Rapid IoT/Embedded Development Platform: OPEN-ARM

Ari E. Siitonen

ABSTRACT

OPEN-ARM is development framework for ARM Cortex-based IoT/Embedded products. It has Application-First approach: we define the requirements first and later decide what/which makes/models can be used. This way we are not locked to one chip-provider.

Furthermore, we can build easily concurrently Products with multiple ARM:s and be prepared for chip shortages. We are 100% manufacturer-independent and use only our own proprietary code, with zero dependency on manufacturer SDK:s.

1. PROBLEM STATEMENT

Current ARM based development usually starts with selection of ARM Chip (manufacturer and make). This leads to dependency to selected chip's development environment and libraries. Also the implementation will be using features specific to the chip.

This will cause problems when selected chip becomes unavailable or in-economical as switching to another chip at production phase can cause disruption in deliveries. To port the chip-bound code to new one will require a lot of testing and validation that takes time.

And after the switch the new chip can become extinct same way that the fist one did, unless one can afford to purchase stock to last a decade.

2. INTRODUCTION

Our Target: Maximal freedom from chip choice constraints and keeping options available.

Once the application requirements have been formally defined, OPEN-ARM starts by mapping them to all supported ARM MCU:s. The chip availability and price can be defined as one of the requirements. System tracks continuously major distributors for market data.

Designer can then pick one or more MCU:s, and continue to map pins to required functionalities. Supporting initialisation and API functions are generated for each MCU along with Schematic symbol with mapped pin-out.

3. DEVELOPMENT CYCLE

OPEN-ARM provides a platform and community to develop products based on ARM Cortex MCU:s.

3.1. *Cloud Compiler*

We maintain on-demand cloud gcc/clang-compiler environments and integrate with github/bitbucket CI-features to build Client's artefacts.

3.2. *VS-Code Integration*

Seamless VSC integration allows native interface for most of the developers. Along with ssh-style console connection to Device Under Development/Test.

3.3. *Artefact Storage*

Binaries and other build products can be stored and archived on our servers securely and deployed into products.

3.4. *Safety and Security*

OPEN-ARM can maintain master keys and serialise Client's products, or we can provide tools to do that in-house.

3.5. *Production*

OPEN-ARM platform includes economical hardware for flashing, serialisation and securing the devices during production. There is no need for j-link or similar third party tool (which of course can be used if wanted).

3.6. *Firmware Updates*

OPEN-ARM generates firmware update bundles with encrypted binary and signature.

The device will always keep the factory-default f/w as fail-safe: If updated f/w fails with watchdog event the default version is automatically taken in use. This way there is always a path for regression.

If configured so, previous version can also be retained as primary regression. And if that fails too, then factory-default is taken into use.

4. COMMUNITY

Everything cannot be generalised and some development tasks can be defined well enough to be outsourced. And OPEN-ARM allows it easily

4.1. *Out-Sourcing S/W Work*

As an OPEN-ARM user one can define inputs, outputs and functionality and make a Community Task Request:

Requirements: Client is responsible to give clear and understandable requirements and test cases. Fulfilment and passing tests will constitute accepted delivery and billing.

Security: The external developer will know nothing of the Client Project, only requirements and test cases are revealed. And a NDA is in effect.

Priority: Time frame and pricing depends on this.

Feasibility: Community can give vote before committing to the task on the feasibility of the task on the selected MCU(s).

License: Client can request Closed or Open Source license for the work. Also, a binary delivery is possible: then software stays on OPEN-ARM servers and is linked to client f/w.

Reviews: We can assist on reviews if requested by client.

Testing: OPEN-ARM will arrange test-setup and verify successful pass of tests defined by the Client.

4.2. *Out Sourcing H/W Work*

Also PCB and Schematic work can be outsourced once I/O and peripherals are defined.

Prototype: First prototype can be built from OPEN-ARM-development boards if necessary to speed up development. External peripherals will be emulated by our test-setup.

Schematics: can be drafted from Device Description either by us or Community member.

PCB: can be defined and designed once Schematics is done, also either by us or Community member.

5. TECHNOLOGY

OPEN-ARM provides fast and easy solution to IoT/ARM development, especially during the times of component shortage.

All internal and external interfaces are abstracted into a static vector with generalised API.

5.1. *Standard Cortex Features*

These features are common to all ARM makes and models and can be supported with our generic code. The module definitions themselves configure the device and will provide API functions to be used in the Application.

RTC: Real Time Clock support in included as standard.

WATCHDOG: Hardware watchdog is supported ensuring resilience of exceptions.

ENERGY: Various sleep modes and MCU clock frequency adjustments are supported and energy budget is provided. It can even be verified in test setup.

GPIO: General Purpose I/O: direction, open-drain, pull-up/down configuration and API for control is automatically provided per definition.

UART: Support for serial ports: RS232, RS485, etc. Polled, IRQ and DMA supported.

SPI: Synchronous Peripheral Interface: Industry Standard interface supporting both Master and Slave modes.

I2C: Inter-Integrated Circuit: Another Industry Standard interface also supporting both Master and Slave modes.

ADC/DAC: Analogue interfaces both way supported in limits of chip hardware. DMA too if available.

5.2. *Standard External Hardware Modules*

Our supported Externals Modules interface to MCU via UART/SPI/I2C.

LORA: Long Range Radio is supported with multiple vendor's modules.

NB-IoT: Narrow Band IoT is also multiple modules.

MQTT: MQTT can be supported both on LTE-module AT-command or as f/w MQTT implementation.

DEVICES: A plethora of common peripheral chips are supported. And simulation is available for them, so Application can be verified fully in testing.

5.3. *Standard Software Modules*

RT0S Provides almost Bare-Metal level minimal OS:

- Fully Static design.

  ⇒ Will not run out of memory or corrupt it.

  ⇒ Deterministic behaviour.

- No Open Source code

  ⇒ No Licences to track.

  ⇒ No overhead from bloated manufacturer SDK:s

- Minimal RTOS with optimal h/w-support→Less Code

  ⇒ Will fit on smallest FLASH/RAM, even on 4kb flash and 1kb ram.

  ⇒ Less Energy.

  ⇒ More MCU choices.

  ⇒ Faster and more reliable f/w updates.

  ⇒ Faster development cycle.

- Modular design:

  ⇒ During development just the new code module binary can be uploaded quickly to RAM and executed.

  ⇒ Installed device can be patched with new s/w modules incrementally.

TASKS: A non-preemptive scheduler is provided if main loop is not sufficient for the Application. Performance is monitored and MCU clock can be governed dynamically by load.

BUFFERS: Circular Buffers and FIFO:s as provided with simple API.

HEAP: All memory besides static variables and stack allocation is free to use by application as Heap-memory. Memory allocation scheme is static, so there is no garbage collection or running out of memory in run-time. Application initialisation should request all required blocks of memory from Heap.

STATE-MACHINES: Either with web tool or with JSON-description one can define multiple finite state machines in a standard fashion.

5.4. *Custom Features*

These features are so varied and sophisticated that they require custom code to be useful. But with the community of developers we can provide efficient and economical out-sourced solutions per Application Requirements.

TIMER: Timers with Input and Output Compares are building blocks for real-time control and also for measurements.

RADIO: Bluetooth/Zigbee style radio support is also quite complex and cannot be generalised.

5.5. *Web Admin and Apps*

OPEN-ARM maintains a web administration site for all Clients allowing configuration and monitoring of their fleet of devices.

Also we provide templates and libraries that provide tools for Clients to build custom web apps for their products.

5.6. *Mobile App*

Building on Capacitor Mobile App Platform OPEN-ARM provides suitable libraries to build custom Mobile Apps for client projects. Also standard debugging and configuration Apps are provided.

5.7. *Testing and Verification*

All standard modules are provided with p-tests, and our test-setup can simulate GPIO, ADC, DAC, SPI and I2C signals and emulate peripheral devices.

For custom modules the developer will build tests for defined coverage by Client.

OPEN-ARM maintains a farm of Devices Under Test (DUT), allowing tests to be run on Client Applications.

Robot Framework: library is available for testing, and it is able to remotely access test-setup at our premises.

Live Debugging: Our test-setup allows UART/SWD/JTAG remote and local debugging via API and web GUI. A API and MQTT interface is available for integration along with CLI-tools.

6. CONCLUSION

Product development with ARM Cortex does not need to be expensive, unpredictable and time consuming.

With proper tools and a supporting community soft- and hardware projects will become predictable