

Remote monitoring en troubleshooting van IoT en internet-connected applicaties

Gerard Fianen

INDES – Integrated Development Solutions BV

D&E
EVENT

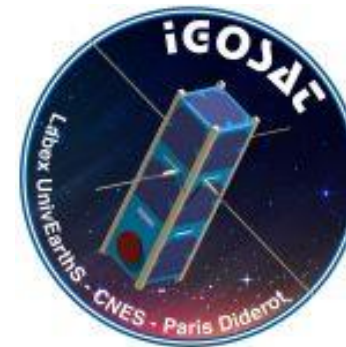


Het ontwerpen van
innovatieve elektronica

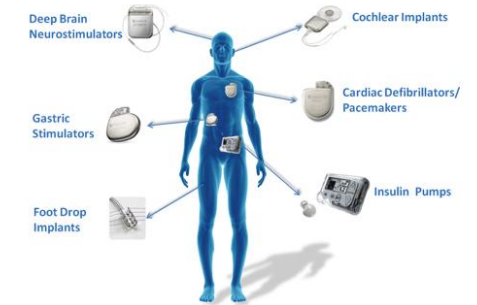
Woensdag 19 april 2023
1931 Congrescentrum 's-Hertogenbosch

IoT en Internet connected Devices

- Kleine sensors in grote aantallen
 - Goedkoop, laag stroomverbruik, beperkt geheugen
- Industriële robots en andere fout tolerante applicaties
 - Non stop



WIRELESS IMPLANTABLE MEDICAL DEVICES



Over-The-Air updates zijn (maar) een deel van de oplossing

- Wat is de lastigste probleemomschrijving ?

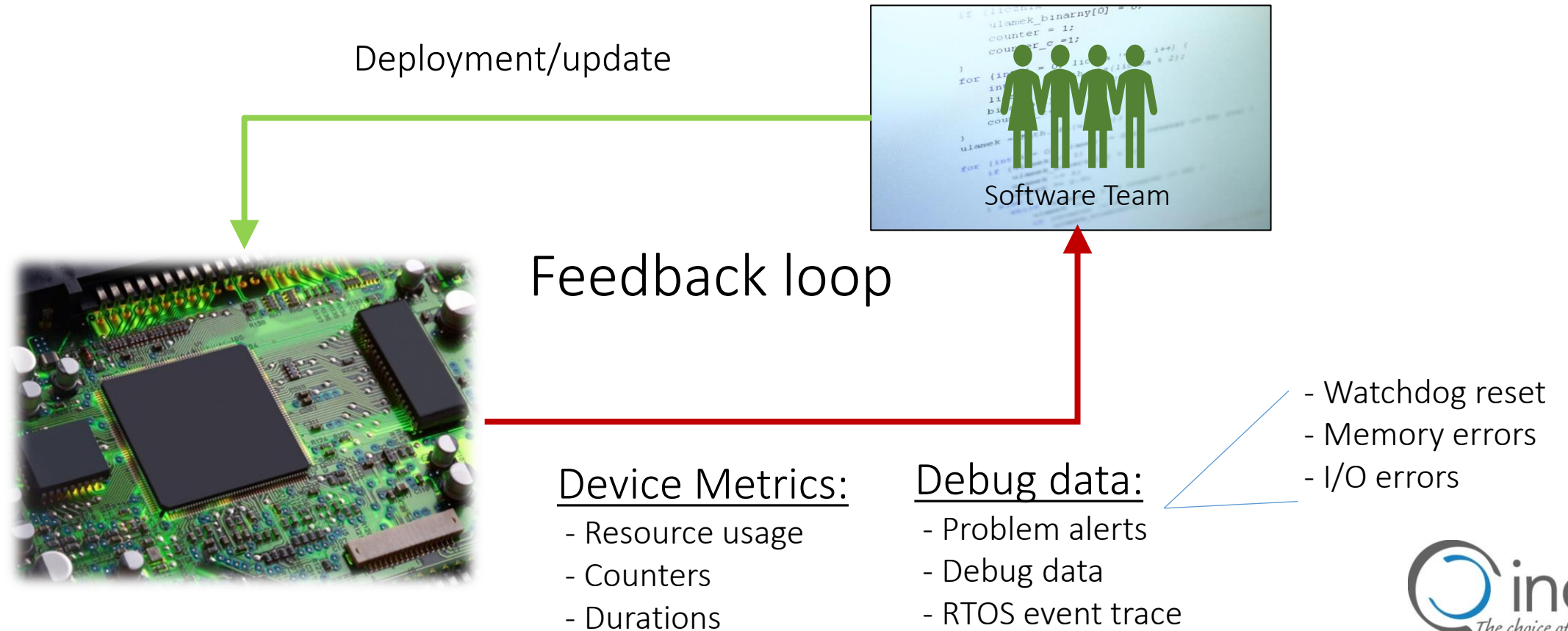
“Hij doet het niet ...”

“ eeh... ... Soms ... ”

- Als je een probleem kunt reproduceren is het zo goed als opgelost
- Root cause analysis / Post Mortem analysis



Feedback loop

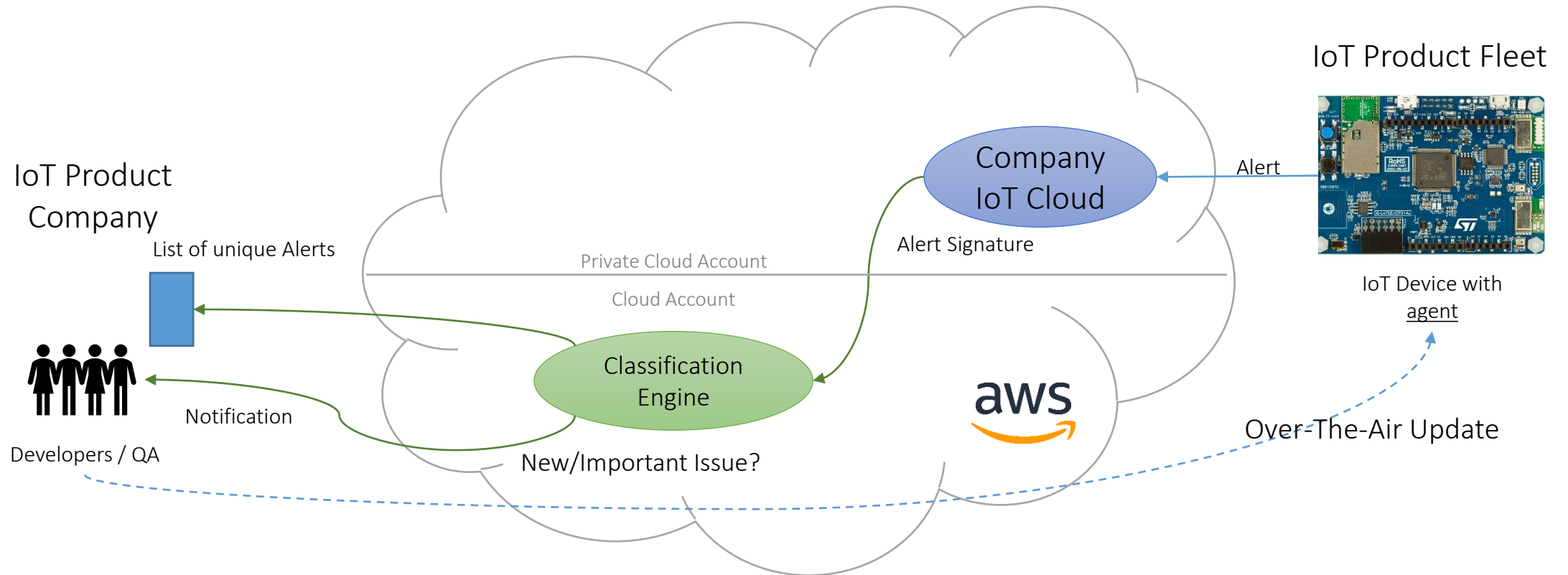


Aandachtspunten

- Hoe ga ik om meer te veel alerts ?
 - Meerdere devices in het veld die eenzelfde error geven
 - Te veel irrelevante data in log
- Data security / privacy !
- Beperkte resources ?



Secure Remote Device Monitoring



High-level dashboard

Issue Overview

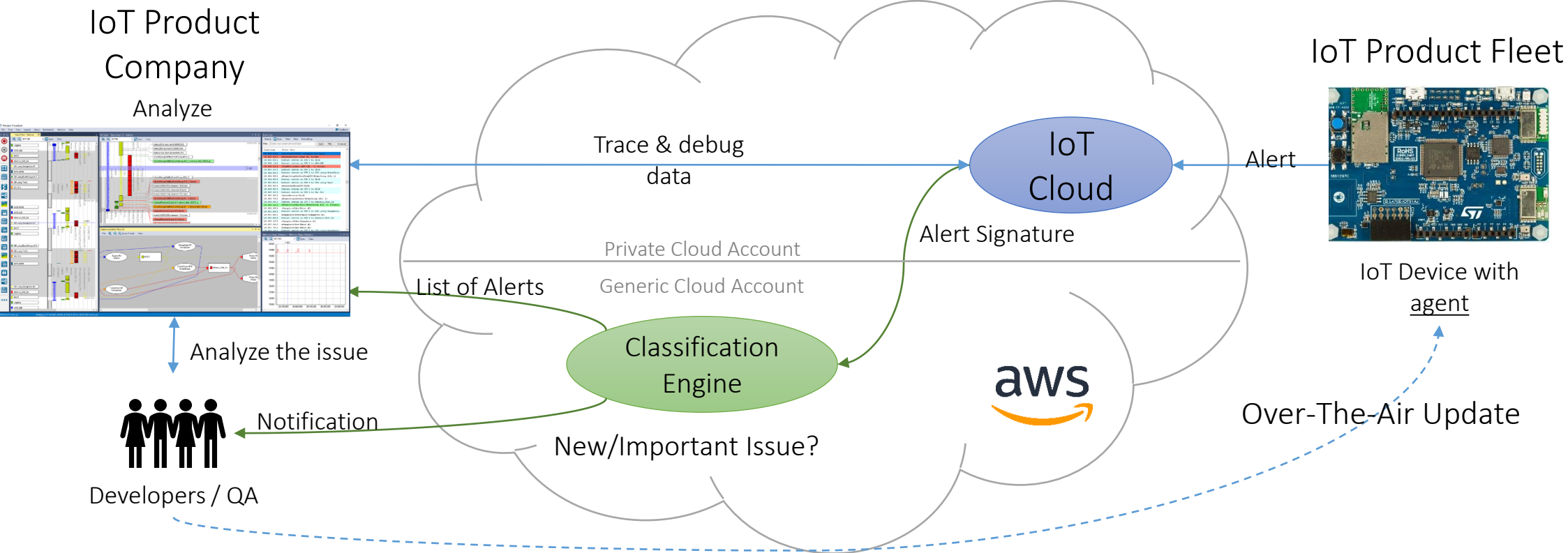
Description ▼	Count ▼	Latest Occurrence ▼	Details	Alerts	Latest Trace
CPU Overload	3	2022-08-02 13:19:12	View	Alerts	Open
Assert Failed	1	2022-08-02 13:18:50	View	Alerts	Open
Heartbeat failure	4	2022-08-02 13:18:32	View	Alerts	Open
Assert Failed	1	2022-07-26 14:20:44	View	Alerts	Open

Each alert can provide multiple types of diagnostics :

- System traces – Timeline of tasks, API calls, etc.
- Crash dumps – Registers, stack trace, memory
- Device logs – Existing application logs?
- Other device data? Anything can be included



Remote Device Monitoring



Register dump

```
crash.txt - Anteckningar
Arkiv Redigera Format Visa Hjälp
Reading symbols from devalert-f407-elevator-controller.elf...

Remote debugging using | C:/DevAlertDispatcher/CrashDebug.exe --elf devalert-f407-elevator-contr
prvIdleTask (pvParameters=0x8005791 <xTaskIncrementTick+524>)
    at ../freertos/tasks.c:3481
Backtrace
#0 prvIdleTask (pvParameters=0x8005791 <xTaskIncrementTick+524>)
    at ../freertos/tasks.c:3481
#1 0x00000000 in ?? ()

Fault status registers
0xe000ed28: 0x00000000 0x00000000 0x00000000
0xe000ed38: 0x00000000

Registers
r0 0x0 0
r1 0xa5a5a5a5 -1515870811
r2 0x2001df9c 536993692
r3 0x0 0
r4 0xa5a5a5a5 -1515870811
```

Debug dump

```
Debug
DevAlertCrashDump [GDB Hardware Debugging]
latest.elf
  Thread #1 <main> (Suspended : User Request)
    __stack_chk_fail() at main.c:215 0x80016bc
    testStackSmash() at main.c:224 0x80016fc
    ButtonTask() at main.c:266 0x800176a
C:/DevAlertDispatcher/arm-none-eabi-gdb.exe (8.3.1.20191211)

main.c tasks.c startup_stm3... aws_dev_mod... iot_wifi.c dfmAlert.c
209
210 void *__stack_chk_guard = (void *)0xdeadbeef;
211
212 void __stack_chk_fail(void)
213 {
214     // If you get a hard fault here, the stack has been corrupted by the calling function.
215     volatile int stackcheckfailed = *(unsigned int*)0x00100000;
216 }
217
218 void testStackSmash(void)
219 {
220     volatile char buf[10];
221
222     // Will trigger the above fault handler on return (using -fstack-protector-strong flag)
223     sprintf(buf, "This will corrupt the stack");
224 }
225
```

Trace data ..

The image shows a debugger interface with two main windows: 'Trace' and 'Disassembly'.

Trace Window:

Timestamp	Trace
2129	404008C2 blt a2, a1, 0x404008A6 s <<= 1;
2130	404008A6 c.slli a0, 1 s += ZERO_2; // Complex adding of 0 ..
2131	404008A8 lw a3, 0x24(gp)
2132	404008AC c.add a0, a3
2133	404008AE lw a3, 0x20(gp)
2134	404008B2 c.add a0, a3
2135	404008B4 lw a3, 0x1C(gp)
2136	404008B8 c.add a0, a3
2137	404008BA lw a3, 0x18(gp)
2138	404008BE c.add a0, a3 for (int i = 0; i < n; i++)
2139	404008C0 c.addi a2, 1 for (int i = 0; i < n; i++)
2140	404008C2 blt a2, a1, 0x404008A6 s <<= 1;
2141	404008A6 c.slli a0, 1 s += ZERO_2; // Complex adding of 0 ..
2142	404008A8 lw a3, 0x24(gp)
2143	404008AC c.add a0, a3
2144	404008AE lw a3, 0x20(gp)
2145	404008B2 c.add a0, a3

Disassembly Window:

Go to: Memory

```
Disassembly
4040089A 6141 c.addi16sp 0x10
4040089C 8082 c.ret
int xpow2(int n)
{
xpow2:
4040089E 85AA c.mv a1, a0
int s = 1;
404008A0 4505 c.li a0, 1
for (int i = 0; i < n; i++)
404008A2 4601 c.li a2, 0
404008A4 A839 c.j 0x404008C2
s <<= 1;
404008A6 0506 c.slli a0, 1
s += ZERO_2; // Complex adding of 0 (to
404008A8 0241A683 lw a3, 0x24(gp)
404008AC 9536 c.add a0, a3
404008AE 0201A683 lw a3, 0x20(gp)
404008B2 9536 c.add a0, a3
404008B4 01C1A683 lw a3, 0x1C(gp)
404008B8 9536 c.add a0, a3
404008BA 0181A683 lw a3, 0x18(gp)
404008BE 9536 c.add a0, a3
for (int i = 0; i < n; i++)
```



Trace met beperkte resources ?

- Wireless Bandbreedte
- Geheugen
- Stroomverbruik



Instruction Trace

```

ime      info
184140   000100fc -> write r31
184141  100a8   IncrementCounterBy1  add      %r0,%
184141   0000002e -> write r0
184142  100ac   IncrementCounterBy1+0x04  extb
184142   0000002e -> write r0
184143  100b0   IncrementCounterBy1+0x08  j_s
184144  100fc   main+0x30      stb      %r0,[%r1]
184144           2e -> write mem [0x011000]
184145  10100   main+0x34      ldb      %r0,[%r2]
184145           5a <- read mem [0x011001] -> write
184146   0000005a -> write r0
184147  10104   main+0x38      bl_s      IncrementC
184147   00010106 -> write r31
184148  100b4   IncrementCounterBy2  add      %r0,%
184148   0000005c -> write r0
184149  100b8   IncrementCounterBy2+0x04  extb
184149   0000005c -> write r0
184150  100bc   IncrementCounterBy2+0x08  j_s
184151  10106   main+0x3a      stb      %r0,[%r2]

```

Printf / Application logging

```

Svc] Starting key provisioning...
Svc] Write root certificate...
Svc] Write device private key...
  Svc] Write device certificate...
  Svc] Key provisioning done...
  Svc] Starting WiFi...
mr Svc] WiFi module initialized.
WS-MAIN] WiFi connected to AP AndroidAP.
AWS-MAIN] Attempt to Get IP.
AWS-MAIN] IP Address acquired 192.168.0.51
AWS-LED] [Shadow 0] MQTT: Creation of dedicated MQTT
AWS-LED] Sending command to MQTT task.
MQTT] Received message 10000 from queue.
MQTT] Looked up a7sw0r7rvpirn.iot.us-east-1.amazona
[MQTT] MQTT Connect was accepted. Connection establ
[MQTT] Notifying task.
[AWS-LED] Command sent to MQTT task passed.

```

Trace on another level : Event Trace

```

09.319] Context switch on CPU 0 to Control
09.330] xQueueReceive(CtrlDataQueue, 100) return
10.253] OS Ticks: 8109
11.253] OS Ticks: 8110
11.270] Context switch on CPU 0 to Pos_ADC_ISR
11.281] xQueueSendFromISR(CtrlDataQueue)
11.290] Context switch on CPU 0 to Control
11.868] xQueueSend(MotorQueue)
11.878] Actor Ready: Motor
11.889] Context switch on CPU 0 to Motor
11.900] xQueueReceive(MotorQueue, 10) returns s
11.934] xQueueReceive(MotorQueue, 10) blocks
11.954] Context switch on CPU 0 to Control
11.965] xQueueReceive(CtrlCmdQueue, 0) timeout,
11.977] xQueueReceive(CtrlDataQueue, 100)
11.990] xQueueReceive(CtrlCmdQueue, 0) timeout,
  
```

	Instruction Trace	(RTOS) Event Trace	Application Logging
Producer	Processor core	Software (API or Kernel)	Software (application)
Abstraction Level	Low	Medium	High
Overhead	None	Some	More
System Requirements	High	Low	Low
Flexibility	Low	High	High

Typical example : With just 5 KB of trace data, 350 ms trace (during a busy period, with many details)

The screenshot displays the Percepio Tracealyzer interface with several key components:

- Trace View - Vertical:** Shows a timeline of events for various components including Logging, AWS-LED, MQTT, Atheros_Wifi_Ta, ISR using Semaphore #1, AWS-MAIN, ISR using EventGroup #19, and Tmr Svc. A specific time range is highlighted with a blue bar.
- No Sync - Trace View [1] - Vertical:** Provides a detailed view of the selected time range, showing memory allocations (malloc) and deallocations (free), semaphore operations, and event group interactions. Key events include:
 - malloc(216) returned 0x0000C2C8
 - malloc(80) returned 0x0000C3A0
 - malloc(160) returned 0x0000C3F0
 - xEventGroupSetBits(EventGroup #19, 1)
 - xEventGroupWaitBits(EventGroup #19, 1) returns after 56694 µs
 - xEventGroupSetBits(EventGroup #20, 1)
 - xEventGroupWaitBits(EventGroup #19, 1) blocks
 - free(0x0000C3F0) released 160 bytes
 - free(0x0000C3A0) released 80 bytes
 - free(0x0000C2C8) released 216 bytes
 - xEventGroupWaitBits(EventGroup #1, 1) blocks
 - xQueueReceive(Queue #1) returns after 28272 µs
 - xEventGroupWaitBits(EventGroup #1, 1) timeout after 689 µs
 - xQueueReceive(Queue #3) blocks
 - free(0x0000C258) released 112 bytes
 - xQueueReceive(Queue #1) blocks
- Event Log:** Lists system events with timestamps and descriptions, such as:
 - [29.858.294] free(0x0000C258) released 112 bytes
 - [29.858.618] xQueueReceive(Queue #1) blocks
 - [29.858.893] Context switch on CPU 0 to IDLE
 - [29.859.118] Context switch on CPU 0 to AWS-LED
 - [29.859.312] xTaskNotifyWait(AWS-LED, -1) blocks
 - [29.859.569] Context switch on CPU 0 to IDLE
 - [29.904.890] Context switch on CPU 0 to ISR using EventGroup...
 - [29.904.890] xEventGroupSetBitsFromISR(EventGroup #19, 1)
 - [29.904.890] Context switch on CPU 0 to IDLE
 - [29.905.067] Context switch on CPU 0 to ISR using TmrQ
 - [29.905.067] xQueueSendFromISR(TmrQ)
 - [29.905.067] Context switch on CPU 0 to IDLE
 - [29.905.320] Context switch on CPU 0 to Tmr Svc
 - [29.905.580] xQueueReceive(TmrQ)
 - [29.905.759] xEventGroupSetBits(EventGroup #19, 1)
 - [29.906.083] Context switch on CPU 0 to Atheros_Wifi_Ta
 - [29.906.247] xEventGroupWaitBits(EventGroup #19, 1) returns...
 - [29.906.478] xSemaphoreTake(Mutex #2)
 - [29.907.088] Context switch on CPU 0 to ISR using Semaphore...
 - [29.907.088] xSemaphoreGiveFromISR(Semaphore #1)
 - [29.907.088] Context switch on CPU 0 to Atheros_Wifi_Ta
 - [29.907.317] xSemaphoreTake(Semaphore #1)
 - [29.907.540] xSemaphoreGive(Mutex #2)
 - [29.907.796] xSemaphoreTake(Mutex #2)
- Communication Flow [1]:** A flow diagram showing the relationships between components like Queue #3 Queue, MQTT, Semaphore #4 Semaphore, EventGroup #19 EventGroup, Atheros_Wifi_Ta, Semaphore #1 Semaphore, Mutex #1 Mutex, and Mutex #2 Mutex.
- Memory Heap Utilization - Memory Heap Utilization:** A graph showing memory usage over time, with a peak of 45000 bytes at approximately 29.900.000 ms.

What can be analyzed ? - 1

Some examples:



- Multi-threading and timing
 - Context switches, internal kernel events
 - Execution time, response time, periodicity...
- API calls (OS, Middleware stacks, Drivers...)
 - Call sequences and timing
 - Parameters and return values
 - Blocking and timeouts
 - Object dependencies
- Application logging
 - Debug messages, variable values...
- Time between important events
- State changes over time

What can be analyzed ? - 2

RTOS Analytics

- Service Call Block Time
- Object Utilization
- Message Receive Time
- Priority Changes
- Event Intensity (three types)

CPU Usage

- CPU Load Graph
- Actor Statistics Report
- Actor Instance Graphs
- Interval Coverage Graph



Application Analytics

- Communication Flow
- State Machine Graph
- User Events Signal Plot
- I/O Intensity and I/O Plot

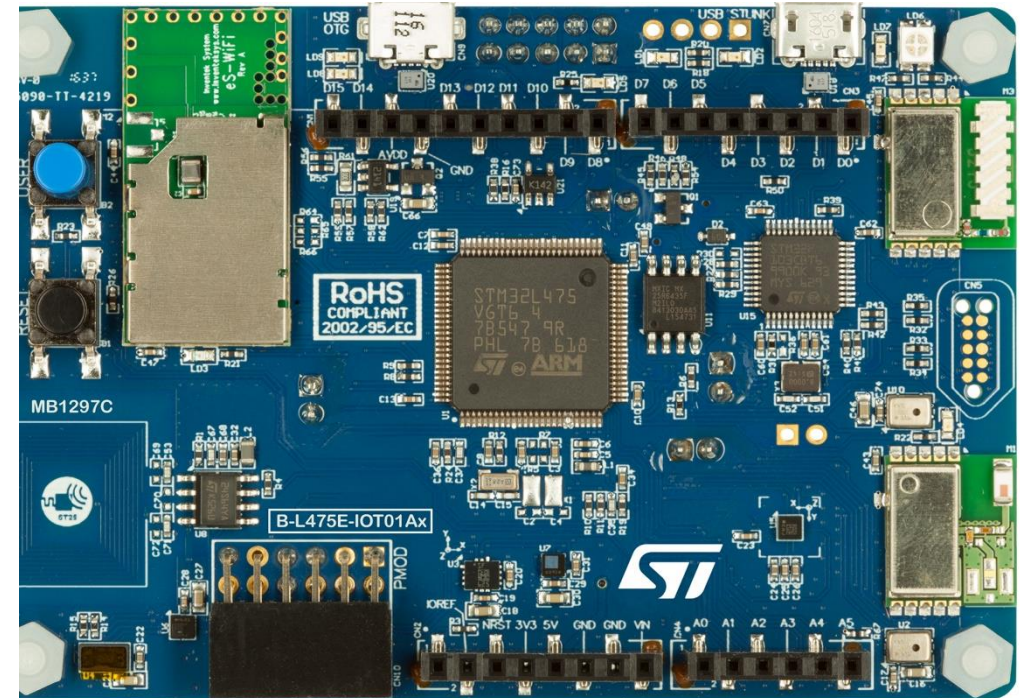
Memory Usage

- Stack Usage
- Memory Heap Utilization

All views are **interconnected** in clever ways, so you can click on a data point in one view and see the corresponding location in another related view.

Demonstratie

- IoT STM Discovery IoT node
- Wifi naar onze AWS cloud
- Introductie van run-time fouten





Cross Compilers, Debuggers, IDE
 RTOS, Middleware, Protocol stacks, GUI, Database
 Debug & Trace probes

Real-Time Trace, RTOS-Event Trace

Static Analysis, Timing Analysis, Stack Analysis

Unit Test, Code Coverage

System-level Test

Test-As-A-Service

Verification-As-a-Service



PoE conformance



www.indes.com

info@indes.com

gerard@indes.com

