**Contents**

1. **What is Hexagon SDK?**
2. **What is CDSP?**
3. **What is RPC?**
4. **What is FastRPC?**
5. **What is IDL file?**
6. **Explain the Software architecture.**
7. **What is Calculator Example?**
8. **What is the folder structure in calculator example?**
9. **Where is IDL file?**
10. **Where are src files?**
11. **Whare are make files?**
12. **What is python walk thru script ? what it is doing? How to tun?**
13. **Individual commands explaination.. like make android, make hexaong, signing, pushing, running simulator,, running on target....**
14. **What are the libs/ executables generated? whare ar they generated?**
15. **How to collect logs?**
16. **Exaplain the logs.**

**Hexagon SDK**

The Hexagon SDK is a collection of code generation and performance analysis tools designed to help developers enhance the features and performance of audio, speech, video, imaging and computer vision software on devices powered by Snapdragon™ processors.

 The Hexagon SDK is also designed to enable high computational burdens on the CPU to be off-loaded into a heterogeneous computing environment with the use of shared remote code objects.

Qualcomm's Hexagon SDK enables you to run your code on the Hexagon DSPs. This is achieved by Computation Offload.

[**Computational offload**](file:///C:\Qualcomm\Hexagon_SDK\3.5.2\docs\Applications_Compute%20offload.html)**:**

Define your own API, implement that API for the Hexagon DSP, and call it directly from the application processor. This is accomplished using a Remote Procedure Call mechanism called [FastRPC](file:///C:\Qualcomm\Hexagon_SDK\3.5.2\docs\APIs_FastRPC.html) and [dynamic loading](file:///C:\Qualcomm\Hexagon_SDK\3.5.2\docs\APIs_Dynamic%20Loading.html)

**Sourec link:** <https://developer.qualcomm.com/software/hexagon-dsp-sdk>

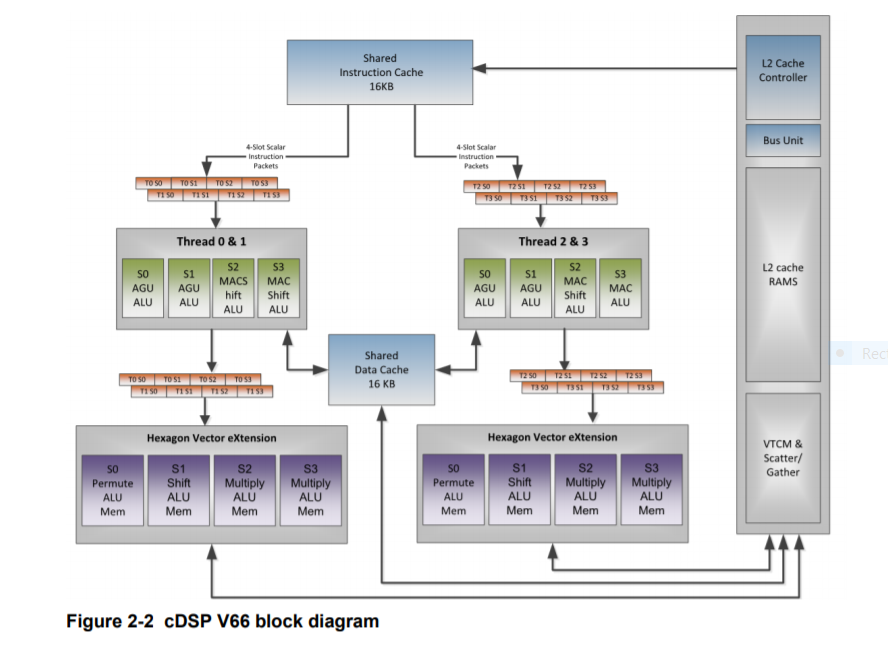
**What is CDSP?**

CDSP intended for compute-intensive tasks such as image processing, computer vision, and camera streaming.

Compared to the host CPU, the DSP typically runs at a lower clock speed but provides more parallelism opportunities at the instruction level. This makes the DSP a better alternative to the CPU for power consumption. As a result, it is preferable to offload as many large compute-intensive tasks as possible onto the DSP to reduce the overall power consumption of the device.

The CDSP has a number of shared resources: Cache memory, HVX, VTCM, hardware threads, memory busses, etc. In some cases, the overhead of sharing these resources while running applications concurrently causes overall performance to deteriorate.

This framework allows participating applications to run their workloads in batch (one after the other). Non-participating clients, i.e. clients not requesting serialization, can still run concurrently with a serialized batch.



**Source link:** <https://www.devever.net/~hl/f/80-VB419-108_Hexagon_DSP_User_Guide.pdf>

**What is RPC?**

Remote Procedure Call (RPC) is a [protocol](https://searchnetworking.techtarget.com/definition/protocol) that one program can use to request a service from a program located in another computer on a [network](https://searchnetworking.techtarget.com/definition/network) without having to understand the network's details.

RPC is used to call other processes on the remote systems like a local system. A procedure call is also sometimes known as a function call or a subroutine call.

RPC uses the [client-server](https://searchnetworking.techtarget.com/definition/client-server) model. The requesting program is a client, and the service-providing program is the [server](https://whatis.techtarget.com/definition/server). Like a regular or local procedure call, an RPC is a [synchronous](https://whatis.techtarget.com/definition/synchronous) operation requiring the requesting program to be suspended until the results of the remote procedure are returned.

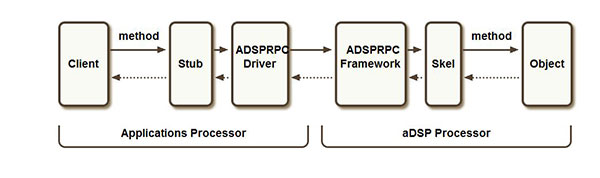
**Source link:** <https://searchapparchitecture.techtarget.com/definition/Remote-Procedure-Call-RPC?amp=1>

**What is FastRPC?**

The FastRPC framework allows for clients to transparently make remote method invocations between applications and DSP processors.

**FastRPC architecture:**

The below diagram depicts invocation of a single method where the client and object reside on different processors.



**Source link**: <https://developer.qualcomm.com/blog/how-use-fastrpc-offload-cpu-qualcomm-hexagon-dsp>

|  |  |
| --- | --- |
| Client | User mode process that initiates the remote invocation |
| Stub |  |
| ADSPRPC Driver | ADSPRPC kernel driver that receives the remote invocation,  queues them up and then waits for the response after  signaling the remote side |
| ADSPRPC Framework | ADSPRPC framework dequeues the messages from the  queue and dispatches them for processing |
| Skel | Auto generated code that takes care of unmarshaling  Parameters |
| Object | Method implementation |

**Source link**: <file:///C:/Qualcomm/Hexagon_SDK/3.5.2/docs/APIs_FastRPC.html>

**What is IDL file?**

The interface definition language (IDL) -- the specification language used to describe a software component's application programming interface (API) -- is commonly used in Remote Procedure Call software. In this case, IDL provides a bridge between the machines at either end of the link that may be using different operating systems (OSes) and computer languages.

**Calculator.idl**

This file present below path.

C:\Qualcomm\Hexagon\_SDK\3.5.2\examples\common\calculator\inc

#include "AEEStdDef.idl"

interface calculator {

long sum(in sequence<long> vec, rout long long res);

};

The “in” and “rout” types in IDL have special meanings:

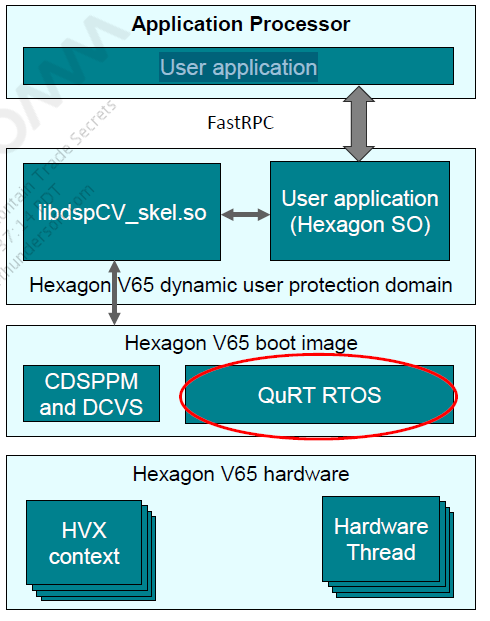
🡪 Declaring a buffer as “in” would result in below behavior:

* Apps flushes the cache for the buffer
* Makes RPC call
* DSP invalidates the cache for the buffer before reading it

🡪Declaring a buffer as “rout” would result in below behavior:

* Makes RPC call
* DSP flushes the cache after writing to the buffer
* Apps invalidates the cache for the buffer before reading it

**Explain Software architecture?**



The Qualcomm QURT RTOS provides MMU with the translation information on how to convert the Virtual Addresses into Physical Addresses. This is done by populating VA-PA translation entries in the MMU TLB buffers table. The number of TLB entries is limited and TLB table cannot represent the entire memory available to the Hexagon DSP processor.

**Source link:** [**file:///C:/Qualcomm/Hexagon\_SDK/3.5.2/docs/Memory%20Management.html**](file:///C:/Qualcomm/Hexagon_SDK/3.5.2/docs/Memory%20Management.html)

CDSPPM(CDSP power management);

**Source link :** <file:///C:/Qualcomm/Hexagon_SDK/3.5.2/docs/DSP%20Power%20&%20Perf.html>

**Source link:** <https://www.devever.net/~hl/f/80-VB419-108_Hexagon_DSP_User_Guide.pdf>

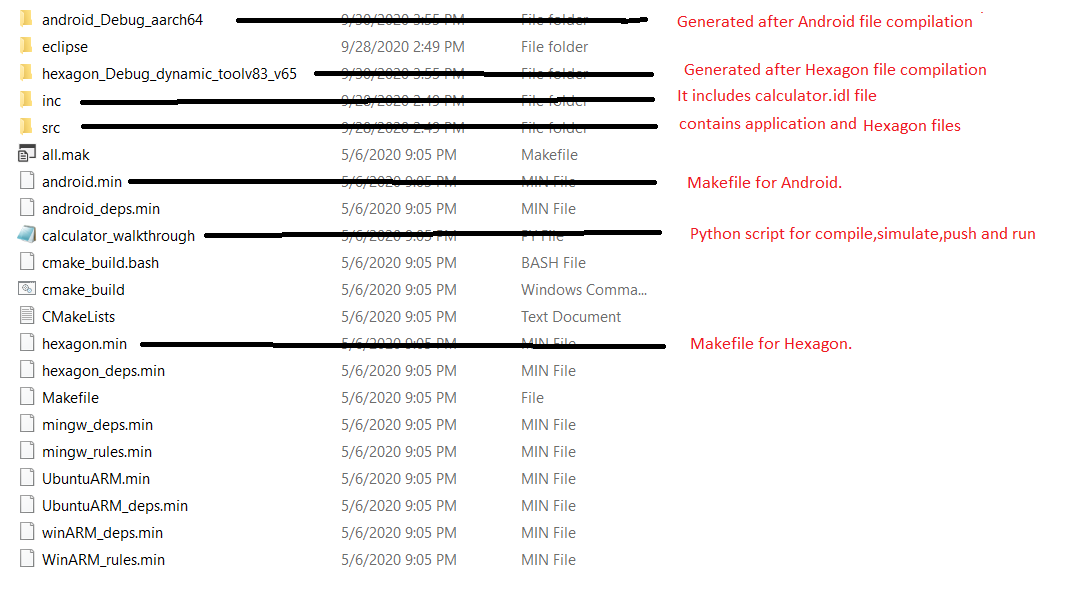
**What is Calculator Example?**

In this example the application's interface is inherited from “remote\_handle64”. This will allow the application to use the Open/Close functions of this interface to open remote sessions to the DSP the user can choose at run-time. The Open function takes a uri parameter as input where the user can specify the target DSP domain he wants the remote session to be connected to. For details on the exact values to be provided to this interface you can refer to the the function “calculator\_test()” in file “calculator\_test.c”. The Open function returns a handle to the remote session created which can then be used to call into ther other interface functions defined in the IDL.

Internally the RPC driver running in the apps processor will know which domain to execute the skel function in based on the remote handle.

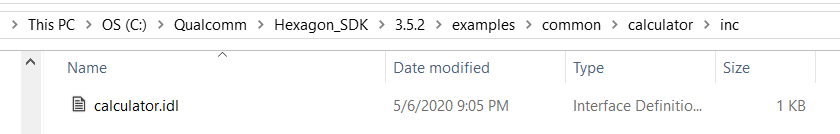
The calculator\_multi\_domains example illustrates this by taking the domain value as a run-time input. Based on this value, the example opens a remote session with the respective DSP and offloads the functions “sum” .

**What is the folder structure in calculator example?**



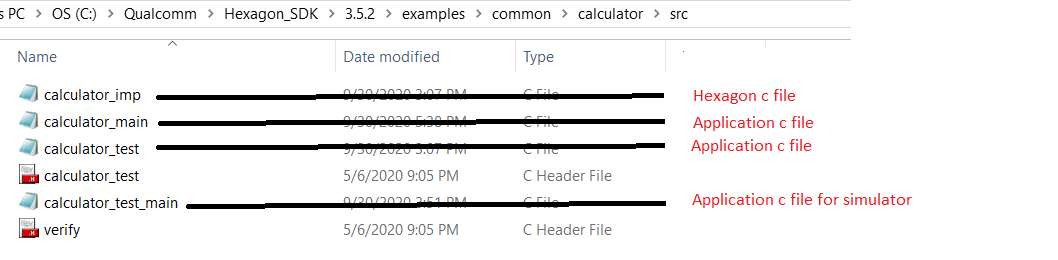
**Where is IDL file?**

IDL file present inC:\Qualcomm\Hexagon\_SDK\3.5.2\examples\common\calculator\inc



**Where are src files?**

src files present in C:\Qualcomm\Hexagon\_SDK\3.5.2\examples\common\calculator\src

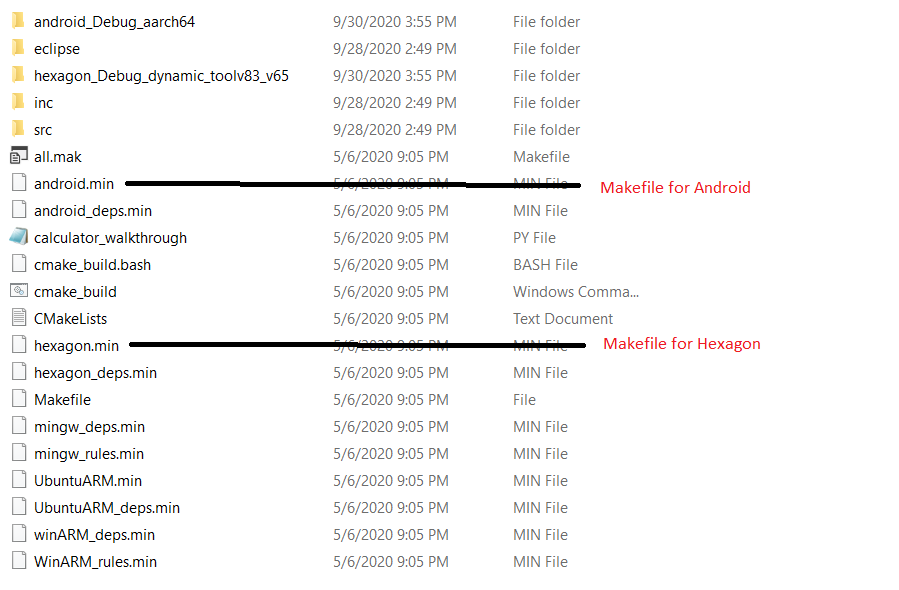


**What are makefiles?**

android.min 🡪 Makefile for Android

hexagon.min 🡪 Makefile for Hexagon

Makefiles are present in C:\Qualcomm\Hexagon\_SDK\3.5.2\examples\common\calculator



**What is python walkthrough script? What it is doing? How to run?**

walk-through script is a step by step guide to building, loading and executing the calculator example on Android.

**Command for run python walk-through script is:**

python calculator\_walkthrough.py -T sdm845 -D cdsp

**Walk-through script does:**

1) Compile Application test app

2) Compile Hexagon Side library and run in simulator

3) Signing of the device

4) Push the test app and libs into the device. into specific paths

App: vendor/bin/

Hexagon lib: /vendor/lib/rfsa/dsp/sdk/

5) Run the application

**Explain individual commands ?**

**1)compile Android test app:**

Make clean for Android:

make tree\_clean V=android\_Debug\_aarch64 CDSP\_FLAG=1 VERBOSE=1

Make Android:

make tree V=android\_Debug\_aarch64 CDSP\_FLAG=1 VERBOSE=1

**2)Compile Hexagon Side library and run in simulator:**

Make clean for Hexagon:

make tree\_clean V=hexagon\_Debug\_dynamic\_toolv83\_v65 VERBOSE=1

Make Hexagon:

make tree V=hexagon\_Debug\_dynamic\_toolv83\_v65 VERBOSE=1

Simulaor command:

C:/Qualcomm/Hexagon\_SDK/3.5.2/tools/HEXAGON\_Tools/8.3.07/Tools/bin/hexagon-sim -mv65 --simulated\_returnval --usefs hexagon\_Debug\_dynamic\_toolv83\_v65 --pmu\_statsfile hexagon\_Debug\_dynamic\_toolv83\_v65/pmu\_stats.txt --l2tcm\_base 0xd800 hexagon\_Debug\_dynamic\_toolv83\_v65/calculator\_q --

3) **Signing of the device**

--- Read serial number from device ---

--- Generate testsig ---

python C:\Qualcomm\Hexagon\_SDK\3.5.2/tools/elfsigner/elfsigner.py -t 0xa7f88a2f -o C:\Qualcomm\Hexagon\_SDK\3.5.2/tools/elfsigner/testsigs

Logging to C:\Qualcomm\Hexagon\_SDK\3.5.2\tools\elfsigner\testsigs\Elfsigner\_log.txt

Attention:

Use of this tool is conditioned upon your compliance with Qualcomm

Technologies'(and its affiliates') license terms and conditions;

including, without limitations, such terms and conditions addressing

the use of such tools with open source software.

Agree? [y/n]:

y

Signing a file may take up to 3 minutes due to network connectivity. Please wait patiently.

------------------------------------------------------------

Signing complete! Output saved at C:\Qualcomm\Hexagon\_SDK\3.5.2\tools\elfsigner\testsigs\testsig-0xa7f88a2f.so

--- Testsig generated sucessfully ---

4) **Push the test app and libs into the device. into specific paths**

--- Push Test Signature ---

Creating ” /vendor/lib/rfsa/dsp/testsig” path in device and pushing testsig-0xa7f88a2f.so in ” /vendor/lib/rfsa/dsp/testsig” path

adb shell mkdir -p /vendor/lib/rfsa/dsp/testsig

adb push C:\Qualcomm\Hexagon\_SDK\3.5.2/tools/elfsigner/testsigs/testsig-0xa7f88a2f.so /vendor/lib/rfsa/dsp/testsig

---- Push Android components ----

Creating “/vendor/bin” in device and pushing calculator in “/vendor/bin” path.

adb shell mkdir -p /vendor/bin

adb push C:\Qualcomm\Hexagon\_SDK\3.5.2/examples/common/calculator/android\_Debug\_aarch64/ship/calculator /vendor/bin

pushing libcalculator.so in “/vendor/lib64/” path

adb push C:\Qualcomm\Hexagon\_SDK\3.5.2/examples/common/calculator/android\_Debug\_aarch64/ship/libcalculator.so /vendor/lib64/

---- Push Hexagon Components ----

Creating “/vendor/lib/rfsa/dsp/sdk/” path in device and pushing libcalculator\_skel.so in ” /vendor/lib/rfsa/dsp/sdk/ ” path.

adb shell mkdir -p /vendor/lib/rfsa/dsp/sdk/

adb push C:\Qualcomm\Hexagon\_SDK\3.5.2/examples/common/calculator/hexagon\_Debug\_dynamic\_toolv83\_v65/ship/libcalculator\_skel.so /vendor/lib/rfsa/dsp/sdk/

---- Direct dsp messages to logcat ---

adb shell "echo 0x1f > /vendor/lib/rfsa/dsp/sdk/calculator.farf"

5) **Run the application**

---- Run Calculator Example Locally on Android ----

adb shell export LD\_LIBRARY\_PATH="/vendor/lib64/;" /vendor/bin/calculator 1 1000

Output:

---Starting calculator test

---Allocate 4000 bytes from ION heap

---Creating sequence of numbers from 0 to 999

---Compute sum locally

Sum = 499500

---Success

---- Run Calculator Example on CDSP ----

adb shell export LD\_LIBRARY\_PATH=/vendor/lib64/:$LD\_LIBRARY\_PATH ADSP\_LIBRARY\_PATH="/vendor/lib/rfsa/dsp/sdk\;/vendor/lib/rfsa/dsp/testsig;" /vendor/bin/calculator 0 1000

Output:

---Starting calculator test

---Allocate 4000 bytes from ION heap

---Creating sequence of numbers from 0 to 999

---Compute sum on the DSP

---Sum = 499500

---Success

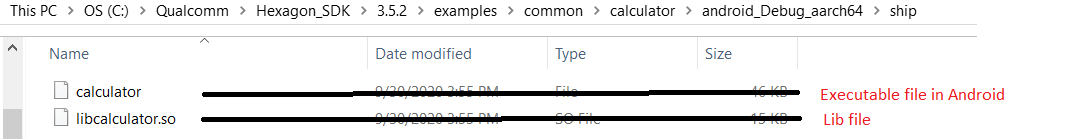
**What are the libs/ executables generated? Where they generated?**

In **android.min**

Generated library is: **libcalculator.so**

Generated Executable is: **calculator**

These are generated in C:\Qualcomm\Hexagon\_SDK\3.5.2\examples\common\calculator\android\_Debug\_aarch64\ship

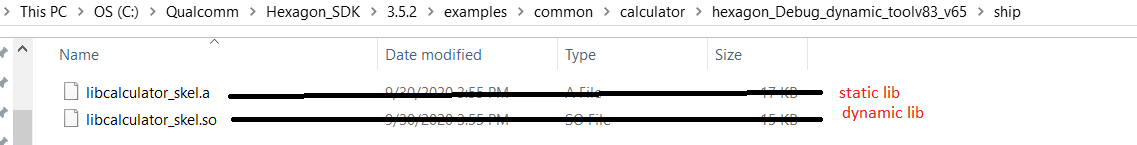
****

**In hexagon.min**

Generated library is: **libcalculator\_skel.so ,** **libcalculator\_skel.a**

Generated Executable is: **calculator\_q**

These are generated in C:\Qualcomm\Hexagon\_SDK\3.5.2\examples\common\calculator\hexagon\_Debug\_dynamic\_toolv83\_v65\ship



**How to collect logs?**

adb logcat -b all –c // clear previous logs

adb logcat -b all > log.txt // collect logs in to log.txt file.

**Explain the logs?**

calculator\_imp.c:19:0x10ea:7: CDSP: =============== DSP: sum result 499500 ===============

This log is present in calculator\_imp.c and it will display sum result of n numbers.

Where n is the input to command line arguments.

**Exaplain android.min (Android Makefile)?**

ifeq ($(CDSP\_FLAG), 1)

LIB\_DSPRPC = libcdsprpc --->If CDSP is 1,we are taking lilibcdsprpc library

DEFINES += CDSP

else ifeq ($(MDSP\_FLAG), 1)

LIB\_DSPRPC = libmdsprpc

DEFINES += MDSP

else ifeq ($(SLPI\_FLAG), 1)

LIB\_DSPRPC = libsdsprpc

DEFINES += SLPI

else

LIB\_DSPRPC = libadsprpc

DEFINES += ADSP

endif

$(info \*\*\*\*\*\*\*\*\*\*\*\* LIB=$(LIB\_DSPRPC) \*\*\*\*\*\*\*\*\*\*\*\*)

# stub library

BUILD\_DLLS += libcalculator ---->Generating library name as libcalculator

libcalculator\_QAICIDLS += inc/calculator ---->Including IDL files into libcalculator\_QAICIDLS

libcalculator\_C\_SRCS += $V/calculator\_stub ---->Including source files into libcalculator\_C\_SRCS

libcalculator\_DLLS += $(LIB\_DSPRPC) ---->Including CDSP lib to libcalculator\_DLLS

# stand-alone calculator executable

BUILD\_EXES += calculator --->Generating executable file as calculator

calculator\_QAICIDLS += inc/calculator --->Including IDL files into calculator\_QAICIDLS

calculator\_C\_SRCS += src/calculator\_main src/calculator\_test --->Including android source files into calculator\_C\_SRCS

calculator\_LIBS += rpcmem --->Including rpcmem into calculator\_LIBS

calculator\_LD\_FLAGS += -llog -ldl

calculator\_DEFINES += VERIFY\_PRINT\_ERROR

# copy final build products to the ship directory

BUILD\_COPIES = \

$(DLLS) \

$(EXES) \

$(LIBS) \

$(SHIP\_DIR)/ ; ---->all generated libraries will be in this path

**Exaplain hexagon.min (Hexagon Makefile)?**

# This builds the skel library

BUILD\_LIBS += libcalculator\_skel --->Generating static library name as libcalculator\_skel

# only build the shared object if dynamic option specified in the variant

ifeq (1,$(V\_dynamic))

BUILD\_DLLS = libcalculator\_skel --->Generating dynamic library name as libcalculator\_skel

endif

libcalculator\_skel\_QAICIDLS = inc/calculator --->Including IDL files into libcalculator\_skel\_QAICIDLS

libcalculator\_skel\_C\_SRCS += $V/calculator\_skel --->Including source file into libcalculator\_skel\_C\_SRCS

libcalculator\_skel.C\_SRCS = src/calculator\_imp.c --->Including Hexagon source files into libcalculator\_skel\_C\_SRCS

# quality test

BUILD\_QEXES += calculator\_q ---->Generaring executable file as calculator\_q

calculator\_q\_C\_SRCS = src/calculator\_test\_main src/calculator\_test src/calculator\_imp --->Including Android & Hexagon source files to calculator\_q\_C\_SRCS(simulator test)

calculator\_q\_LIBS = rtld rpcmem test\_util atomic

# copy final build products to the ship directory

BUILD\_COPIES = \

$(DLLS) \

$(EXES) \

$(LIBS) \

$(SHIP\_DIR)/ ;