SW architecture impact on development, testing and certification of embedded wearable medical devices

- 19th FRUCT Conference, Jyvaskyla, 10-11-2016
- Michel Gillet
- Senior architect, embedded device
- Nokia Technologies, Digital Health

Outline

- Introduction and context
- SW architectures: Bottom up vs. Top down
- Abstraction Layers or xALs
- Benefits & conclusions

Introduction and context

Introduction

- Nokia, from rubber boots, to wood pulp, to mobile and what's next?
  - 25 April, 2014: Nokia completes sale of substantially all of its Devices & Services business to Microsoft (link)
  - 2014, Nokia Technologies is created
  - November 2014, first product Nokia N1
  - July 2015, launch of Ozo
  - April 26, 2016, Nokia acquires Withings
  - May 2016, HMD global Oy is created: it's a Finnish company developing mobile devices under the "Nokia" brand name
  - 31 May 2016, the Digital Health division in created in Nokia Tech
- My current role is to define the SW, HW & HW security architecture for a family of future digital health **medical devices**

Medical devices

- Medical means that those devices fall under medical regulations, i.e. FDA in US
- To simply, there are roughly 3 main classes of medical devices
  - Class 1: have the least regulatory control, because malfunction are not harmful to user, i.e. digital thermometer
  - Class 2: have mandatory performance standards, including proofs that a malfunction can't harm its user, i.e. infusion pumps
  - Class 3: have very strong regulatory control, usually devices supporting or sustaining human life, i.e. pacemakers
- For class 2 and 3, the manufacturer needs to provide guarantees that the device will perform according to its specification, it means that lots of extra work is required beside building a device : documents for risk analysis, mitigation strategies, well defined and documented product life cycle, etc.
- But there is also a potential emerging new market between current medical devices and consumer devices, mainly created because of IoT and ever increasing heath care costs

Wearable medical devices

- Small in size ⇒ small battery
- Reusable ⇒ rechargeable battery
- Low power ⇒ moves towards MCU
- Guarantee operation ⇒ moves towards RTOS
- Appealing design ⇒ usually contradicting mechanical requirements
- Connected ⇒ wireless connectivity is required
- Very high SW quality
- Measure vital signs
- Carried or on the body of the user
- Lower cost than equivalent fixed medical device ⇒ miniaturization, new business models, higher volume

Summary so far

- **Potentially a new huge market and lots of opportunities**
  - Same medical functions, but miniaturized, with significantly lower cost and lower power consumption
  - But still in a regulated space where many medical and safety standards need to be followed
  - Finite number of vital signs to measure
  - Health care costs are unsustainable
- Of course lots of companies will try to capture a sizeable part of that new market
- So how to differentiate from other players to have a bigger share of the market?
  - Could there be something which could be done already at the device SW architecture level? To reduce costs? Reach higher SW quality faster?
  - Yes, by allowing to push reuse further and the idea that “every line of code written counts”

Vision and requirements

- Products may cover a wide area from consumer to medical device, the architecture must then be able to support any (RT)OS, from very expensive medically pre-certified to cheap general purpose RTOS or OS
- The SW must support any HW partitioning, i.e. 1 or 2 MCU or more
- Writing and testing device driver for HW component and/or sensor is very costly and time consuming, we want to be able to write it once and reused it everywhere, meaning for all RTOS and all MCUs
- The device drivers must be independent of bus topology
- All or at least most unit tests can be reused without modifications across RTOS, MCUs, bus topologies, etc.
- The SW architecture must allow true continuous integration, i.e. Git/Gerrit/Jenkins, knowing that MCUs use flash which can be reprogrammed only a limited amount of times. By “true”, I mean that every single commit must be runtime tested
- Because of regulatory requirements, it must support static only memory allocation

SW architectures

Bottom up vs. Top down

Bottom up architecture

- In this context, a bottom up architecture means:
- The SW is build on top of the SW platform provided by the HW manufacturer and/or RTOS provider
- If the HW manufacturer and/or RTOS provider changes, the most if not the whole SW work is redone from scratch, including testing
- Likely, there is very little synergies between SW development and SW maintenance across device families and/or over the whole product range
- Minimal up front cost, since architecture is grown organically over time

NOKIA

Top down architecture

- In this context, a top down architecture means:
- We define the complete architecture directly from our requirements and needs
- This SW architecture is identical with every device
- The testing strategy is aligned with the SW architecture and is reused across all devices
- The SW architecture is strongly layered or using abstraction layers, and thus only about the bottom 10% are specific to the HW manufacturer and/or RTOS provider
- If the HW manufacturer and/or RTOS supplier changes, the overall SW impact is bounded and minimal
- There are very strong synergies between SW development and SW maintenance across devices
- There is an up front cost, since architecture must be designed first, but this anyway a requirement when going to regulated space and finally it must be implemented

Abstraction Layers or xALs

NOKIA

Basic functionalities of wearable medical device and its design

- Some kind of OS to provide the basic runtime features needed by applications, i.e. RTOS
- A SW library supporting all required wired communication interfaces, i.e. SPI, I2C
- A SW library for all wireless communication interfaces, i.e 3G, WLAN, BT, BLE
- A SW library to write unit tests
- A library defining all SW drivers and configuration for supported HW board
- Upon those, higher level functionality is build, i.e. to process sensor data
- We will name them as
  - RAL for the OS/RTOS, RTOS Abstraction Layer
  - PAL for the wired communication interfaces, Peripheral Abstraction Layer
  - CAL for the wireless communication interfaces, Connectivity Abstraction Layer
  - UTAL for the unit test framework/library, Unit Test Abstraction Layer
  - SAL Board for the board files defining the SW for one board

RAL

- Provides the basic functionality of an embedded OS or RTOS:
  - Thread, called Task thereafter
  - Synchronization functions: Semaphore, Mutex
  - Timing functions: Timer, getting/setting time, delays
  - Queues, etc.
- But also some additional functionality typically not found in RTOS:
  - To solve the “static initialization order fiasco”
  - Overall deterministic initialization of the whole system
  - Clear split between applications and middleware/drivers/board files:
    - Applications are defined by Tasks and one TaskMnger
    - The rest is defined by a System defined for a particular board in SAL Board
  - Fully written in C++ for maintainability and modularity, but also benefit of C++ templates
- Support virtual HW with implementation with Boost and SystemC

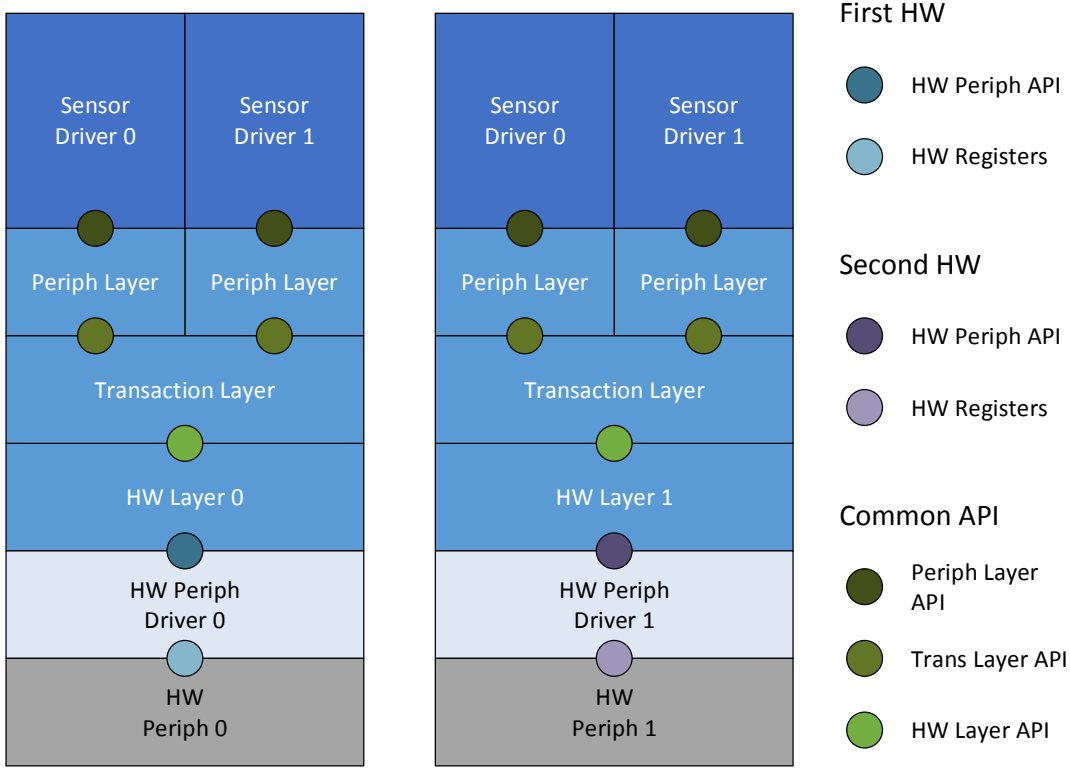
NOKIA

PAL (1/2)

- Provides access and support for all relevant wired interfaces:
  - SPI
  - I2C, including I2C multiplexers, IO expanders, E2PROM, etc.
  - UART
  - SSC
  - USB
- PAL is using a SW model based on the concept of Transaction and structured in 3 layers:
  - Periph Layer
  - Transaction Layer
  - HW Layer
- Fully supported also for virtual HW, allowing running unit test on a PC for testing a HW component or sensor

PAL (2/2)

- One device driver can be used unchanged regardless of :
  - the bus topology, requires a special SW layering
  - How HW peripheral are shared by drivers
- PAL is then divided in 3 “layers”
  - HW Layer
  - Transaction Layer
  - Peripheral Layer



NOKIA

CAL

- Provides access and support for all relevant wireless interfaces, including:
  - BT
  - BLE
  - 3G
  - LTE
  - WLAN
- Note that one fundamental architecture design of CAL is to abstract and support different architecture partitioning for the communication stacks
  - The TCP/IP stack can be offloaded to an external module or run on our own MCU, without impact on the application software
  - Same for the “TLS” stack, which could be offloaded or run on our own MCU
  - And pretty much any intermediate steps in between those extremes

UTAL

- Provides a platform independent library to write unit test
- The same unit test can then be run in ay virtual HW or real HW
- This simplifies and reduces significantly the effort needed to design and maintain a set of unit tests, which can then be used for continuous integration
- Furthermore, it increases overall the SW quality, since there can’t be porting mistakes or bugs between the same unit test running in a VHW or a real HW

- It’s implemented as a port of Boost unit test framework. Note this is based on an open source project

NOKIA

SAL Board

- Provides a library to define what is often called “board file” or BSP (Board Support Package), which bundles together
  - The OS/RTOS
  - Drivers for the MCU used and its peripherals
  - Drivers for all components found on the board
  - Etc.
- In this context, it basically bundles and instantiates RAL, PAL, CAL, etc. for a specific HW board, simplifying the process of writing application and/or unit test for this HW board

Benefits & conclusions

NOKIA

SW development

- For one project, we currently support 7 hardware boards, the same SW up to application is identical and running on all boards
- For every commit in Gerrit, a full list of unit tests are run in 2 different virtual environments (Boost, SystemC), allowing to catch bugs as early as possible
- All the application SW can be totally developed and tested on a PC using the virtual environment, which increase significantly productivity
- The testing strategy from virtual HW to HW is inherent part of the mitigation of risks and thus an essential for achieving medical certification
- Since most unit tests can be reused unchanged in the virtual HW and HW, there is a significant decrease of workload to maintain them, which essentially means their quality will be higher

Costs

- Under the assumptions
  - High reliability SW, for medical device or medical grade consumer device ⇒
    - Ratio of 1 SW developer to 3 to 5 SW testers (dependent of testing strategy)
    - Ratio of one QA engineer per 3 to 10 SW developer, for certification documentation
  - New MCUs every 6 to 12 months ⇒ likely that every product has different MCU/platform
- The needed SW workforce can be reduced by at least 50%, but up to 70%
- Assuming
  - 10k EUR per person/month
  - about 100 persons less over 2 years ⇒ 20 millions EUR
  - 1 millions device
  - Very aggressive sell price set at 3 times overall costs
  - The “small” up front cost of getting the xALs SW architecture is not taken into account
- The saving is about 60 EUR per device

Conclusions

- To be successful in this field, a company must get just right many aspects including engineering, marketing, business strategy, etc.
- But one essential aspect is very often overlooked, the overall architecture and specifically the SW architecture
- The SW architecture chosen drives, defines and impacts
  - The level of SW reuse cross HW, (RT)OS
  - The testing strategy
  - The process to achieve medical certification
- Choosing a given architecture can have a direct impact on the medical device costs, i.e. up to few tenth of EUR

NOKIA


NOKIA