Unveiling the Mysteries of Hexagon QDSP6 JTAG

A Journey into Advanced Theoretical Reverse Engineering

Alisa Esage
Zero Day Engineering Research & Training
Black Hat Asia 2025, Singapore

About me

Alisa Esage Shevchenko

- Independent Hacker
- Founder of Zero Day Engineering
- Researcher of God Mode* since 1999

Zero Day Engineering

research & training

* gaming term

About this talk

What is Hexagon?

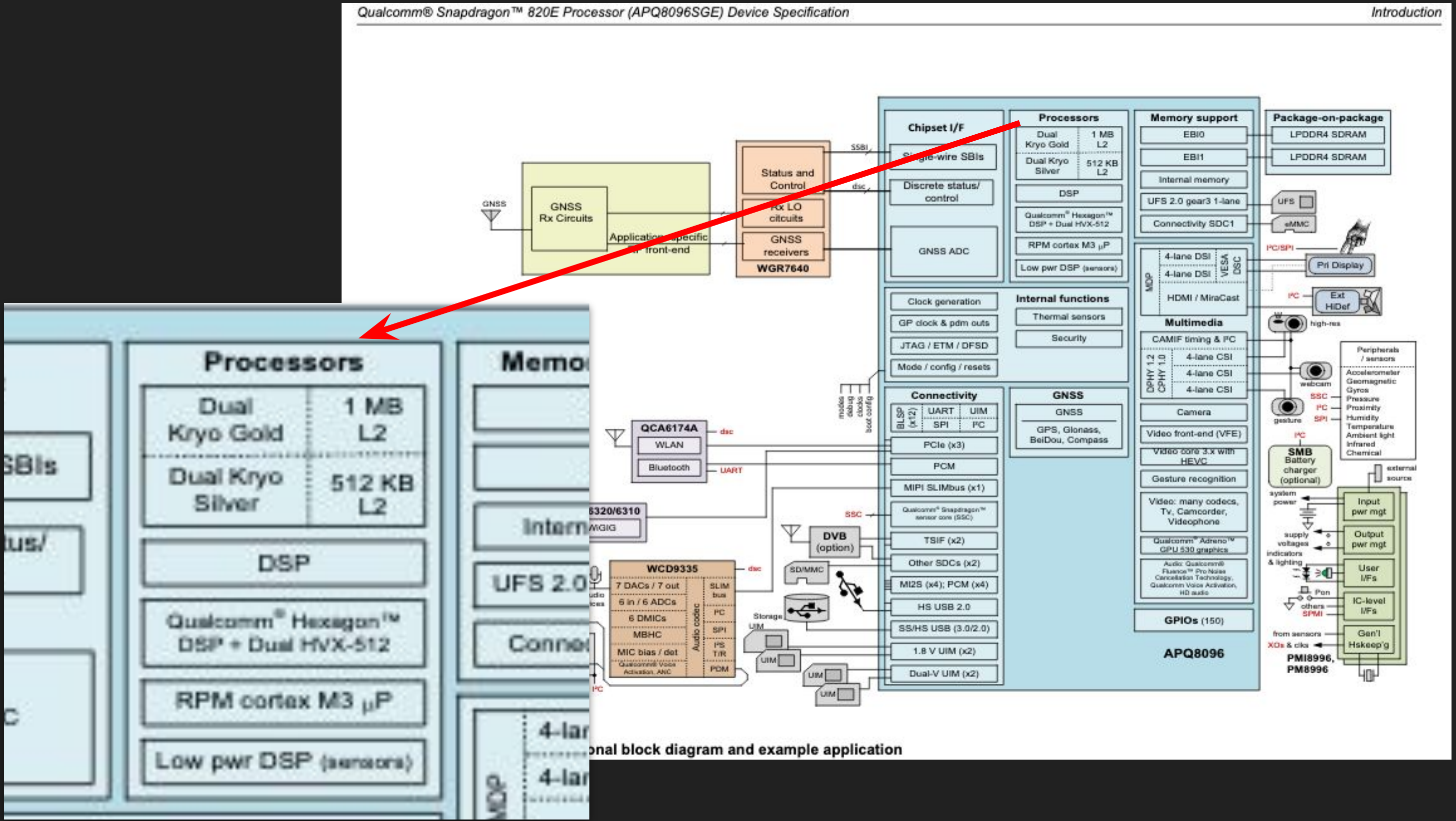
- Qualcomm Snapdragon & MDM chips
  - ~30% of smartphone market
  - Now entering **laptop market**
  - One or more specialized cores on the Snapdragon SoC are Hexagon cores
- Hexagon architecture
  - Proprietary by Qualcomm, secure
  - Mostly fw code behind Secure Boot
  - VLIW optimized for parallel execution, solid benchmarks
  - Started as DSP for specialized media workloads
  - Runs modem on Android MSM, aka baseband. Variety of attack vectors
  - Now, **NPU**

What is the problem with Hexagon?

- You can’t debug it

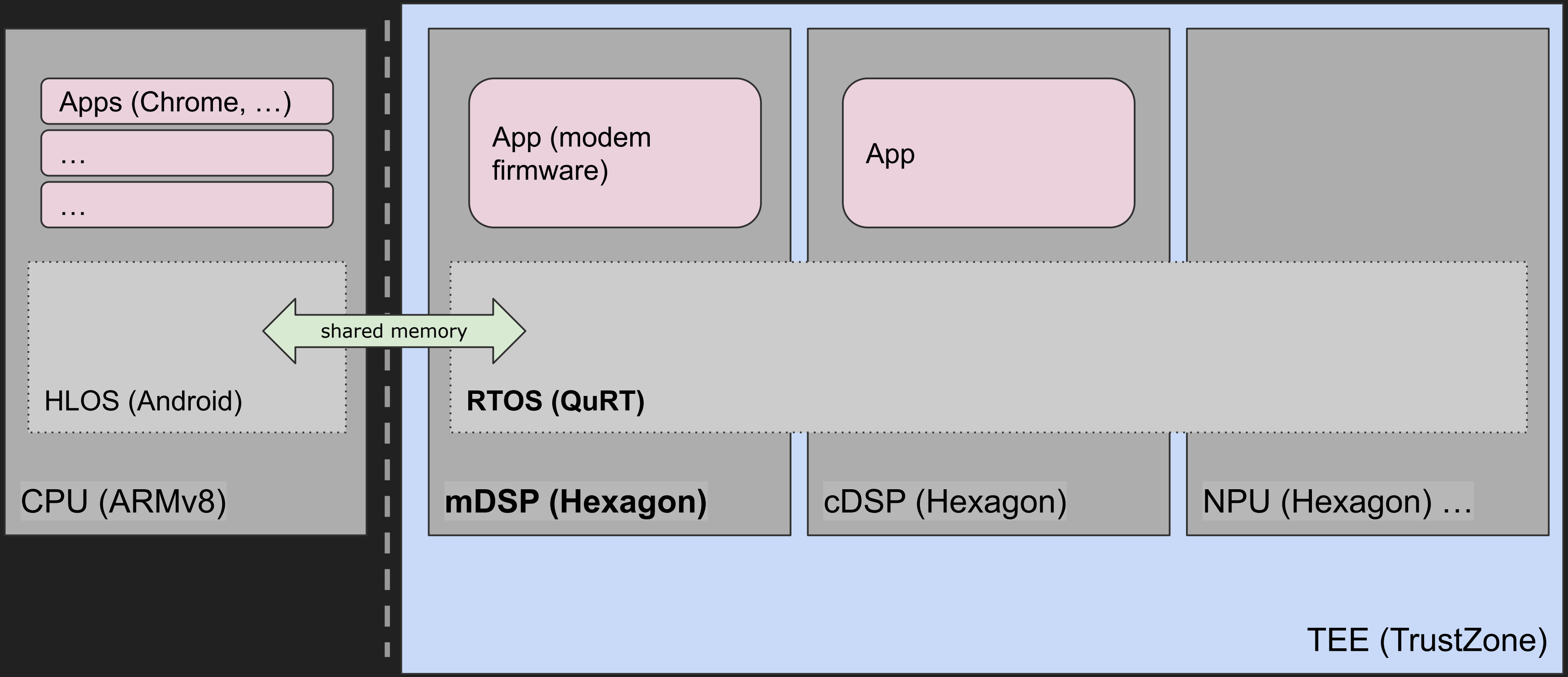
Intro

Hexagon and Snapdragon



https://developer.qualcomm.com/download/sd820e/qualcomm-snapdragon-820e-processor-apq8096sge-device-specification.pdf

Inside your smartphone (msm based)



Recap: Hexagon architecture

Hexagon: programmer's view

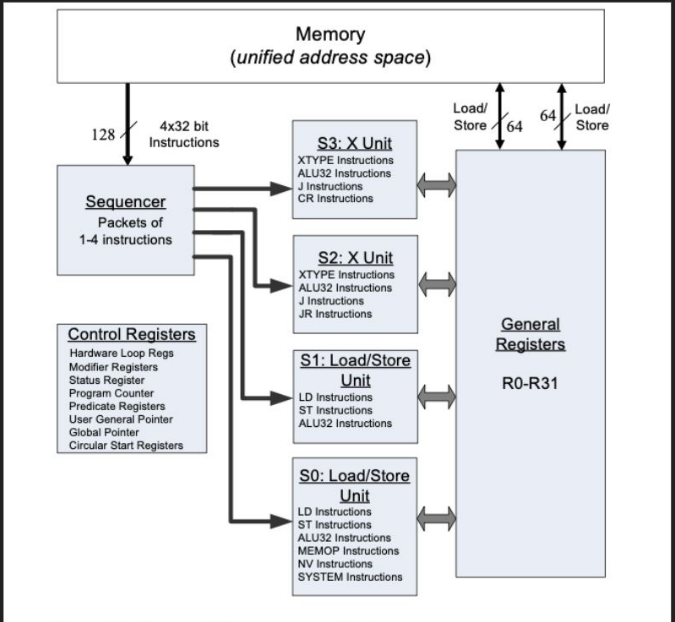
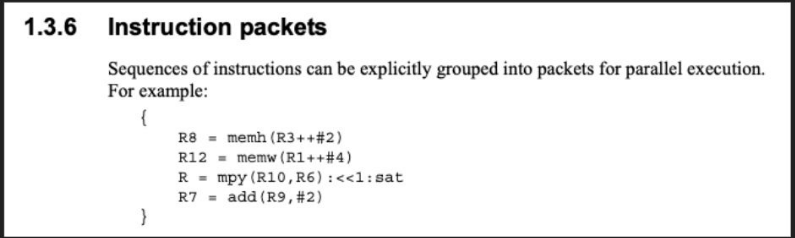


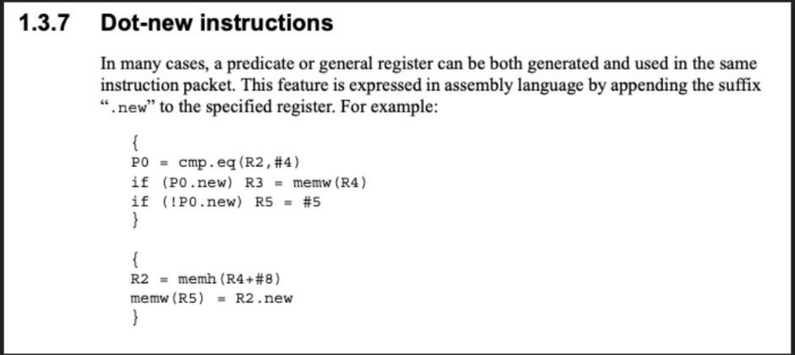
Figure 1-1 Hexagon V62 processor architecture



1.3.6 Instruction packets

Sequences of instructions can be explicitly grouped into packets for parallel execution. For example:

```
{
    R8 = memh(R3++#2)
    R12 = memw(R1++#4)
    R = mpy(R10,R6):<<1:sat
    R7 = add(R9,#2)
}
```



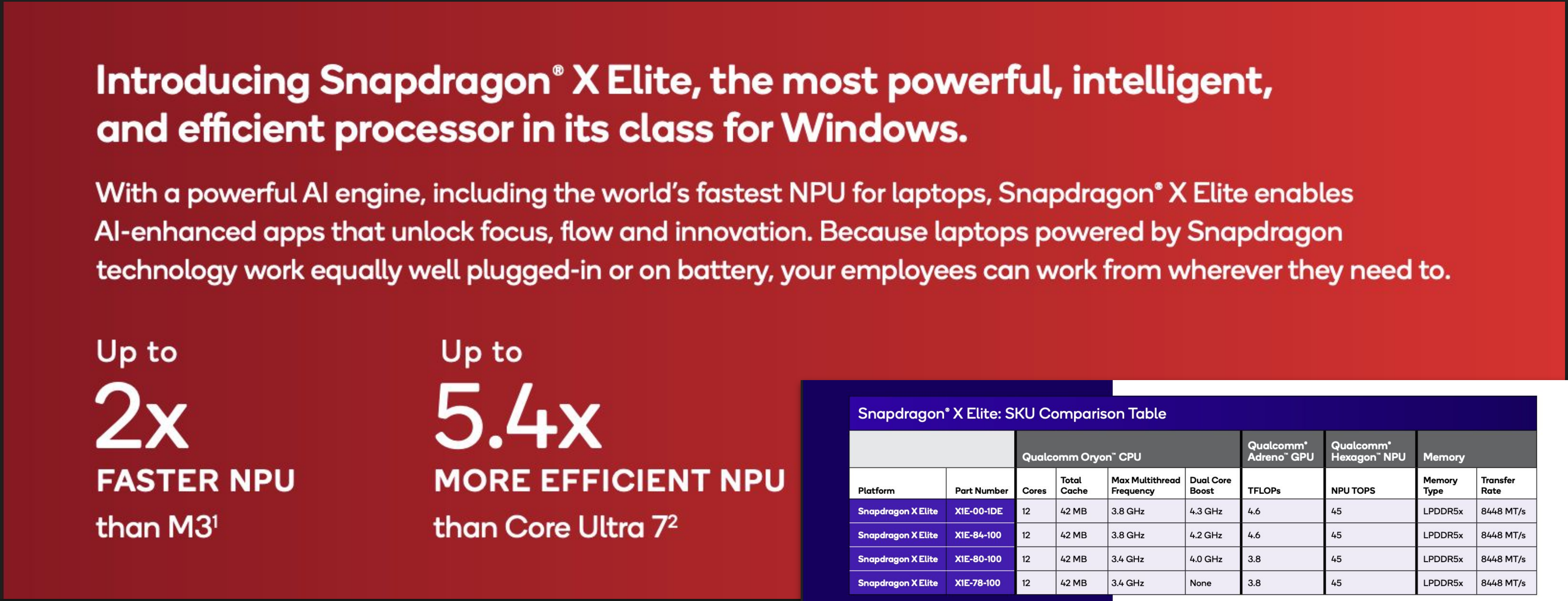
1.3.7 Dot-new instructions

In many cases, a predicate or general register can be both generated and used in the same instruction packet. This feature is expressed in assembly language by appending the suffix "`.new`" to the specified register. For example:

```
{
P0 = cmp.eq(R2,#4)
if (P0.new) R3 = memw(R4)
if (!P0.new) R5 = #5
}

{
R2 = memh(R4+#8)
memw(R5) = R2.new
}
```

[HEXAGONISA] https://developer.qualcomm.com/download/hexagon/hexagon-v62-programmers-reference-manual.pdf

Hexagon update

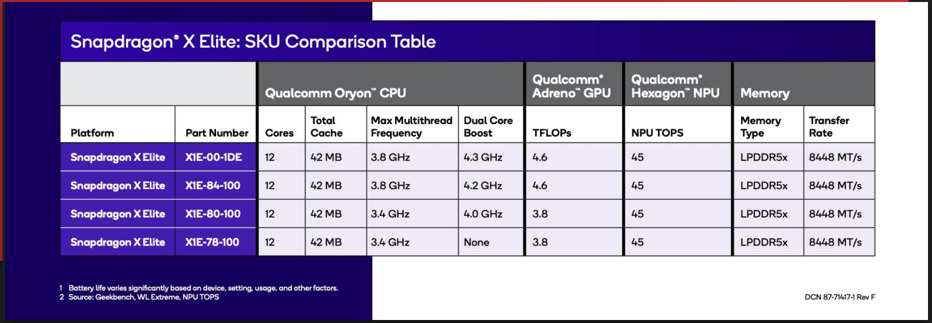


Introducing Snapdragon® X Elite, the most powerful, intelligent, and efficient processor in its class for Windows.

With a powerful AI engine, including the world's fastest NPU for laptops, Snapdragon® X Elite enables AI-enhanced apps that unlock focus, flow and innovation. Because laptops powered by Snapdragon technology work equally well plugged-in or on battery, your employees can work from wherever they need to.

Up to **2x** **FASTER NPU** than M3[1]

Up to **5.4x** **MORE EFFICIENT NPU** than Core Ultra 7[2]



Snapdragon® X Elite: SKU Comparison Table

		Qualcomm Oryon™ CPU				Qualcomm® Adreno™ GPU	Qualcomm® Hexagon™ NPU	Memory	
Platform	Part Number	Cores	Total Cache	Max Multithread Frequency	Dual Core Boost	TFLOPs	NPU TOPS	Memory Type	Transfer Rate
Snapdragon X Elite	X1E-00-1DE	12	42 MB	3.8 GHz	4.3 GHz	4.6	45	LPDDR5x	8448 MT/s
Snapdragon X Elite	X1E-84-100	12	42 MB	3.8 GHz	4.2 GHz	4.6	45	LPDDR5x	8448 MT/s
Snapdragon X Elite	X1E-80-100	12	42 MB	3.4 GHz	4.0 GHz	3.8	45	LPDDR5x	8448 MT/s
Snapdragon X Elite	X1E-78-100	12	42 MB	3.4 GHz	None	3.8	45	LPDDR5x	8448 MT/s

1 Battery life varies significantly based on device, setting, usage, and other factors.
2 Source: Geekbench, WL Extreme, NPU TOPS

DCN 87-71417-1 Rev F

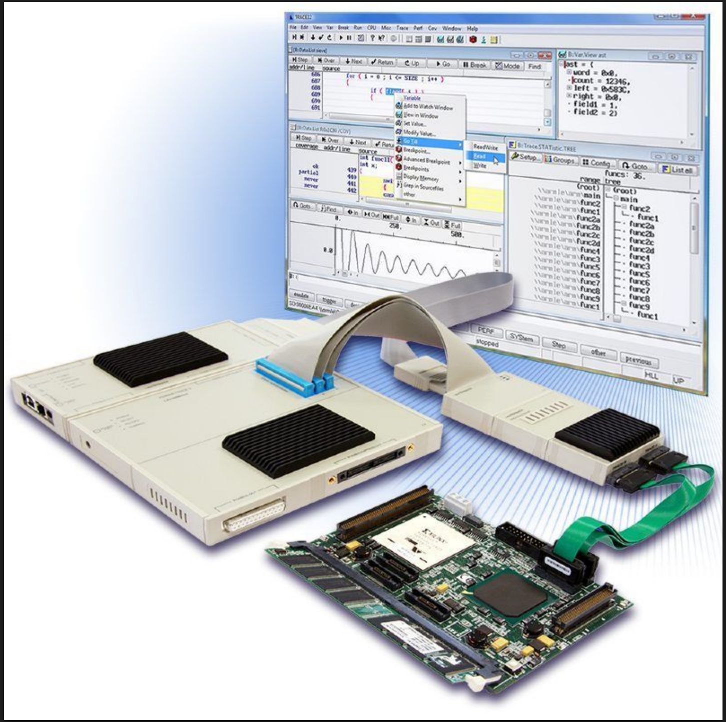
How do they debug Hexagon cores?

Hardware debugger

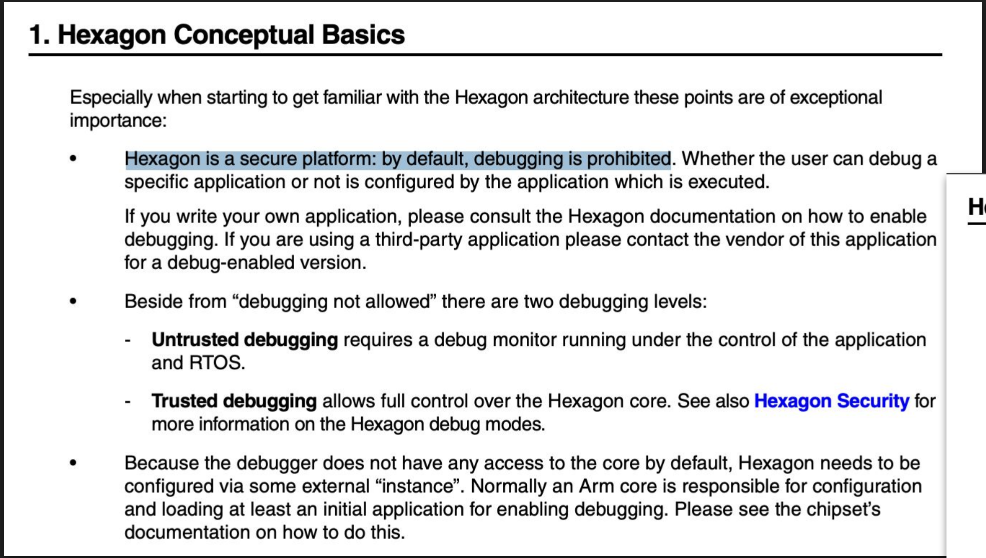
- Lauterbach TRACE32 (JTAG/Coresight)
  - 3rd party product, endorsed by Qualcomm
  - **Requires Qualcomm "partner enrollment" level support to configure debugging (impossible)**
  - Not applicable to off-the-shelf devices
  - Expensive

Software debugger

- Doesn't exist
  - Code that runs on Hexagon arch is heavily proprietary and undocumented, you are not supposed to know about it, let alone debug it
- Engineer your own gdb server, inject via software vulnerability exploit primitives
  - **DIY** reports in the past - a lot of impressive effort
  - But limited, unreliable & unsustainable in use
- Hexagon emulator/simulator are available
  - You can write high-level app code in Hexagon SDK and "debug" it on simulator, no problem with that
  - Mostly useless for deep security research



Trace32 User's Manual is pessimistic…



1. Hexagon Conceptual Basics

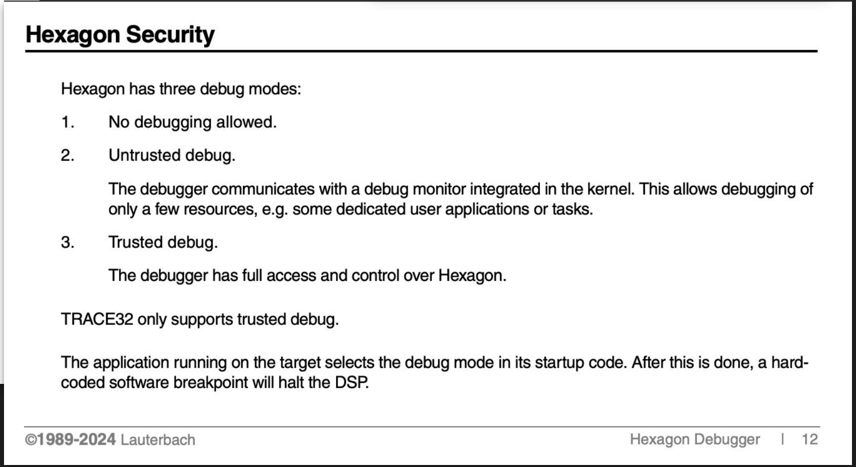
Especially when starting to get familiar with the Hexagon architecture these points are of exceptional importance:

- Hexagon is a secure platform: by default, debugging is prohibited. Whether the user can debug a specific application or not is configured by the application which is executed.

  If you write your own application, please consult the Hexagon documentation on how to enable debugging. If you are using a third-party application please contact the vendor of this application for a debug-enabled version.

- Beside from "debugging not allowed" there are two debugging levels:
  - **Untrusted debugging** requires a debug monitor running under the control of the application and RTOS.
  - **Trusted debugging** allows full control over the Hexagon core. See also **Hexagon Security** for more information on the Hexagon debug modes.

- Because the debugger does not have any access to the core by default, Hexagon needs to be configured via some external "instance". Normally an Arm core is responsible for configuration and loading at least an initial application for enabling debugging. Please see the chipset's documentation on how to do this.



Hexagon Security

Hexagon has three debug modes:

1. No debugging allowed.
2. Untrusted debug.

   The debugger communicates with a debug monitor integrated in the kernel. This allows debugging of only a few resources, e.g. some dedicated user applications or tasks.

3. Trusted debug.

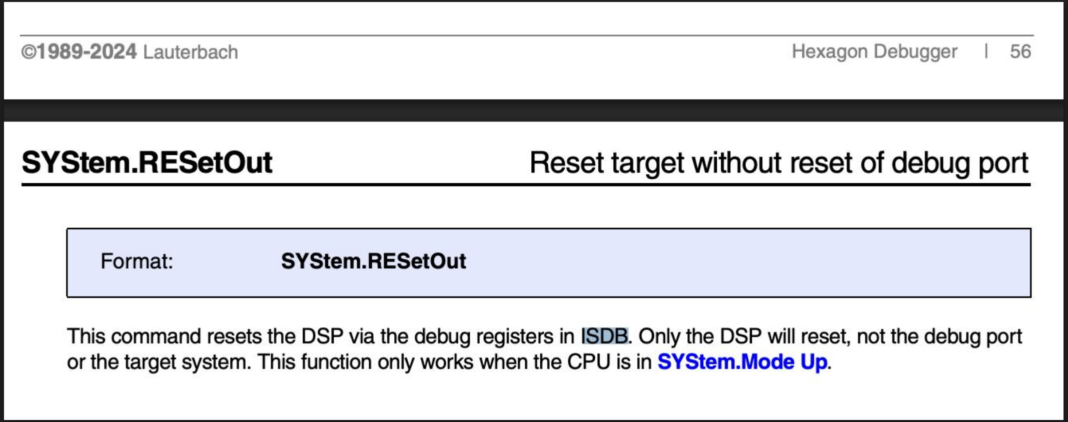
   The debugger has full access and control over Hexagon.

TRACE32 only supports trusted debug.

The application running on the target selects the debug mode in its startup code. After this is done, a hard-coded software breakpoint will halt the DSP.

©1989-2024 Lauterbach Hexagon Debugger | 12

Wait, what is this?

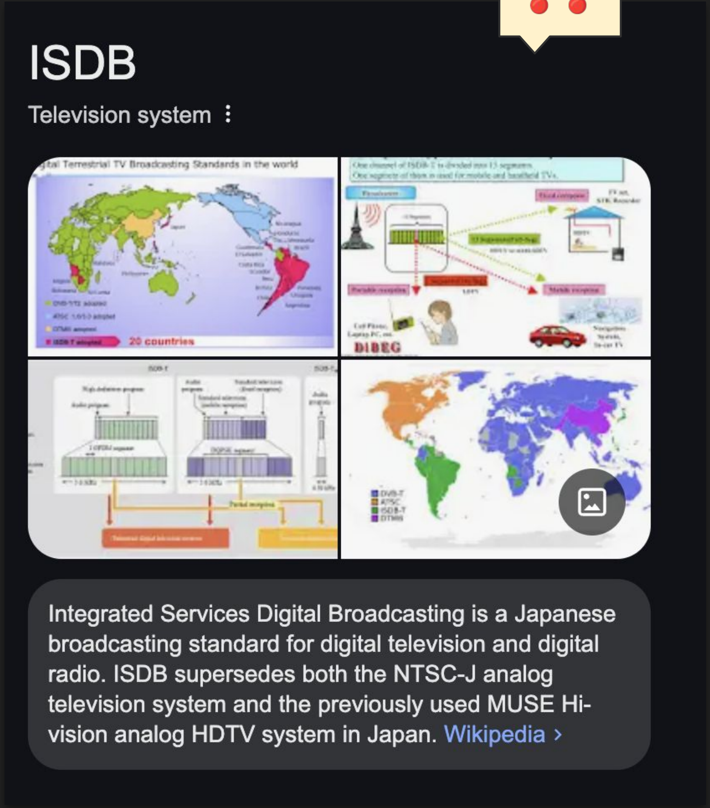


©1989-2024 Lauterbach Hexagon Debugger | 56

SYStem.RESetOut Reset target without reset of debug port

Format:	**SYStem.RESetOut**

This command resets the DSP via the debug registers in ISDB. Only the DSP will reset, not the debug port or the target system. This function only works when the CPU is in **SYStem.Mode Up**.

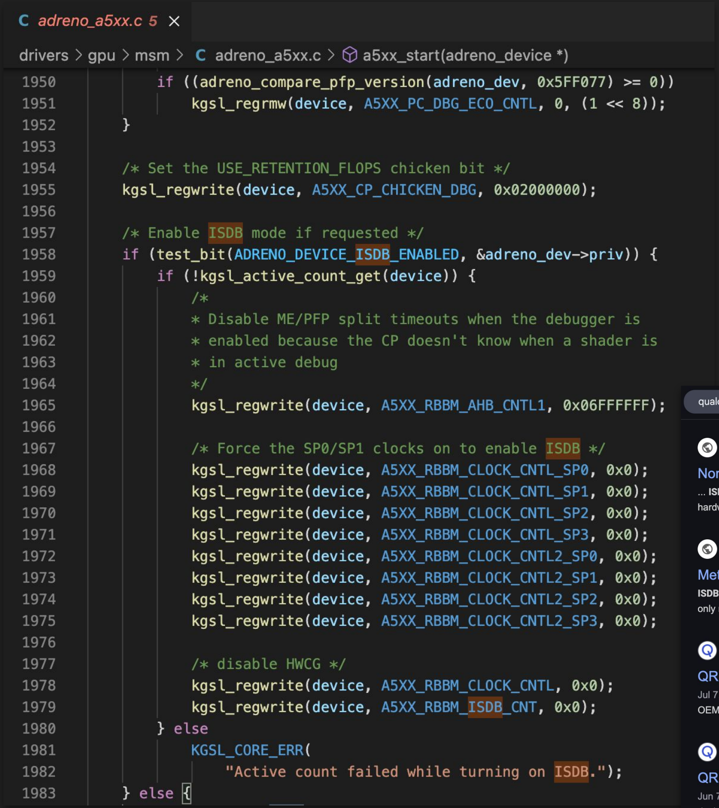


ISDB

Television system

Integrated Services Digital Broadcasting is a Japanese broadcasting standard for digital television and digital radio. ISDB supersedes both the NTSC-J analog television system and the previously used MUSE Hi-vision analog HDTV system in Japan. Wikipedia ›

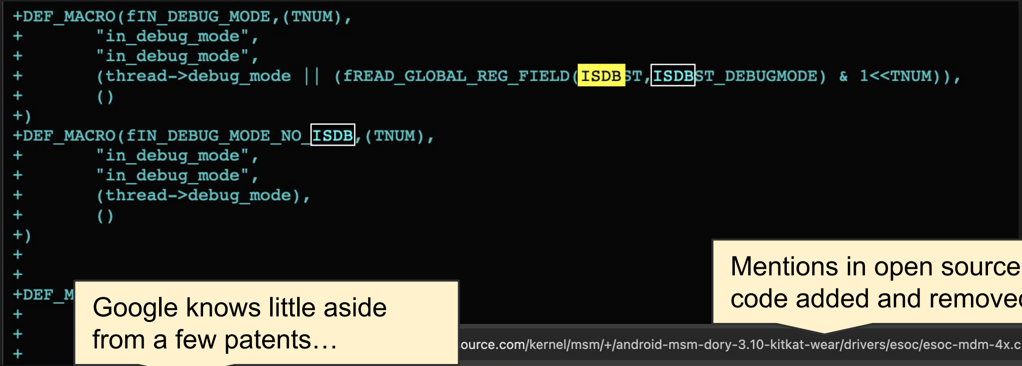
Start researching, mystery builds up…



C adreno_a5xx.c 5 ×

drivers > gpu > msm > C adreno_a5xx.c > a5xx_start(adreno_device *)

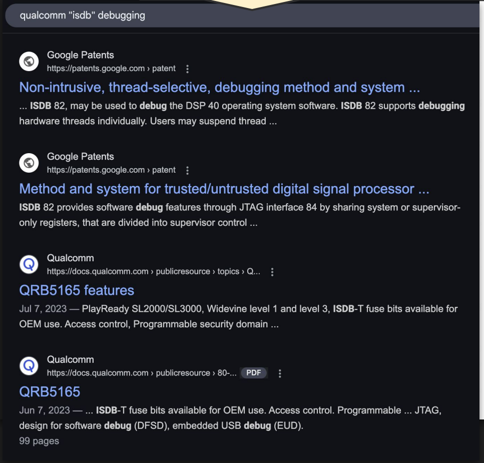
```
        if ((adreno_compare_pfp_version(adreno_dev, 0x5FF077) >= 0))
            kgsl_regrmw(device, A5XX_PC_DBG_ECO_CNTL, 0, (1 << 8));
    }

    /* Set the USE_RETENTION_FLOPS chicken bit */
    kgsl_regwrite(device, A5XX_CP_CHICKEN_DBG, 0x02000000);

    /* Enable ISDB mode if requested */
    if (test_bit(ADRENO_DEVICE_ISDB_ENABLED, &adreno_dev->priv)) {
        if (!kgsl_active_count_get(device)) {
            /*
             * Disable ME/PFP split timeouts when the debugger is
             * enabled because the CP doesn't know when a shader is
             * in active debug
             */
            kgsl_regwrite(device, A5XX_RBBM_AHB_CNTL1, 0x06FFFFFF);

            /* Force the SP0/SP1 clocks on to enable ISDB */
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL_SP0, 0x0);
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL_SP1, 0x0);
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL_SP2, 0x0);
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL_SP3, 0x0);
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL2_SP0, 0x0);
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL2_SP1, 0x0);
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL2_SP2, 0x0);
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL2_SP3, 0x0);

            /* disable HWCG */
            kgsl_regwrite(device, A5XX_RBBM_CLOCK_CNTL, 0x0);
            kgsl_regwrite(device, A5XX_RBBM_ISDB_CNT, 0x0);
        } else
            KGSL_CORE_ERR(
                "Active count failed while turning on ISDB.");
    } else {
```



```
+DEF_MACRO(fIN_DEBUG_MODE,(TNUM),
+        "in_debug_mode",
+        "in_debug_mode",
+        (thread->debug_mode || (fREAD_GLOBAL_REG_FIELD(ISDBST,ISDBST_DEBUGMODE) & 1<<TNUM)),
+        ()
+)
+DEF_MACRO(fIN_DEBUG_MODE_NO_ISDB,(TNUM),
+        "in_debug_mode",
+        "in_debug_mode",
+        (thread->debug_mode),
+        ()
+)
+
+
+DEF_M
+
+
+
```

Google knows little aside from a few patents…

Mentions in open source code added and removed…



qualcomm "isdb" debugging

Google Patents
https://patents.google.com › patent

Non-intrusive, thread-selective, debugging method and system ...

... **ISDB** 82, may be used to **debug** the DSP 40 operating system software. **ISDB** 82 supports **debugging** hardware threads individually. Users may suspend thread ...

Google Patents
https://patents.google.com › patent

Method and system for trusted/untrusted digital signal processor ...

ISDB 82 provides software **debug** features through JTAG interface 84 by sharing system or supervisor-only registers, that are divided into supervisor control ...

Qualcomm
https://docs.qualcomm.com › publicresource › topics › Q...

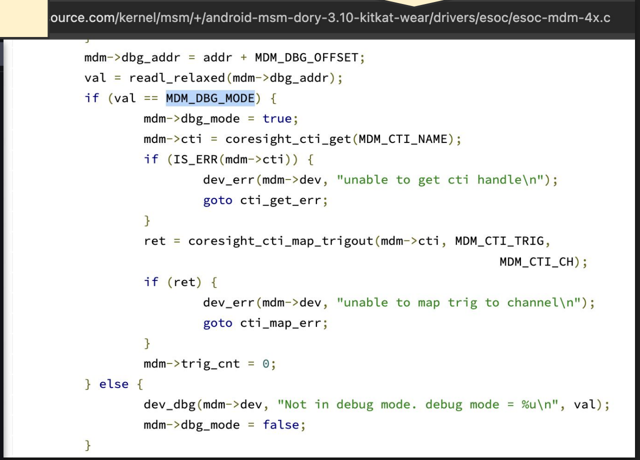
QRB5165 features

Jul 7, 2023 — PlayReady SL2000/SL3000, Widevine level 1 and level 3, **ISDB**-T fuse bits available for OEM use. Access control, Programmable security domain ...

Qualcomm
https://docs.qualcomm.com › publicresource › 80-... PDF

QRB5165

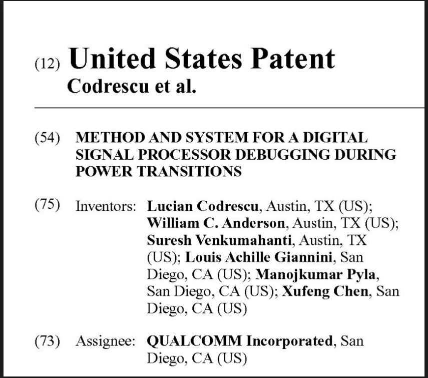
Jun 7, 2023 — ... **ISDB**-T fuse bits available for OEM use. Access control. Programmable ... JTAG, design for software **debug** (DFSD), embedded USB **debug** (EUD).
99 pages



ource.com/kernel/msm/+/android-msm-dory-3.10-kitkat-wear/drivers/esoc/esoc-mdm-4x.c

```
    mdm->dbg_addr = addr + MDM_DBG_OFFSET;
    val = readl_relaxed(mdm->dbg_addr);
    if (val == MDM_DBG_MODE) {
            mdm->dbg_mode = true;
            mdm->cti = coresight_cti_get(MDM_CTI_NAME);
            if (IS_ERR(mdm->cti)) {
                    dev_err(mdm->dev, "unable to get cti handle\n");
                    goto cti_get_err;
            }
            ret = coresight_cti_map_trigout(mdm->cti, MDM_CTI_TRIG,
                                                     MDM_CTI_CH);
            if (ret) {
                    dev_err(mdm->dev, "unable to map trig to channel\n");
                    goto cti_map_err;
            }
            mdm->trig_cnt = 0;
    } else {
            dev_dbg(mdm->dev, "Not in debug mode. debug mode = %u\n", val);
            mdm->dbg_mode = false;
    }
```

Patent documentation FTW

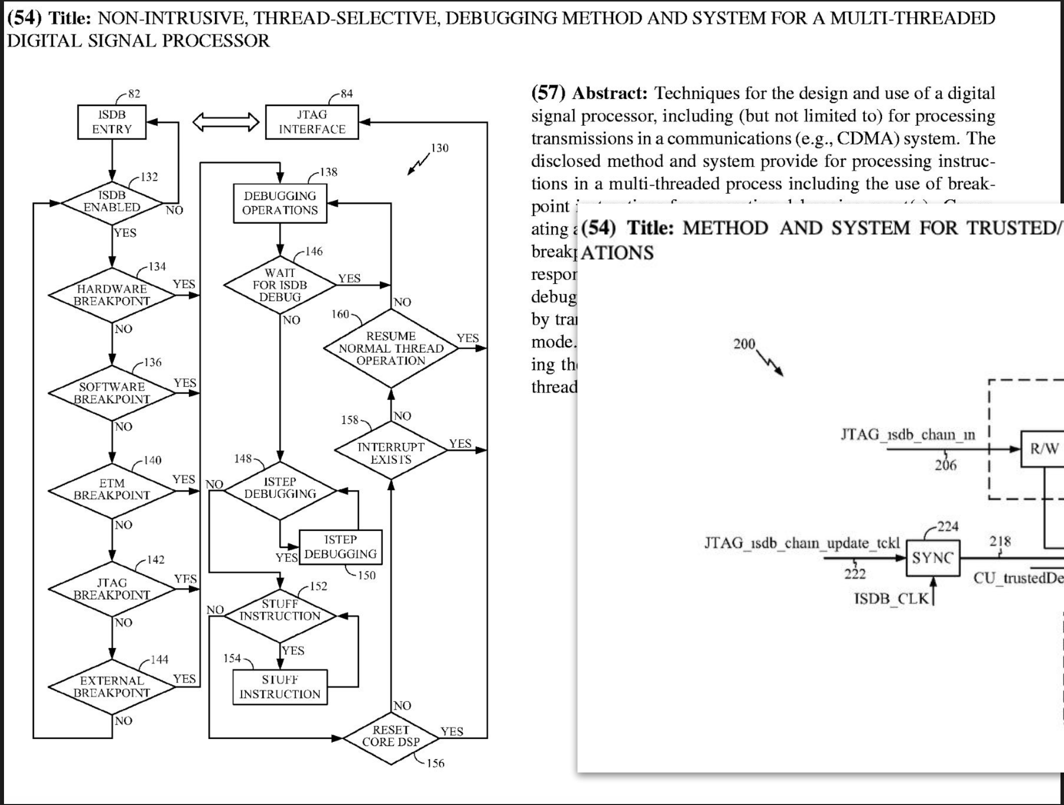


(12) **United States Patent**
Codrescu et al.

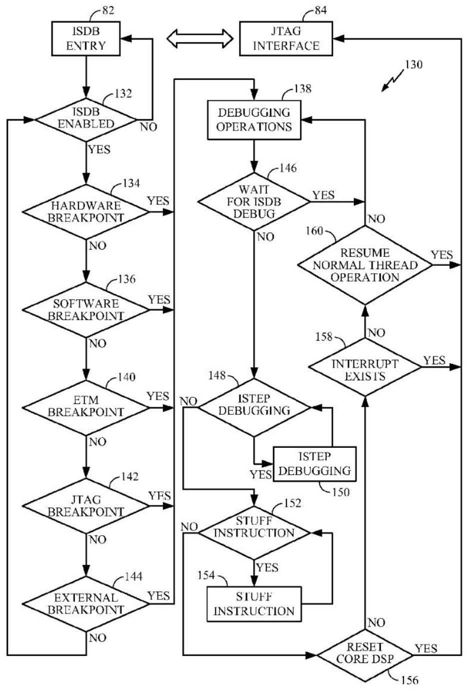
(54) **METHOD AND SYSTEM FOR A DIGITAL SIGNAL PROCESSOR DEBUGGING DURING POWER TRANSITIONS**

(75) Inventors: **Lucian Codrescu**, Austin, TX (US); **William C. Anderson**, Austin, TX (US); **Suresh Venkumahanti**, Austin, TX (US); **Louis Achille Giannini**, San Diego, CA (US); **Manojkumar Pyla**, San Diego, CA (US); **Xufeng Chen**, San Diego, CA (US)

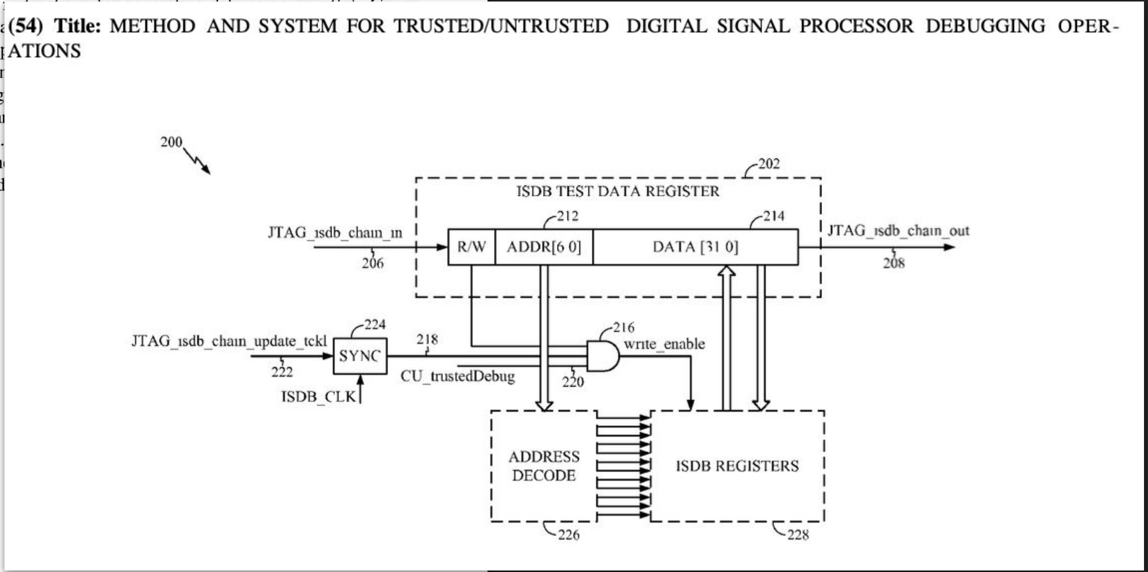
(73) Assignee: **QUALCOMM Incorporated**, San Diego, CA (US)



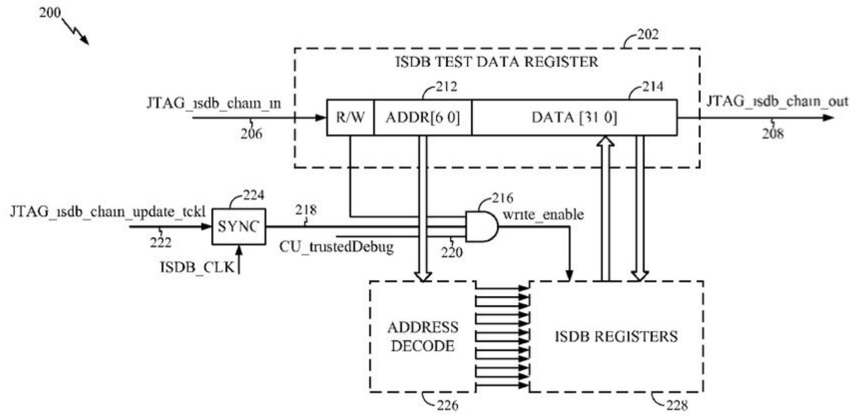
(54) Title: NON-INTRUSIVE, THREAD-SELECTIVE, DEBUGGING METHOD AND SYSTEM FOR A MULTI-THREADED DIGITAL SIGNAL PROCESSOR



(57) Abstract: Techniques for the design and use of a digital signal processor, including (but not limited to) for processing transmissions in a communications (e.g., CDMA) system. The disclosed method and system provide for processing instructions in a multi-threaded process including the use of breakpoint i
ating a
breakp
respon
debug
by tra
mode.
ing th
thread



(54) Title: METHOD AND SYSTEM FOR TRUSTED/UNTRUSTED DIGITAL SIGNAL PROCESSOR DEBUGGING OPERATIONS



Project card: RE Hexagon Debugging

Sources

- Patent documentation
- Qualcomm Programmer's Reference Manuals
- Open source code
- Datasheets

Methods

- OSINT
- Thinking
- Grepping QURT binaries for strings
- Open baseband firmware in IDA and close it

Funding

- This research project was partially sponsored by a company that chose to remain anonymous
- Findings approved for disclosure
- Thank you!

Results

- Qualcomm **ISDB system internals** revealed here for the first time
- Outlined basic prerequisites to **enable and operate debugging of Hexagon firmware**
- This talk will focus on the **core aspects** of the matter due to limited time and disclosure, a lot had to be left out
- Still a lot to uncover

Fast forward to findings >>>

Hexagon Debugging Internals

ISDB (In Silicone Debugger)

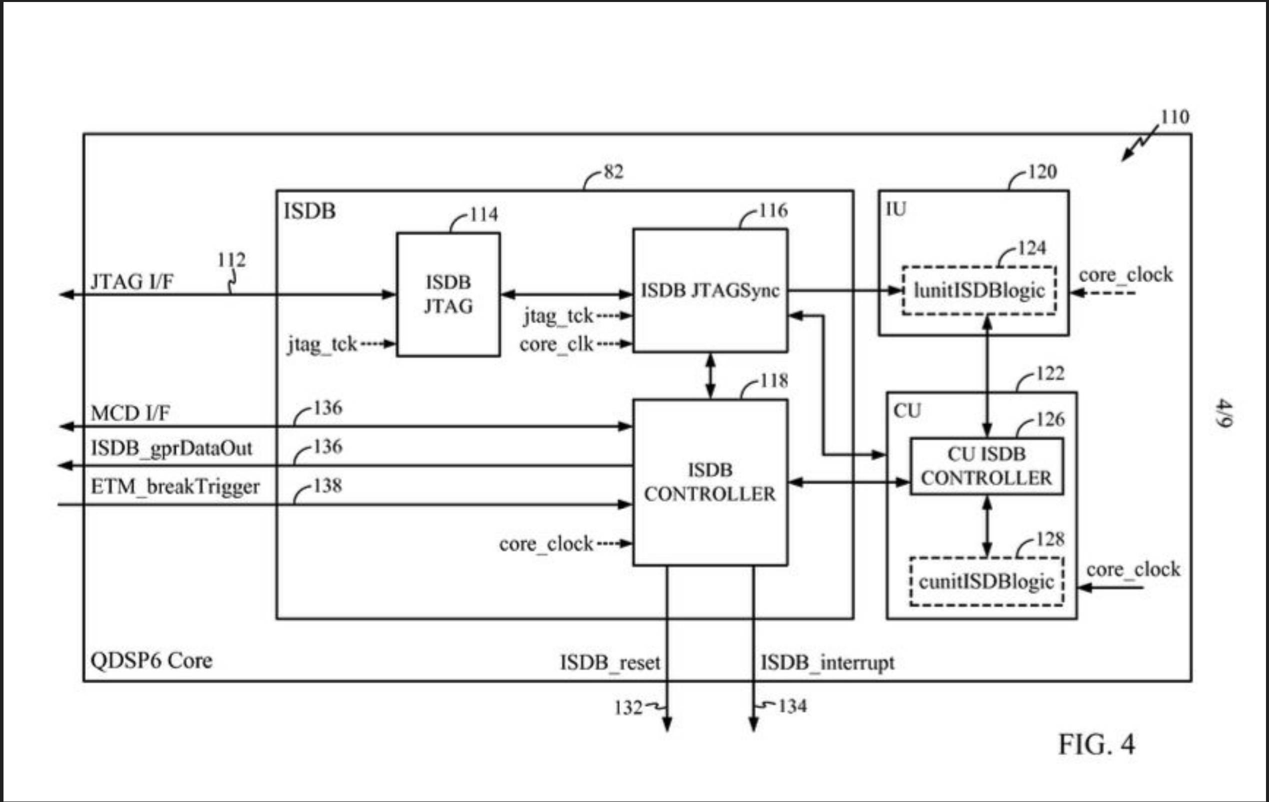
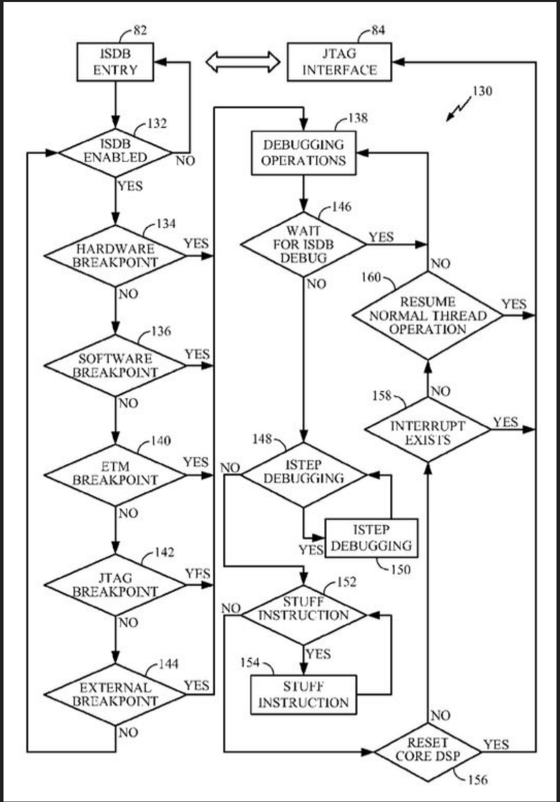


FIG. 4



Breakpoint processing circuitry

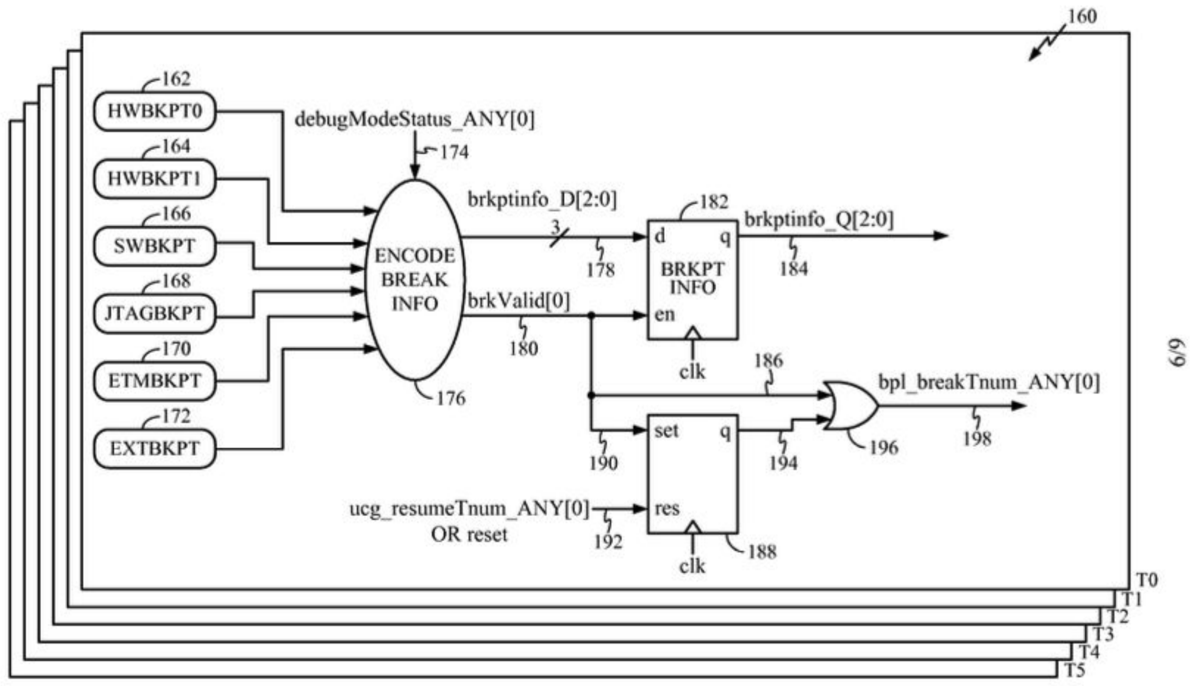
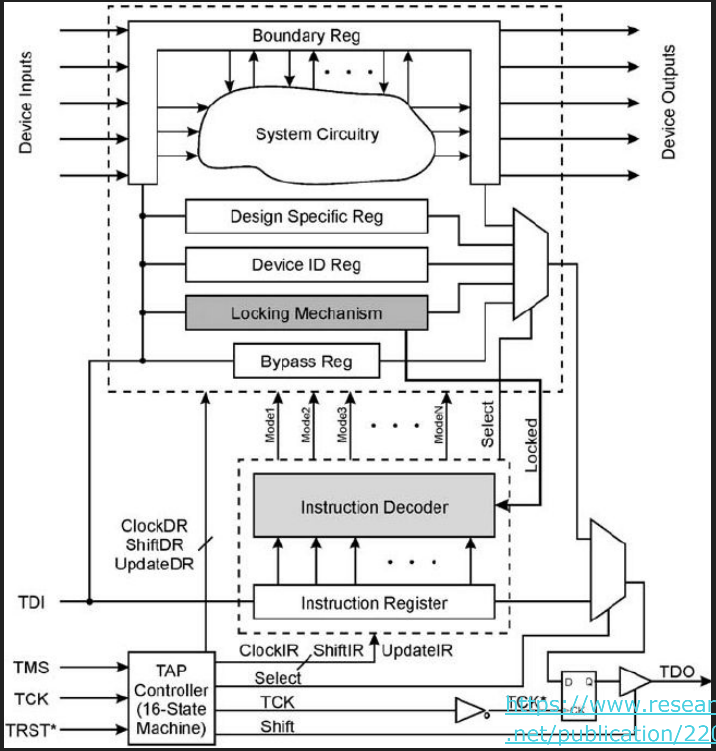


FIG. 6

Recap: JTAG IEEE 1149.1

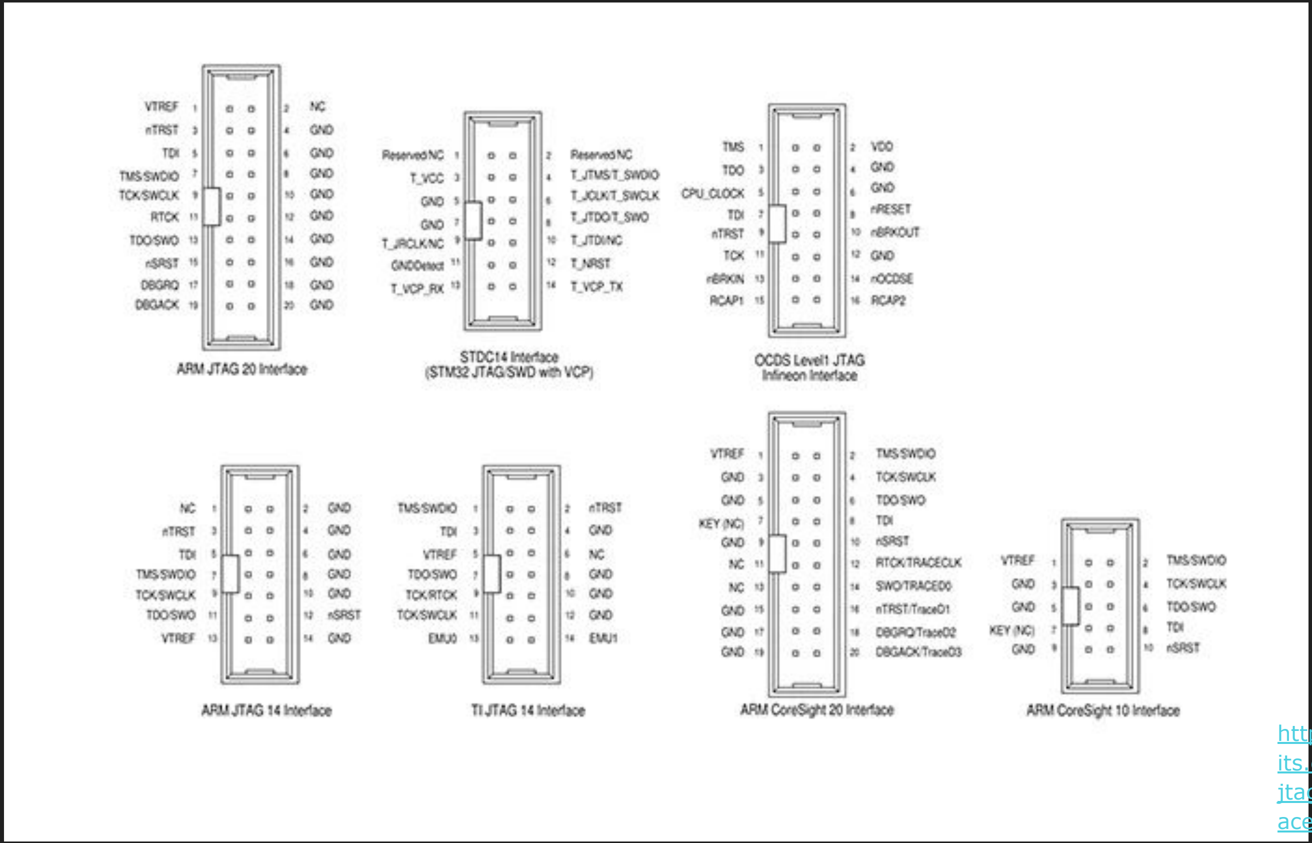
The standard

- Basic technology for testing microelectronic circuits
- Simple interface - serial pins
  - TDI (Test Data In), TDO (Test Data Out)
  - Test mode selection, clock, reset
- **Very powerful**
- **No access control**
- **No resource control**
- Most device vendors either don't care or rely on "security by obscurity" to hide JTAG port



https://www.researchgate.net/publication/220648926_Security_extension_for_IEEE_Std_11491

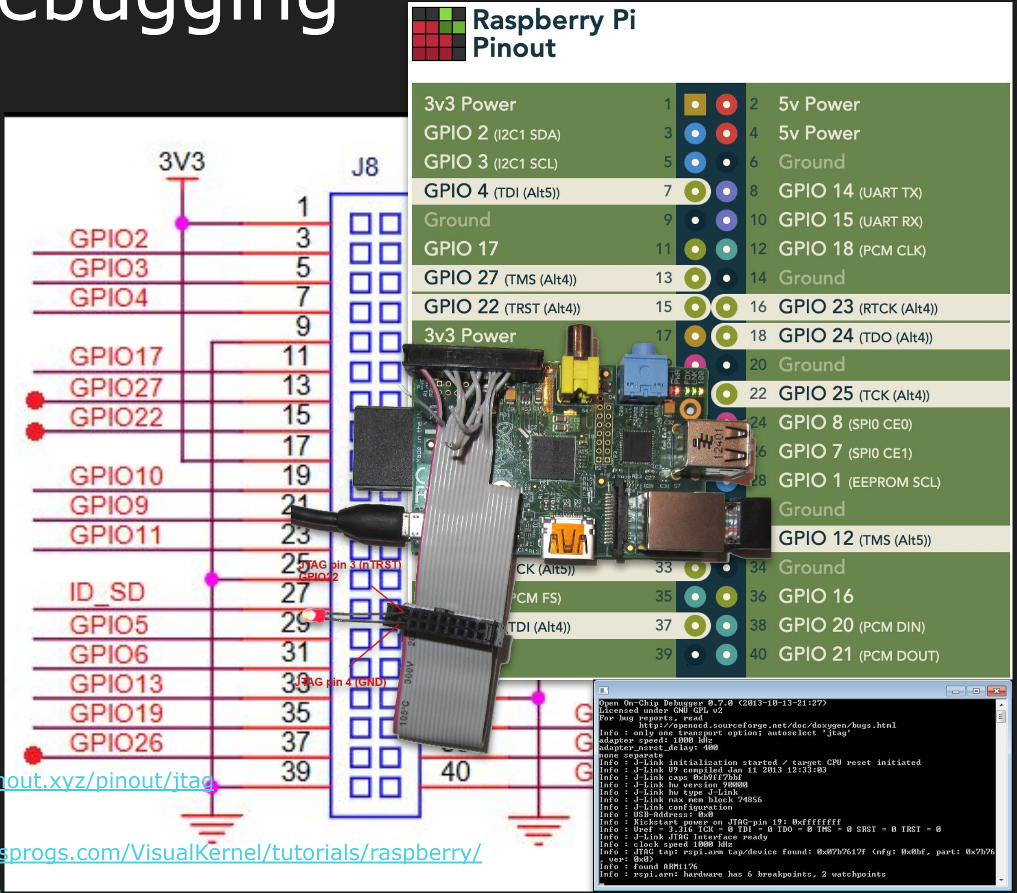
Extended JTAG pinouts



https://www.allaboutcircuits.com/technical-articles/jtag-connectors-and-interfaces/

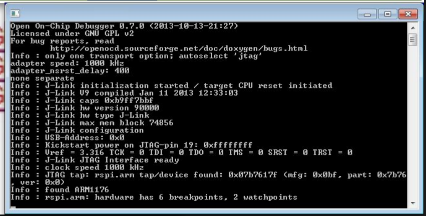
JTAG and software debugging

- Powerful primitives
  - Access to memory
  - Access to registers
  - Halt signal
- Software debugger engineering
  - Build standard debugging ops on JTAG hardware primitives
  - wrap in GUI/CLI/gdb
  - FTDI (USB-TTL) for wiring
- Example: tracing/single step
  - Halt signal + program counter register modification
- Example: breakpoint
  - Hardware bp: program the register
  - Software bp: inject the opcode
- **OpenOCD**

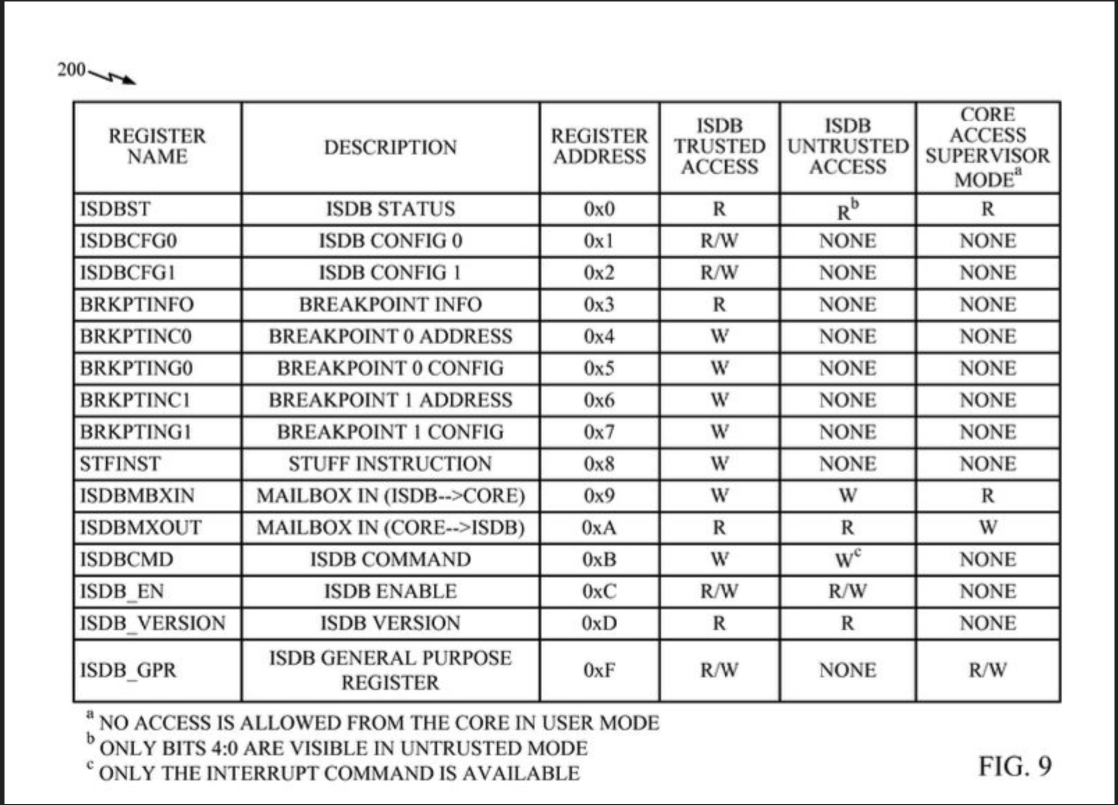

Raspberry Pi Pinout
3V3
J8
GPIO2
GPIO3
GPIO4
GPIO17
GPIO27
GPIO22
GPIO10
GPIO9
GPIO11
ID_SD
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
JTAG pin 3 (nTRST) GPIO22
JTAG pin 4 (GND)
3v3 Power 1 2 5v Power
GPIO 2 (I2C1 SDA) 3 4 5v Power
GPIO 3 (I2C1 SCL) 5 6 Ground
GPIO 4 (TDI (Alt5)) 7 8 GPIO 14 (UART TX)
Ground 9 10 GPIO 15 (UART RX)
GPIO 17 11 12 GPIO 18 (PCM CLK)
GPIO 27 (TMS (Alt4)) 13 14 Ground
GPIO 22 (TRST (Alt4)) 15 16 GPIO 23 (RTCK (Alt4))
3v3 Power 17 18 GPIO 24 (TDO (Alt4))
20 Ground
22 GPIO 25 (TCK (Alt4))
24 GPIO 8 (SPI0 CE0)
GPIO 7 (SPI0 CE1)
GPIO 1 (EEPROM SCL)
Ground
GPIO 12 (TMS (Alt5))
CK (Alt5)) 33 34 Ground
PCM FS) 35 36 GPIO 16
TDI (Alt4)) 37 38 GPIO 20 (PCM DIN)
39 40 GPIO 21 (PCM DOUT)

https://pinout.xyz/pinout/jtag

https://sysprogs.com/VisualKernel/tutorials/raspberry/jtagsetup/


```
Open On-Chip Debugger 0.7.0 (2013-10-13-21:27)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
adapter speed: 1000 kHz
adapter_nsrst_delay: 400
none separate
Info : J-Link initialization started / target CPU reset initiated
Info : J-Link V9 compiled Jan 11 2013 12:33:03
Info : J-Link caps 0xb9ff7bbf
Info : J-Link hw version 90000
Info : J-Link hw type J-Link
Info : J-Link max mem block 74856
Info : J-Link configuration
Info : USB-Address: 0x0
Info : Kickstart power on JTAG-pin 19: 0xffffffff
Info : Vref = 3.316 TCK = 0 TDI = 0 TDO = 0 TMS = 0 SRST = 0 TRST = 0
Info : J-Link JTAG Interface ready
Info : clock speed 1000 kHz
Info : JTAG tap: rspi.arm tap/device found: 0x07b7617f (mfg: 0x0bf, part: 0x7b76
, ver: 0x0)
Info : found ARM1176
Info : rspi.arm: hardware has 6 breakpoints, 2 watchpoints
```

ISDB Registers



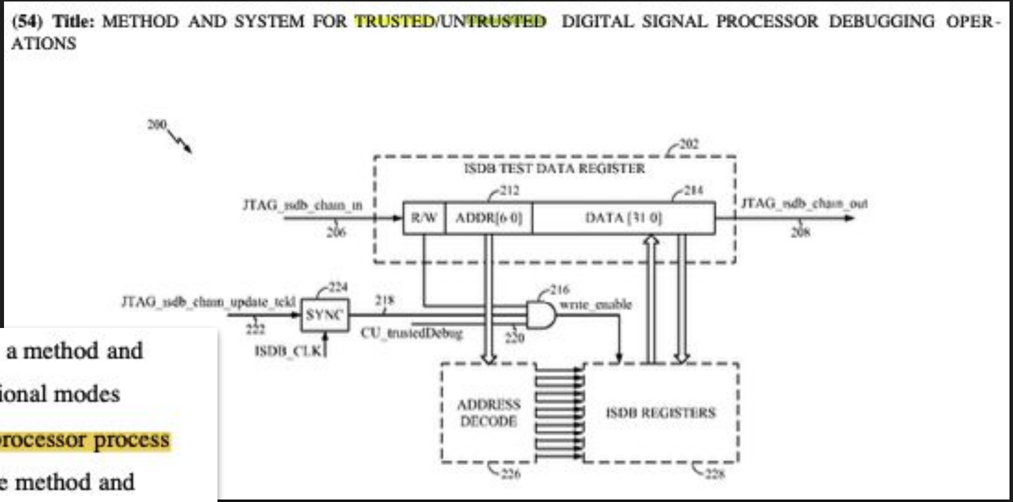
200

REGISTER NAME	DESCRIPTION	REGISTER ADDRESS	ISDB TRUSTED ACCESS	ISDB UNTRUSTED ACCESS	CORE ACCESS SUPERVISOR MODE[a]
ISDBST	ISDB STATUS	0x0	R	R[b]	R
ISDBCFG0	ISDB CONFIG 0	0x1	R/W	NONE	NONE
ISDBCFG1	ISDB CONFIG 1	0x2	R/W	NONE	NONE
BRKPTINFO	BREAKPOINT INFO	0x3	R	NONE	NONE
BRKPTINC0	BREAKPOINT 0 ADDRESS	0x4	W	NONE	NONE
BRKPTING0	BREAKPOINT 0 CONFIG	0x5	W	NONE	NONE
BRKPTINC1	BREAKPOINT 1 ADDRESS	0x6	W	NONE	NONE
BRKPTING1	BREAKPOINT 1 CONFIG	0x7	W	NONE	NONE
STFINST	STUFF INSTRUCTION	0x8	W	NONE	NONE
ISDBMBXIN	MAILBOX IN (ISDB-->CORE)	0x9	W	W	R
ISDBMXOUT	MAILBOX IN (CORE-->ISDB)	0xA	R	R	W
ISDBCMD	ISDB COMMAND	0xB	W	W[c]	NONE
ISDB_EN	ISDB ENABLE	0xC	R/W	R/W	NONE
ISDB_VERSION	ISDB VERSION	0xD	R	R	NONE
ISDB_GPR	ISDB GENERAL PURPOSE REGISTER	0xF	R/W	NONE	R/W

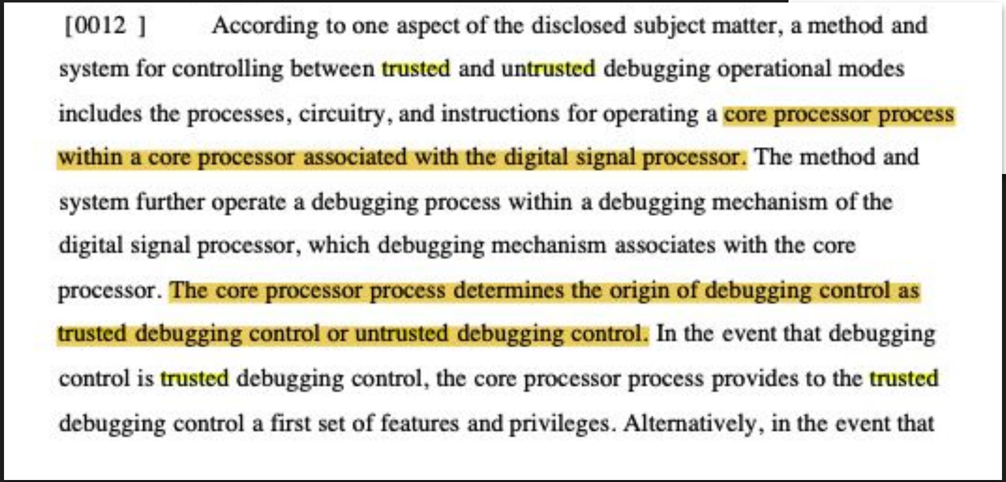
[a] NO ACCESS IS ALLOWED FROM THE CORE IN USER MODE
[b] ONLY BITS 4:0 ARE VISIBLE IN UNTRUSTED MODE
[c] ONLY THE INTERRUPT COMMAND IS AVAILABLE

FIG. 9

Trusted and Untrusted debugging mode

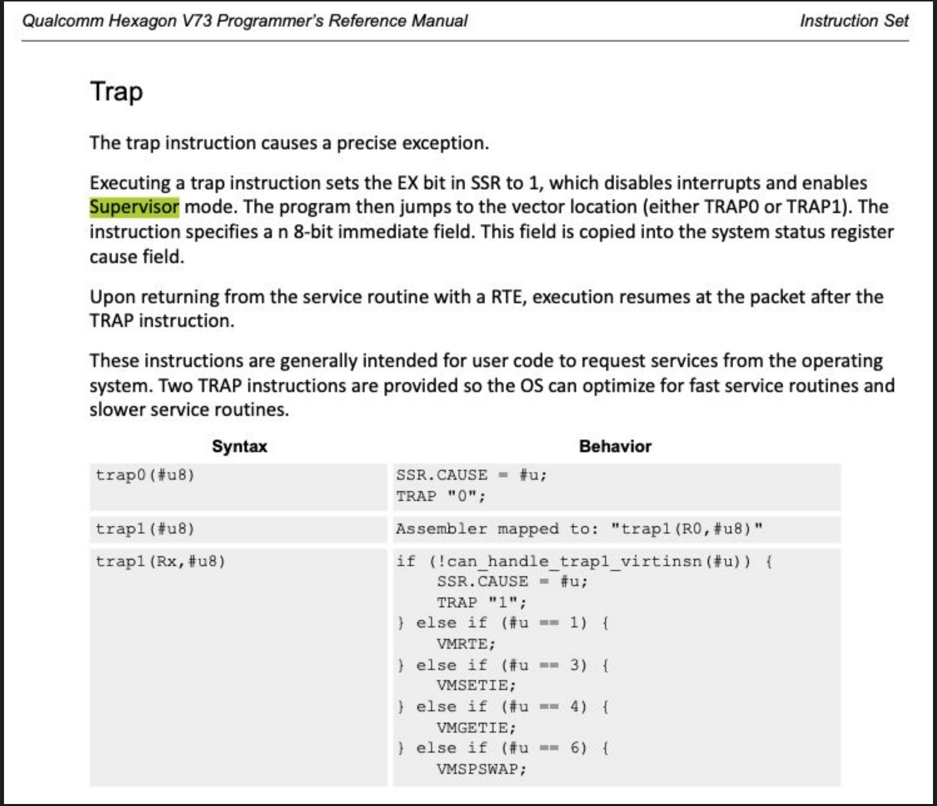


(54) **Title:** METHOD AND SYSTEM FOR TRUSTED/UNTRUSTED DIGITAL SIGNAL PROCESSOR DEBUGGING OPERATIONS



[0012] According to one aspect of the disclosed subject matter, a method and system for controlling between trusted and untrusted debugging operational modes includes the processes, circuitry, and instructions for operating a core processor process within a core processor associated with the digital signal processor. The method and system further operate a debugging process within a debugging mechanism of the digital signal processor, which debugging mechanism associates with the core processor. The core processor process determines the origin of debugging control as trusted debugging control or untrusted debugging control. In the event that debugging control is trusted debugging control, the core processor process provides to the trusted debugging control a first set of features and privileges. Alternatively, in the event that

Supervisor Mode



Trap

The trap instruction causes a precise exception.

Executing a trap instruction sets the EX bit in SSR to 1, which disables interrupts and enables Supervisor mode. The program then jumps to the vector location (either TRAP0 or TRAP1). The instruction specifies a n 8-bit immediate field. This field is copied into the system status register cause field.

Upon returning from the service routine with a RTE, execution resumes at the packet after the TRAP instruction.

These instructions are generally intended for user code to request services from the operating system. Two TRAP instructions are provided so the OS can optimize for fast service routines and slower service routines.

Syntax	Behavior

`trap0(#u8)`	`SSR.CAUSE = #u;`
	`TRAP "0";`

`trap1(#u8)`	`Assembler mapped to: "trap1(R0,#u8)"`

`trap1(Rx,#u8)`	`if (!can_handle_trap1_virtinsn(#u)) {`
	`    SSR.CAUSE = #u;`
	`    TRAP "1";`
	`} else if (#u == 1) {`
	`    VMRTE;`
	`} else if (#u == 3) {`
	`    VMSETIE;`
	`} else if (#u == 4) {`
	`    VMGETIE;`
	`} else if (#u == 6) {`
	`    VMSPSWAP;`

SYSCFG register

- Hexagon architecture register, exposed to assembler
  - But, undocumented
  - Patent shows "one way of forming the register" →
- **Supervisor-only (privileged)**
  - QURT kernel OR application in privileged mode of execution; eg. modem firmware in early boot
- Use to set **ISDB_TRUSTED** bit
  - 0x28 == 0b0..1000
- ISDB status bit will be tested by host debugger and eligible others

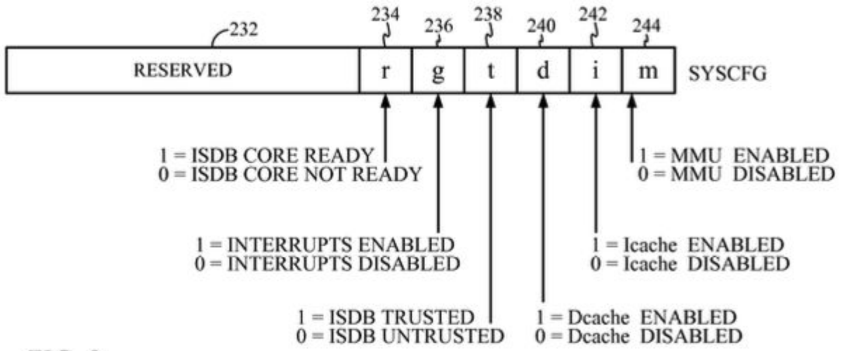
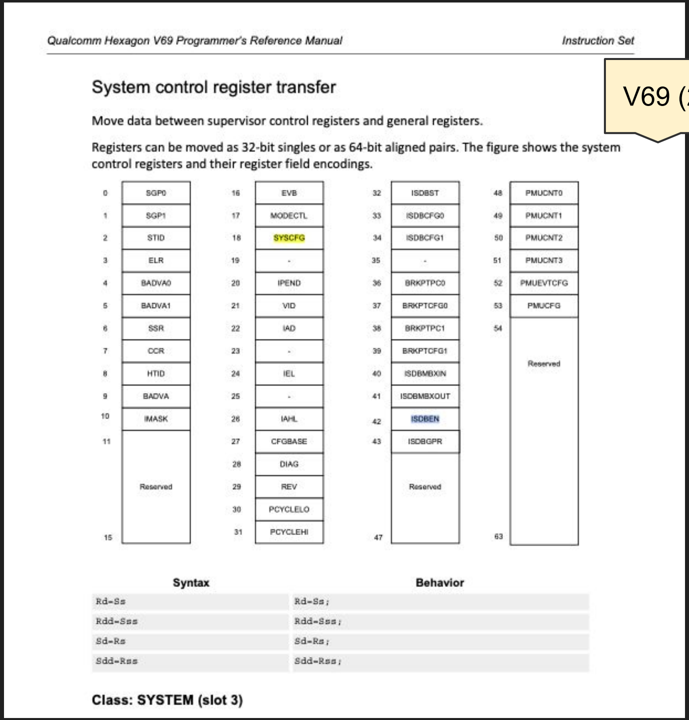


FIG. 8

- Patent documentation:
  - "Communication through a SYSCFG register as a 40-bit packet identifies the ISDB register to read/write and a 32-bit data payload"
  - RESERVED part?

How to program SYSCFG register?



Qualcomm Hexagon V69 Programmer's Reference Manual *Instruction Set*

System control register transfer

Move data between supervisor control registers and general registers.

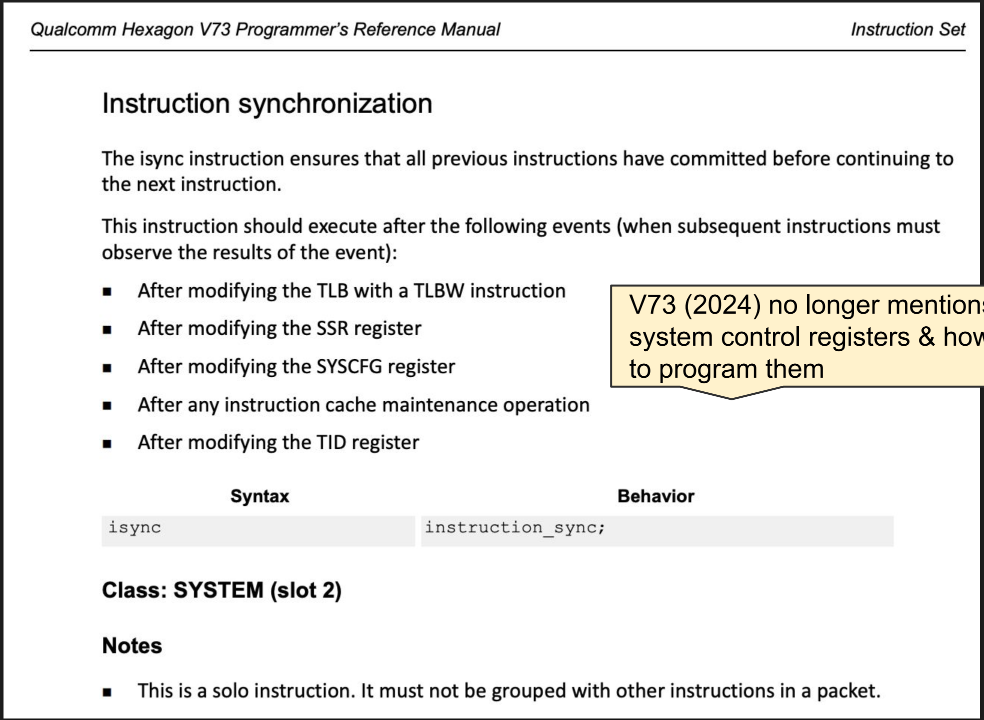
Registers can be moved as 32-bit singles or as 64-bit aligned pairs. The figure shows the system control registers and their register field encodings.

0	SGP0	16	EVB	32	ISDBST	48	PMUCNT0
1	SGP1	17	MODECTL	33	ISDBCFG0	49	PMUCNT1
2	STID	18	SYSCFG	34	ISDBCFG1	50	PMUCNT2
3	ELR	19	-	35	-	51	PMUCNT3
4	BADVA0	20	IPEND	36	BRKPTPC0	52	PMUEVTCFG
5	BADVA1	21	VID	37	BRKPTCFG0	53	PMUCFG
6	SSR	22	IAD	38	BRKPTPC1	54	Reserved
7	CCR	23	-	39	BRKPTCFG1		
8	HTID	24	IEL	40	ISDBMBXIN		
9	BADVA	25	-	41	ISDBMBXOUT		
10	IMASK	26	IAHL	42	ISDBEN		
11	Reserved	27	CFGBASE	43	ISDBGPR		
		28	DIAG		Reserved		
		29	REV				
		30	PCYCLELO				
15		31	PCYCLEHI	47		63	

Syntax	Behavior
`Rd=Ss`	`Rd=Ss;`
`Rdd=Sss`	`Rdd=Sss;`
`Sd=Rs`	`Sd=Rs;`
`Sdd=Rss`	`Sdd=Rss;`

Class: SYSTEM (slot 3)

V69 (2022)



Qualcomm Hexagon V73 Programmer's Reference Manual *Instruction Set*

Instruction synchronization

The isync instruction ensures that all previous instructions have committed before continuing to the next instruction.

This instruction should execute after the following events (when subsequent instructions must observe the results of the event):

- After modifying the TLB with a TLBW instruction
- After modifying the SSR register
- After modifying the SYSCFG register
- After any instruction cache maintenance operation
- After modifying the TID register

Syntax	Behavior
`isync`	`instruction_sync;`

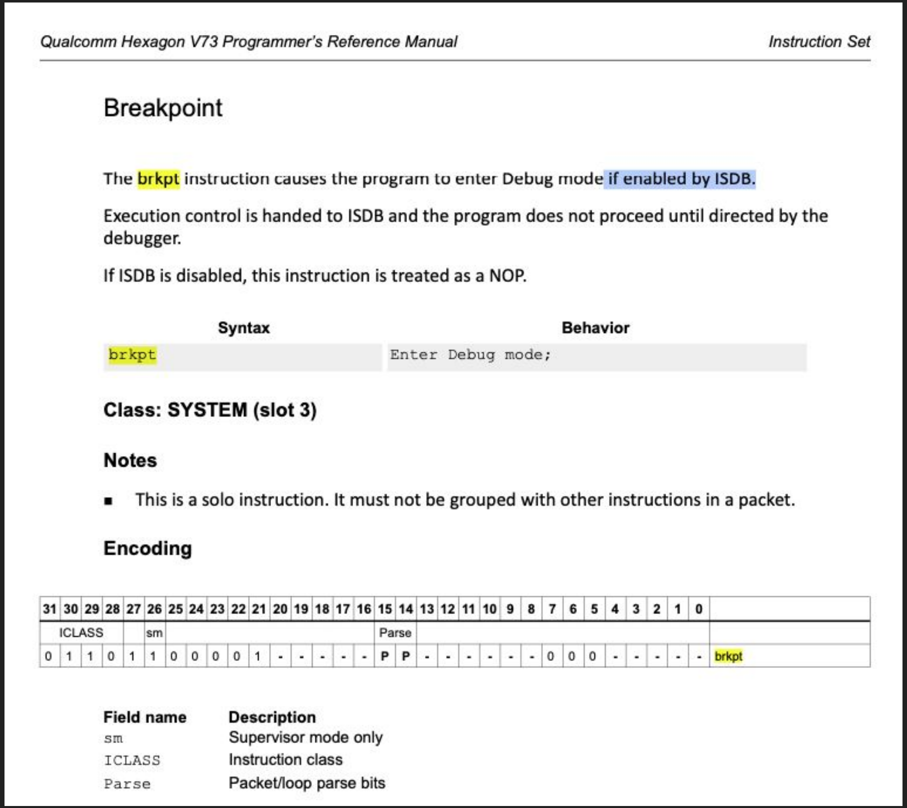
Class: SYSTEM (slot 2)

Notes

- This is a solo instruction. It must not be grouped with other instructions in a packet.

V73 (2024) no longer mentions system control registers & how to program them

Software breakpoint



Qualcomm Hexagon V73 Programmer's Reference Manual *Instruction Set*

Breakpoint

The brkpt instruction causes the program to enter Debug mode if enabled by ISDB.

Execution control is handed to ISDB and the program does not proceed until directed by the debugger.

If ISDB is disabled, this instruction is treated as a NOP.

Syntax	Behavior
`brkpt`	`Enter Debug mode;`

Class: SYSTEM (slot 3)

Notes

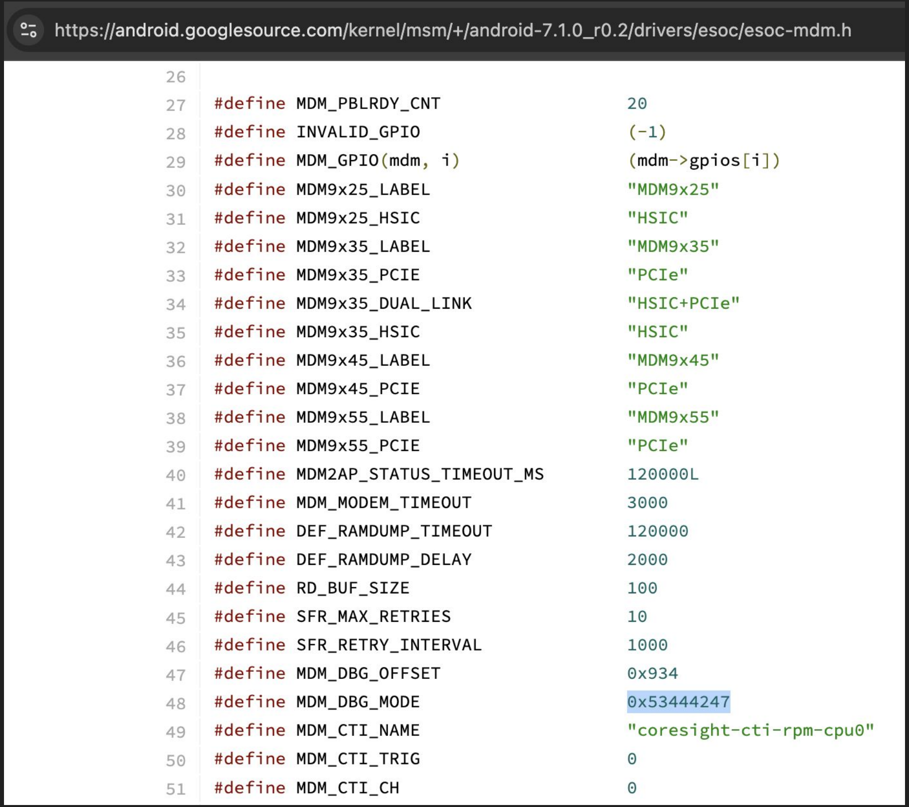
- This is a solo instruction. It must not be grouped with other instructions in a packet.

Encoding

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
ICLASS					sm											Parse																
0	1	1	0	1	1	0	0	0	0	1	-	-	-	-	-	P	P	-	-	-	-	-	-	0	0	0	-	-	-	-	-	brkpt

Field name	Description
`sm`	Supervisor mode only
`ICLASS`	Instruction class
`Parse`	Packet/loop parse bits

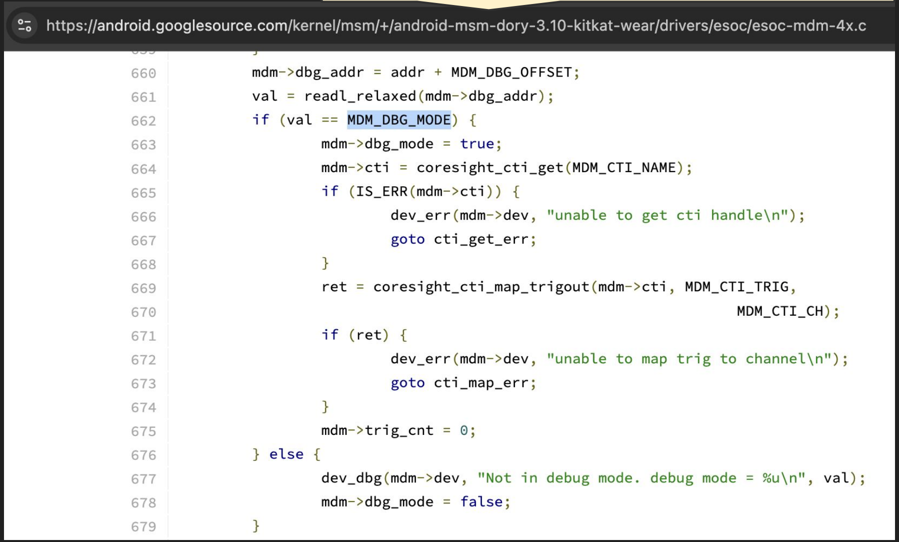
Magic Cookie



https://android.googlesource.com/kernel/msm/+/android-7.1.0_r0.2/drivers/esoc/esoc-mdm.h

```c

#define MDM_PBLRDY_CNT                  20
#define INVALID_GPIO                    (-1)
#define MDM_GPIO(mdm, i)                (mdm->gpios[i])
#define MDM9x25_LABEL                   "MDM9x25"
#define MDM9x25_HSIC                    "HSIC"
#define MDM9x35_LABEL                   "MDM9x35"
#define MDM9x35_PCIE                    "PCIe"
#define MDM9x35_DUAL_LINK               "HSIC+PCIe"
#define MDM9x35_HSIC                    "HSIC"
#define MDM9x45_LABEL                   "MDM9x45"
#define MDM9x45_PCIE                    "PCIe"
#define MDM9x55_LABEL                   "MDM9x55"
#define MDM9x55_PCIE                    "PCIe"
#define MDM2AP_STATUS_TIMEOUT_MS        120000L
#define MDM_MODEM_TIMEOUT               3000
#define DEF_RAMDUMP_TIMEOUT             120000
#define DEF_RAMDUMP_DELAY               2000
#define RD_BUF_SIZE                     100
#define SFR_MAX_RETRIES                 10
#define SFR_RETRY_INTERVAL              1000
#define MDM_DBG_OFFSET                  0x934
#define MDM_DBG_MODE                    0x53444247
#define MDM_CTI_NAME                    "coresight-cti-rpm-cpu0"
#define MDM_CTI_TRIG                    0
#define MDM_CTI_CH                      0
```

Newer msm kernels no longer leak this piece



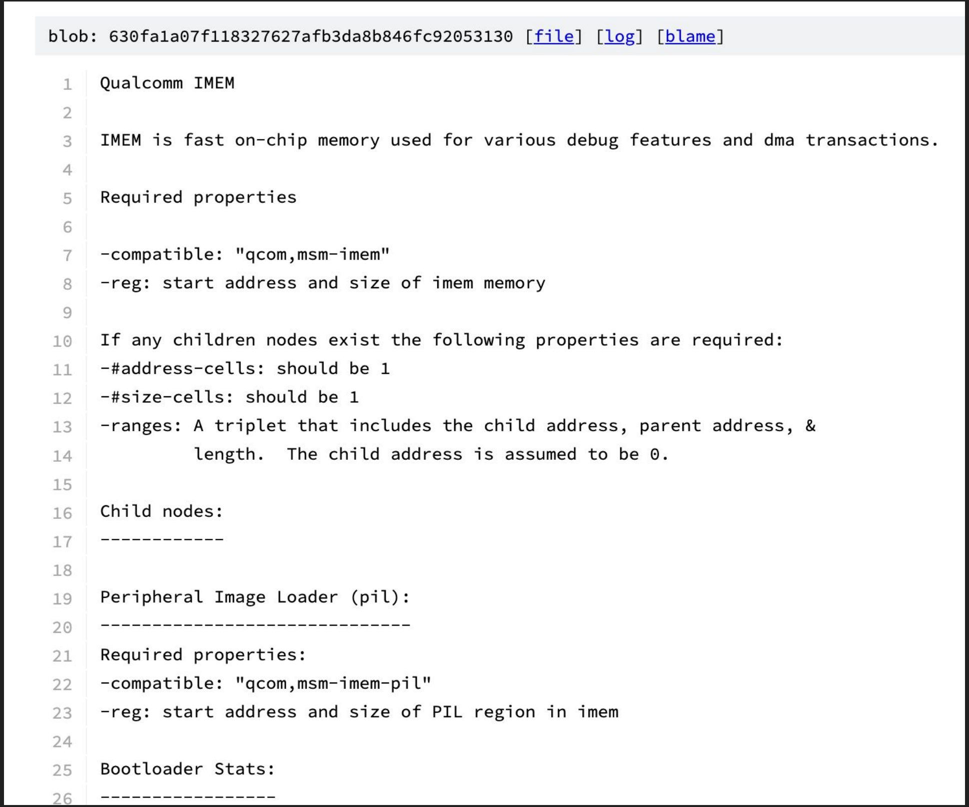
https://android.googlesource.com/kernel/msm/+/android-msm-dory-3.10-kitkat-wear/drivers/esoc/esoc-mdm-4x.c

```c
	mdm->dbg_addr = addr + MDM_DBG_OFFSET;
	val = readl_relaxed(mdm->dbg_addr);
	if (val == MDM_DBG_MODE) {
		mdm->dbg_mode = true;
		mdm->cti = coresight_cti_get(MDM_CTI_NAME);
		if (IS_ERR(mdm->cti)) {
			dev_err(mdm->dev, "unable to get cti handle\n");
			goto cti_get_err;
		}
		ret = coresight_cti_map_trigout(mdm->cti, MDM_CTI_TRIG,
								MDM_CTI_CH);
		if (ret) {
			dev_err(mdm->dev, "unable to map trig to channel\n");
			goto cti_map_err;
		}
		mdm->trig_cnt = 0;
	} else {
		dev_dbg(mdm->dev, "Not in debug mode. debug mode = %u\n", val);
		mdm->dbg_mode = false;
	}
```

0x53444247 ‘SDBG’

Qualcomm IMEM

- Shared memory
- Exposed in MSM →
- Undocumented

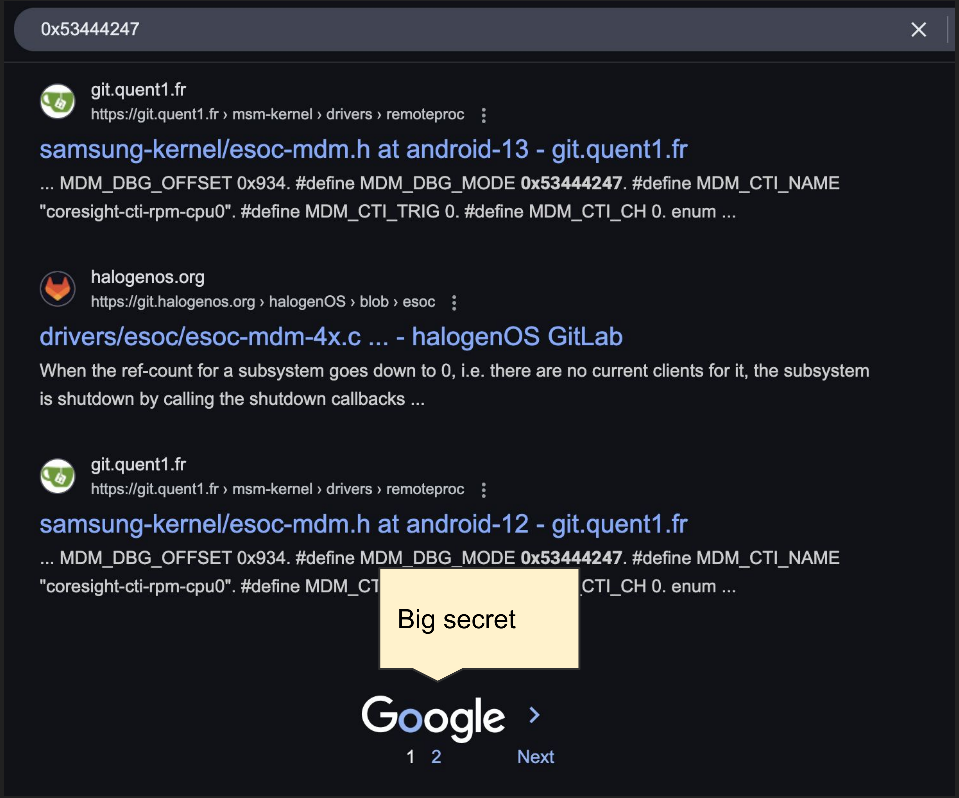


blob: 630fa1a07f118327627afb3da8b846fc92053130 [file] [log] [blame]

```
Qualcomm IMEM

IMEM is fast on-chip memory used for various debug features and dma transactions.

Required properties

-compatible: "qcom,msm-imem"
-reg: start address and size of imem memory

If any children nodes exist the following properties are required:
-#address-cells: should be 1
-#size-cells: should be 1
-ranges: A triplet that includes the child address, parent address, &
         length.  The child address is assumed to be 0.

Child nodes:
------------


Peripheral Image Loader (pil):
------------------------------
Required properties:
-compatible: "qcom,msm-imem-pil"
-reg: start address and size of PIL region in imem

Bootloader Stats:
-----------------
```

Enable Hexagon debugging with Magic Cookie

- QURT kernel operates ISDB, mostly via privileged mode
- It uses a simple flag-based mechanism to trigger ISDB operations for applications/users
- 0x53444247 (‘SDBG’ in hex)
- **Put the magic cookie in IMEM via JTAG**
  - You need to know **specific offset** in IMEM for each application/control
  - Modem, PIL, mba, Android msm, QURT kernel will check the cookie
  - Triggers software setup consistent with debug mode of thread, and/or **enter debug mode** via ISDB


0x53444247

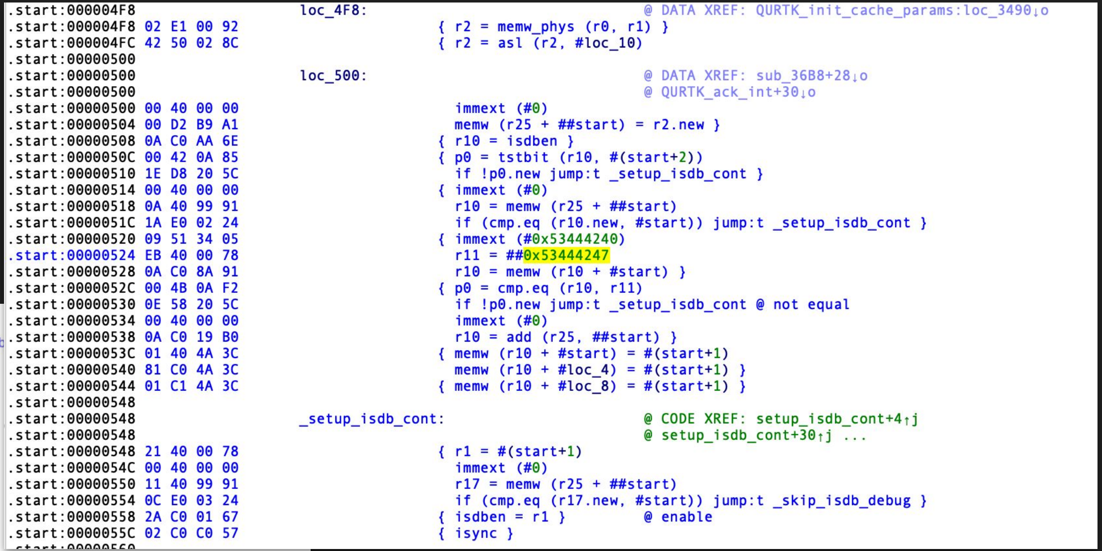
git.quent1.fr
https://git.quent1.fr › msm-kernel › drivers › remoteproc
samsung-kernel/esoc-mdm.h at android-13 - git.quent1.fr
... MDM_DBG_OFFSET 0x934. #define MDM_DBG_MODE **0x53444247**. #define MDM_CTI_NAME "coresight-cti-rpm-cpu0". #define MDM_CTI_TRIG 0. #define MDM_CTI_CH 0. enum ...

halogenos.org
https://git.halogenos.org › halogenOS › blob › esoc
drivers/esoc/esoc-mdm-4x.c ... - halogenOS GitLab
When the ref-count for a subsystem goes down to 0, i.e. there are no current clients for it, the subsystem is shutdown by calling the shutdown callbacks ...

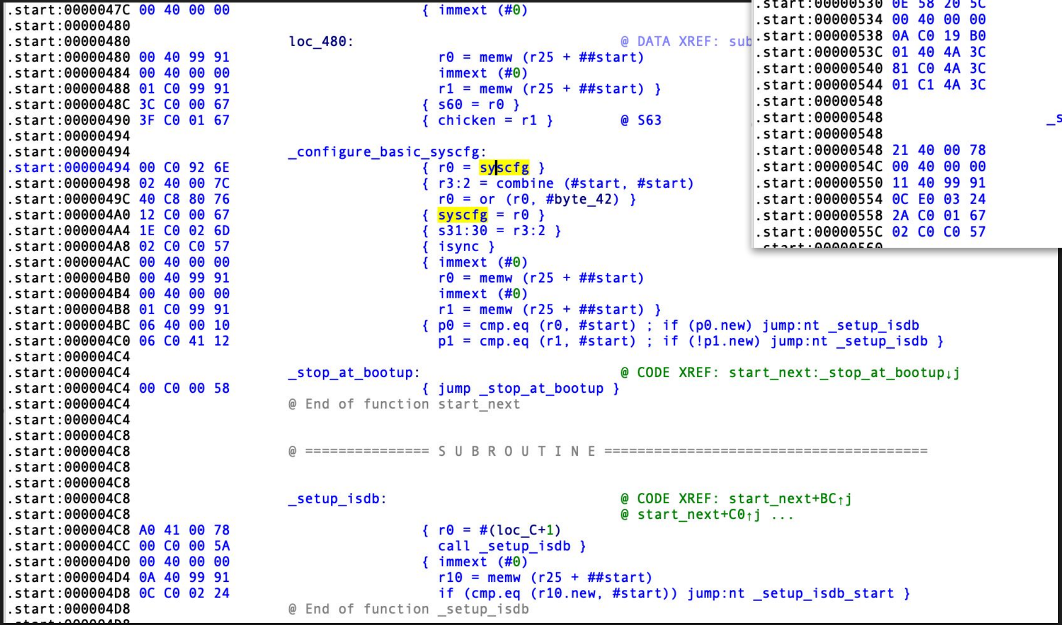
git.quent1.fr
https://git.quent1.fr › msm-kernel › drivers › remoteproc
samsung-kernel/esoc-mdm.h at android-12 - git.quent1.fr
... MDM_DBG_OFFSET 0x934. #define MDM_DBG_MODE **0x53444247**. #define MDM_CTI_NAME "coresight-cti-rpm-cpu0". #define MDM_CT CTI_CH 0. enum ...

Big secret

qurtkernel.o



```
.start:000004F8                         loc_4F8:                                @ DATA XREF: QURTK_init_cache_params:loc_3490↓o
.start:000004F8 02 E1 00 92                             { r2 = memw_phys (r0, r1) }
.start:000004FC 42 50 02 8C                             { r2 = asl (r2, #loc_10)
.start:00000500
.start:00000500                         loc_500:                                @ DATA XREF: sub_36B8+28↓o
.start:00000500                                                                 @ QURTK_ack_int+30↓o
.start:00000500 00 40 00 00                               immext (#0)
.start:00000504 00 D2 B9 A1                               memw (r25 + ##start) = r2.new }
.start:00000508 0A C0 AA 6E                             { r10 = isdben }
.start:0000050C 00 42 0A 85                             { p0 = tstbit (r10, #(start+2))
.start:00000510 1E D8 20 5C                               if !p0.new jump:t _setup_isdb_cont }
.start:00000514 00 40 00 00                             { immext (#0)
.start:00000518 0A 40 99 91                               r10 = memw (r25 + ##start)
.start:0000051C 1A E0 02 24                               if (cmp.eq (r10.new, #start)) jump:t _setup_isdb_cont }
.start:00000520 09 51 34 05                             { immext (#0x53444240)
.start:00000524 EB 40 00 78                               r11 = ##0x53444247
.start:00000528 0A C0 8A 91                               r10 = memw (r10 + #start) }
.start:0000052C 00 4B 0A F2                             { p0 = cmp.eq (r10, r11)
.start:00000530 0E 58 20 5C                               if !p0.new jump:t _setup_isdb_cont @ not equal
.start:00000534 00 40 00 00                               immext (#0)
.start:00000538 0A C0 19 B0                               r10 = add (r25, ##start) }
.start:0000053C 01 40 4A 3C                             { memw (r10 + #start) = #(start+1)
.start:00000540 81 C0 4A 3C                               memw (r10 + #loc_4) = #(start+1) }
.start:00000544 01 C1 4A 3C                             { memw (r10 + #loc_8) = #(start+1) }
.start:00000548
.start:00000548                         _setup_isdb_cont:                       @ CODE XREF: setup_isdb_cont+4↑j
.start:00000548                                                                 @ setup_isdb_cont+30↑j ...
.start:00000548 21 40 00 78                             { r1 = #(start+1)
.start:0000054C 00 40 00 00                               immext (#0)
.start:00000550 11 40 99 91                               r17 = memw (r25 + ##start)
.start:00000554 0C E0 03 24                               if (cmp.eq (r17.new, #start)) jump:t _skip_isdb_debug }
.start:00000558 2A C0 01 67                             { isdben = r1 }         @ enable
.start:0000055C 02 C0 C0 57                             { isync }
```



```
.start:0000047C 00 40 00 00                             { immext (#0)
.start:00000480
.start:00000480                         loc_480:                                @ DATA XREF: sub
.start:00000480 00 40 99 91                               r0 = memw (r25 + ##start)
.start:00000484 00 40 00 00                               immext (#0)
.start:00000488 01 C0 99 91                               r1 = memw (r25 + ##start) }
.start:0000048C 3C C0 00 67                             { s60 = r0 }
.start:00000490 3F C0 01 67                             { chicken = r1 }        @ S63
.start:00000494
.start:00000494                         _configure_basic_syscfg:
.start:00000494 00 C0 92 6E                             { r0 = syscfg }
.start:00000498 02 40 00 7C                             { r3:2 = combine (#start, #start)
.start:0000049C 40 C8 80 76                               r0 = or (r0, #byte_42) }
.start:000004A0 12 C0 00 67                             { syscfg = r0 }
.start:000004A4 1E C0 02 6D                             { s31:30 = r3:2 }
.start:000004A8 02 C0 C0 57                             { isync }
.start:000004AC 00 40 00 00                             { immext (#0)
.start:000004B0 00 40 99 91                               r0 = memw (r25 + ##start)
.start:000004B4 00 40 00 00                               immext (#0)
.start:000004B8 01 C0 99 91                               r1 = memw (r25 + ##start) }
.start:000004BC 06 40 00 10                             { p0 = cmp.eq (r0, #start) ; if (p0.new) jump:nt _setup_isdb
.start:000004C0 06 C0 41 12                               p1 = cmp.eq (r1, #start) ; if (!p1.new) jump:nt _setup_isdb }
.start:000004C4
.start:000004C4                         _stop_at_bootup:                        @ CODE XREF: start_next:_stop_at_bootup↓j
.start:000004C4 00 C0 00 58                             { jump _stop_at_bootup }
.start:000004C4                         @ End of function start_next
.start:000004C4
.start:000004C8
.start:000004C8                         @ ============== S U B R O U T I N E =======================================
.start:000004C8
.start:000004C8
.start:000004C8                         _setup_isdb:                            @ CODE XREF: start_next+BC↑j
.start:000004C8                                                                 @ start_next+C0↑j ...
.start:000004C8 A0 41 00 78                             { r0 = #(loc_C+1)
.start:000004CC 00 C0 00 5A                               call _setup_isdb }
.start:000004D0 00 40 00 00                             { immext (#0)
.start:000004D4 0A 40 99 91                               r10 = memw (r25 + ##start)
.start:000004D8 0C C0 02 24                               if (cmp.eq (r10.new, #start)) jump:nt _setup_isdb_start }
.start:000004D8                         @ End of function _setup_isdb
```

Conclusions

Technology summary

- ISDB is the low-level debugging circuitry of Hexagon architecture which sits in-between JTAG and the core
  - Don’t confuse with ISDB-T, a digital TV broadcasting standard
- Debugging works by reading/writing ISDB registers, via either JTAG or software
- Multiple ways of doing things
- This research is the first step
  - System internals of ISDB
  - Key requirements to enable and control debugging over JTAG and via software
  - **Untested - may need extra config!**

Security aspects

- Basically, ISDB is the **core gatekeeper of debugging** on Hexagon cores
  - Blocks JTAG access if is ISDB_TRUSTED register is not set
  - Exposes software-based debugging controls via proprietary kernel code
- **Trusted or Untrusted** mode of operation
  - Trusted: Qualcomm’s kernel dev
  - Untrusted: you
  - Actually programmable
- Specialized enablement and configuration protocols
- **Qurt Kernel will check other debugging controls before enabling ISDB**
  - Build-time configuration variables
  - CoT & Attestation Certificates, Fuses, IMEM
  - Inject your own ISDB enablement logic somewhere to bypass it (supervisor mode)

References

1. A.Esage, “Advanced Hexagon Diag”, Chaos Communications Congress (2020)
2. A.Esage, “Deep Dive: Qualcomm MSM Linux Kernel & ARM Mali GPU 0-day Exploit Attacks of October 2023”, Zero Day Engineering Research Blog (2023)
3. APQ8016E Technical Reference Manual
4. Qualcomm® Snapdragon™ 410 Processor APQ8016 Hardware Register Description
5. Qualcomm® Snapdragon™ 410E (APQ 8016E) Processor Device Specification
6. WIPO patent no.2008/061067 A2
7. WIPO patent no.2008/061089 A2
8. US patent no.7,657,791 B2 of Feb. 2, 2010
9. Qualcomm Hexagon V66 Programmer’s Reference Manual (2017)
10. Qualcomm Hexagon V69 Programmer’s Reference Manual (2022)
11. Qualcomm Hexagon V73 Programmer’s Reference Manual (2024)

Q&A

Twitter/Youtube: @alisaesage

Email: contact@zerodayengineering.com

http://zerodayengineering.com