a 32-bit target). Same as `LOCAL_NO_2ND_ARCH := true`
  - `LOCAL_MULTILIB := 32`, build only 32-bit, same as `LOCAL_32_BIT_ONLY := true`
  - `LOCAL_MULTILIB := 64`, build only 64-bit.
  - `LOCAL_MULTILIB := both`, build for both architectures on a Multilib target.
  - `LOCAL_MULTILIB := ""`, build depends on other global or `LOCAL_*` module-scoped variables.
Building Multilib Modules

○ Local build variables:

To set up a custom local build env, use the LOCAL_* variables.

■ Set an arch-specific variable, LOCAL_* variable with a target arch suffix i.e. LOCAL_* $(TARGET_ARCH) and LOCAL_* $(TARGET_2ND_ARCH).
  ● For example:
    LOCAL_CFLAGS_arm64 += -DARCH_ARM64_HAVE_NEON
    LOCAL_SRC_FILES_arm := xyz_arm.c

■ Or set LOCAL_* variable with a _32 or _64 suffix based on whether to build for 32-bit or 64-bit, independent of target arch.
  ● For example:
    LOCAL_CFLAGS_64 += -DARCH_GENERIC_HAVE_ABC
    LOCAL_SRC_FILES_32 += xyz_generic.c

Note: Not all LOCAL_* variables support arch/target specific variants. Refer to build/core/clear_vars.mk for an up-to-date list.
Building Multilib Modules

- **Building for specific arch(s):**
  To drive an arch-specific build, use the following variables.
  
  - `LOCAL_MODULE_TARGET_ARCH` and `LOCAL_MODULE_UNSUPPORTED_TARGET_ARCH` specifies that a module can or cannot be built for one or more architectures.
    
    ```
    LOCAL_MODULE_TARGET_ARCH := "arm arm64 x86_64"
    LOCAL_MODULE_UNSUPPORTED_TARGET_ARCH := "arm arm64 .."
    ```
  
  - `LOCAL_MODULE_TARGET_ARCH_WARN` and `LOCAL_MODULE_UNSUPPORTED_TARGET_ARCH_WARN` are same, but warn that the arch is not supported, which is useful for modules that are critical but not yet working.
Building Multilib Modules

- **Installation Path:**
  - Libraries: `/system/lib` always host 32-bit libraries, and `/system/lib64` 64-bit libraries.
  - Executables: If you build an executable as both 32-bit and 64-bit, then either set `LOCAL_MODULE_STEM_{32,64}` to distinguish the installed file name, or set `LOCAL_MODULE_PATH_{32,64}` to distinguish the install path.
  - In multilib builds the install location depends on the CPU target. Set `LOCAL_MODULE_RELATIVE_PATH` to set the install location instead of `LOCAL_MODULE_PATH`.
    - For example, HALs will generally use: `LOCAL_MODULE_RELATIVE_PATH := hw`
Building Multilib Modules

- Handling pre-built Multilib modules:
  - Set `LOCAL_SRC_FILES$(ARCH_SUFFX)` to point to arch specific prebuilt binaries, similarly `LOCAL_SRC_FILES{32,64}` can be used for arch independent target binaries.
  - `$({TARGET_ARCH})` and `$({TARGET_2ND_ARCH})` can’t be used reliably to tell the build system what arch the prebuilt binary is targeted for. Use `LOCAL_MODULE{,_UNSUPPORTED}_TARGET_ARCH` local variables instead.
  - All the build rules for Multilib executables hold true for pre-built executables as well. For example: if you don’t provide `LOCAL_MODULE_STEM_{32,64}` or `LOCAL_MODULE_PATH_{32,64}`, then _32 executable will override the _64 executable in `/system/bin`. 
Building Multilib Modules

- **Dex-preopt:**
  - By default Multilib build generate both 32-bit and 64-bit odex files for the boot image and any Java libraries.
  - For APKs, by default odex files are generated only for the primary 64-bit arch.
    - If the app can be launched in both 32-bit and 64-bit processes, then set `LOCAL_MULTILIB := both` to make sure both 32-bit and 64-bit odex files are generated.
    - `LOCAL_MULTILIB := both` also include both 32-bit and 64-bit JNI libraries in the build, if the app has any.
Building Multilib Modules

- Generated sources:
  - In Multilib, intermediate generated source files will be required by both 32-bit and 64-bit builds.
  - Legacy $(local-intermediates-dir) and $(intermediates-dir-for) variables do not work reliably. Use $(local-generated-sources-dir) and $(generated-sources-dir-for) instead.
  - If a source file is generated to the new dedicated directory and picked up by LOCAL_GENERATED_SOURCES, it is built for both 32-bit and 64-bit build.
Multilib Examples From AOSP

- `device/htc/flounder/Boardconfig.mk`
  - Device config example

- `system/core/debuggerd/Android.mk`
  - Local or Module scoped build variables example
- **64_32 device config: Flounder device/htc/flounder/BoardConfig.mk**
  - Set Primary, Secondary CPUs and supported ABIs
  - `TARGET_USES_64_BIT_BINDER` should be set even while doing a 32-bit only build for a 64-bit arch.
  - `TARGET_SUPPORTS_{64,32}_BIT_APPS`, target support 64-bit applications only.
Multilib Android Module: debuggerd

system/core/debuggerd/Android.mk

- LOCAL_SRC_FILES, common src
- LOCAL_SRC_FILES_*, arch specific src
- TARGET_IS_64_BIT, true if TARGET_ARCH is 64-bit i.e. {arm64, x86_64 or mips64}.
- LOCAL_MODULE_STEM_*, install executables at same location i.e. /system/bin with different names.
- LOCAL_MULTILIB, build module for both the archs.
References

- AOSP changelog
- [android-64] New variables and macros of make system in android 64/32-bit build
- Android Platform 64-bit Build Instructions