[SSC22-WKP2-16]

An Evaluation of Potential Compute Platforms for Picosatellites

Ofir Cohen and Sivan Toledo
Blavatnik School of Computer Science
Tel-Aviv University

ABSTRACT

What compute platform should picosatellites use? CubeSats, classified as nanosatellites, are transitioning from microcontrollers that cannot run modern operating systems and modern programming environments to Linux-capable compute platforms. As electronics continue to shrink, picosatellite missions are likely to become more common, perhaps using the PocketQube standard. This paper characterizes the requirements that compute platforms for picosatellites should satisfy and analyzes in detail 4 potential platforms. We show that suitable hardware does exist, but that it is not yet supported well enough to allow small teams to use it in satellites or other specialized sensor nodes.

1 INTRODUCTION

Picosatellites, satellites weighing 0.1–1 kg are on the rise. A standard called the PocketQube standard for such satellites is emerging.[18,23] The first batch of PocketQubes was launched in 2013 and two more were launched after a long hiatus in 2019 and 2021. Not many have been launched but this may change soon, reflecting the emergence of nanosatellites following the publication of the CubeSat standard in the early 2000s. Electronic components keep shrinking, allowing smaller satellites to perform certain missions.

The first CubeSats were controlled by simple microcontrollers (MCUs). Today, many CubeSats still use relatively weak processors and relatively primitive operating systems. Is this really necessary? No. At least one major vendor of CubeSat components sells an on-board computer (OBC) that runs Linux. We take the question one step further: can picosatellites be driven by compute platforms that can run a modern operating system like Linux? That is, are there Linux platforms that satisfy the requirements of a picosatellite, such as small physical size and low energy budget? After all, the smart watch that you may be wearing right now is mostly likely powered by a Linux or Unix operating system, it is far smaller and lighter than picosatellites, and it is battery powered.

The introduction of the Raspberry Pi and later the Raspberry Pi Zero signaled that for a vast range of instruments and products, microcontrollers (MCUs) no longer have significant advantages in terms of cost or physical size over platforms that run Linux. Many current MCUs offer reasonable amounts of flash and RAM and their operating systems support multithreading and multitasking. Still, platforms that run full-featured operating systems like Linux offer orders of magnitude more flash and RAM and are much easier to program (for example, a vast number of libraries are available and ready to use under Linux, whereas they would require porting to an MCU). The introduction of the Raspberry Pi brought down the cost of these fully-featured platforms to tens of dollars, sizes to 10-by-7 cm or smaller, and power consumption down to several watts. The Raspberry Pi Zero went even further, bringing the cost down to less than ten dollars and reducing the physical size and power consumption even further. More recently, Raspberry Pi platforms are available as *compute modules*, which are boards designed to be stacked on a carrier with custom electronics and interfaces. We refer to such modules as *systems on a module* (SOM).

The advent of smart watches, many of which are now also Linux or Unix based, implies that these full-featured operating systems can now run efficiently on battery-powered computers with a tiny physical size.

This paper investigates whether SOMs that run Linux can drive a picosatellite. Our analyses and conclusions are also relevant to other remote devices with similar physical constraints on size and power, and similar development constraints, namely small-volume devices that are designed and built by small teams. The restriction to small volume applications stems from the fact that for large-volume applications the answer is clearly "yes", given the success of smart watches. A good example of devices similar to picosatellites from this perspective is wildlife tracking devices. Most animal species are much smaller than humans so they can only carry a small tracking device, and the device must be powered by a primary battery or by harvesting energy, almost always from solar cells, much like a satellite.

The main contributions of this paper, beyond the

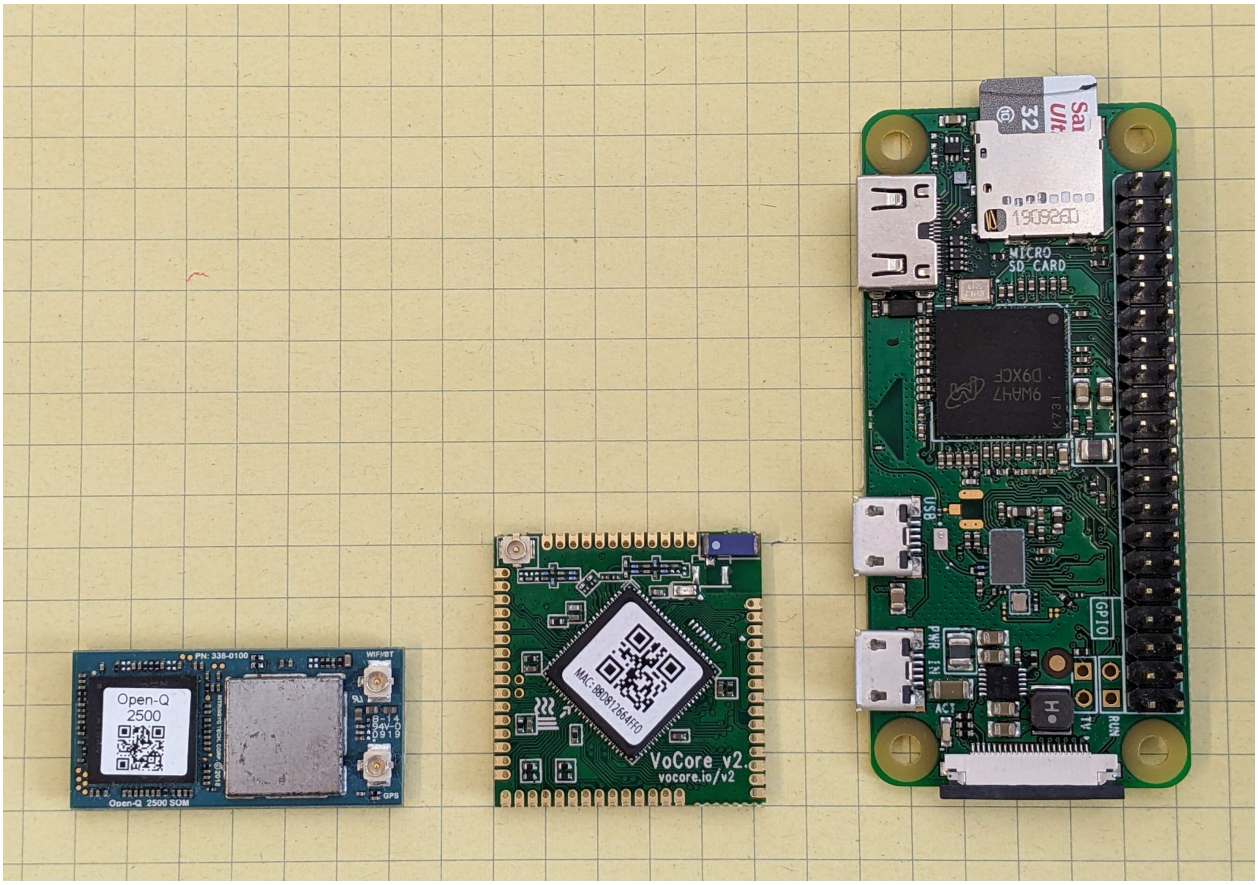

Figure 1: Three of the four compute platforms that we evaluated. From left to right: the OpenQ2500, the VoCore2, and the Raspberry Pi Zero. Lines in the background are 5mm apart.

background material in the next section, are:

- A detailed discussion of the requirements that compute platforms for picosatellites and similar projects must satisfy; this is presented in Section 3.
- A classification of potential compute platforms, starting from barefoot microcontrollers and ending with compute modules capable of running full-featured Linux distributions. This is presented in Section 4.
- An evaluation of 4 potential compute platforms relative to the requirements presented in Section 3. The evaluation, presented in Section 5, covers both quantitative aspects, such as physical size and power consumption, and qualitative aspects, such as support for developers and support for sophisticated power management strategies. Three of these modules are shown in Figure 1.
- Section 5 also constitutes an important higher-level contribution of this paper, namely, a methodology for testing future candidate platforms. This is particularly important since our conclusions from the evaluation, presented in Section 6, are that no existing SOM is appropriate for picosatellites, but that it is highly likely that in the next months or years suitable SOMs will emerge. All the platforms that we evaluated have crippling defects, but these defects are unlikely to last long. Hence, a methodology for testing them is important.

We note that in the past, similar investigations often used a completely different methodology, one in which researchers focused on one interesting compute platform, designed a working satellite, and launched it to fully test the platform.[11,24] This alternative approach can deliver definitive affirmative evidence that the platform is viable, but it is also narrower than our methodology (focusing on just one candidate platform rather than on all the viable ones) and that it cannot report on viability gaps and deficiencies. Both approaches clearly have value.

2 BACKGROUND

We frame our assessment of compute platforms using a set of concrete requirements that are presented in the next section. To justify the requirements, we provide in this section background on target applications, namely small satellites and wildlife trackers and loggers.

2.1 Nanosatellites and CubeSats

As the title of the paper states, we are interested in compute platforms for picosatellites, satellites weighing between 100 g and 1 kg (Table 1 names satellite size classes). But picosatellites are still very rare, so it makes sense to explore first the now-mature technology of nanosatellites weighing 1 to 10 kg, which are now common and relatively easy to design, build, and deploy.

Table 1: Satellite Classification by Mass

Satellite Class	Mass (kg)
large satellite	>1000
mini satellite (or small satellite)	100-1000
micro satellite	10–100
nano satellite	1–10
pico satellite	0.1–1
femto satellite (or satellite on a chip)	0.001—0.1

For decades, almost all useful satellites were very large. The first artificial satellite, the Russian Sputnik 1, launched in 1957, weighed a modest 83.6 kg (but was useful for nothing but demonstrating that a satellite can be placed in orbit and be tracked). Many subsequent satellites were much bigger. The first GPS satellite, Navstar1, launched in 1978, weighed 450 kg. Recent Israeli imaging satellites, Ofek 7, 9, 10, and 11, launched between 2007 and 2016, weigh between 300 and 330 kg. The European Earth-observing Envisat, launched in 2002, weighs 8211 kg (and cost 2.3 billion Euros). Amos 17, a geosynchronous communication satellite launched in 2019, weighs 6500 kg.

According to,[15] until about 2000, very few nanosatellites have been launched. A few have been launched by radio amateurs; for example, the first two, OSCAR 1 and OSCAR 2, weighed 10 kg each, but most subsequent amateur satellites were bigger. SNAP-1, designed in the UK and launched in 2000, weighed 6.5 kg. TUBSAT-N and TUBSAT-N+, designed by a German university and launched in 1998, weighed 8 and 3 kg, respectively. Munin, designed by a Swedish university and launched in 2001, weighed 6 kg. That's about it

A revolutionary idea, published in 2000,[13] led to a dramatic reduction in the cost and effort required to launch small satellites and consequently to a large number of launches of such satellites, as well as a large volume of research and literature. A group of university researchers noticed that most of the difficulty in designing small research satellites stemmed not from the mission's unique payload, but by the design and implementation of generic components, such as the on-board computer, the deployment mechanism, energy harvesting and power management, and so on. They proposed a standardized design of satellites measuring 10-by-10-by-10 cm or 20-by-10-by-10 cm or 30-by-10-by-10 cm (sizes commonly referred to now as 1U, 2U, and 3U). The standard, called the CubeSat standard,[21] specifies the physical dimensions and the mechanical interface to a simple deployer. CubeSat satellites are typically constructed from a stack of circuit boards housed in a 10-by-10 cm frame (so the boards are a little smaller than 10 by 10), with side panels covered by solar panels. Over time, commercial companies started offering complete or partial kits of the generic components of a CubeSat, including the mechanical frame, solar (side) panels and power management, an on-board computer (OBC), transceiver and antennas, and attitude control.

CubeSats often carry solar panels on all six sides. A 1U CubeSat in low Earth orbit harvests 1 W on average from its solar panels. This defines the power envelope for the satellite. The minimal weight for a 1U CubeSat is around 1 kg.

The CubeSat idea has been extremely successful. The first was launched in 2003, and the 100th was launched in 2012.[26] From 2003 to June 2021, a total of 246 1U CubeSats have launched into space, as well as many more larger ones.[16]

Importantly, while CubeSats were originally envisaged primarily as educational tools that allow students to design, deploy, and operate space missions, they have proved to be effective tools for scientific observation and experimentation.[20]

Several commercial companies design and sell CubeSat components and subsystems. The major vendors are ISIS (the Netherlands), Pumpkin Space Systems (USA), AAC Clyde Space (Sweden) and GomSpace (Denmark). The offerings from these vendors essentially allow customers to build a satellite from a kit; typically each satellite also includes custom electronics and/or custom software.

Table 2 documents compute boards that are currently offered today (2021) by several of these vendors. The compute platforms described in the table fall into three categories. In one, with an ARM9 processor and 64 MB of RAM, the platform has enough resources to run a modern operating system, Linux, but just barely. In the second we have motherboards that carry compute modules that run Linux and can

Table 2: Compute Platforms by Major CubeSat vendors

Vendor	ISIS (OBC)	Pumpkin (MBM2)	AAC Clyde Space (KRYTEN-M3)	GomSpace (NanoMind A3200)	SPUTNIX (SXC-MB-04)
Operating System	KubOS or FreeRTOS	KubOS (Linux)	RTEMS RTOS	FreeRTOS	Raspberry Pi
OBC CPU	400 MHz ARM 9	1GHz ARM Cortex-A8 + 2 specialized cores + Sitara GPU	50MHz ARM Cortex-M3	64MHz Atmel AVR32	1.2GHzh ARM Cortex-A53 + Cortex-M4 + ADCS CPU
Volatile Memory	64MB SDRAM	512MB DDR3 RAM	8MB MRAM	32MB SDRAM + 64KB SRAM	1GB SDRAM
Nonvolatile memory	1MB NOR + 512KB FRAM + 2 SD card sockets	4GB eMMC +SD card socket	4GB + 256KB	128MB NOR + 512KB	4GB + 256KB +2MB
Power	400mW average	Not specified	400mW average, 1W max	170mW average, 0.9W max	0.9W
Peripherals	I2C, SPI, UART, ADC, PWM; on-board sensors for voltages, RTC	UART, I2C, USB	I2C, SPI, UART, CAN, RS422, DTMF	I2C, UART, CAN, ADC, 9-axis IMU	I2C, SPI, UART, CAN, USB, camera
Extra features	Daughter board socket	Carrier board that a BeagleBone-Black board plugs into			Carrier board that a Raspberry Pi CM3 plugs into

also run software written in a high-level programming environment, say Python. In the third category we find 32-bit microcontrollers, like the ARM Cortex-M3 and an AVR32, which are incapable of running a modern operating system, and therefore incapable of running software written in many popular high-level environments.

Many CubeSats have used even weaker compute platforms, such as 16-bit micocontrollers like those from the MSP430 family.

2.2 Picosatellites

Continued miniaturization of electronic components and higher levels of integration led to the realization that useful satellites even smaller than CubeSats might be feasible.[7] The first batch of picosatellites were launched in November 2013.[27] These satellites, as well as later picosatellites, adhered to the PocketQube standard,[18] which defines a basic minimal size of 5-by-5-by-5 cm, called 1p, and multiples. The standard has been in development since 2009. The first batch included one 1p satellite, Wren, and a few that are not whole multiples (1.5p and 2.5p). The next batches were launched much later, in 2019 and 2021. The total number of launches is still small.

The volume and surface area of a 1p PocketQube satellite are 1/8 and 1/4 of those of a 1U CubeSat, so we expect its weight to be around 125 g and its power envelope to be around 250 mW. Larger PocketQube satellites, especially those with deployable external solar panels, can harvest more solar energy, perhaps as much as 10 W for a 3p satellite.

As in CubeSats, the PocketQube standard is expected to simplify the design and implementation of picosatellites.

Currently one vendor, Alba Orbital, offers a PocketQube kit.[1] The compute platform in the kit uses a 16-bit Texas Instruments TI MSP430 microcontroller. This on-board computer (OBC) is user programmable. The vendor provides a set of drivers for reading on-board sensors (temperature, an inertial measurement unit that includes an accelerometer, gyroscope, and magnetometer, and a real-time clock). The board relies on regulated 3.3V supply voltage.

2.3 Wildlife Tracking Devices

Wildlife tracking devices, usually called *tags*, range from sub-gram devices, obviously with very limited and focused functionality[10,17] all the way to devices weighing more than 10 kg that are used to track large mammals.[9] We focus on tracking devices whose physical characteristics are similar to those of picosatellites. The relevant characteristics for picosatellites include board sizes around 5-by-5 cm, weights of 250 g or less, and the ability to solar charge a battery. The inaccessibility of the device once launched or attached to a wild animal is another important characteristic; the devices must be highly reliable.

A typical modern device is the ES-400 from Cellular Tracking Technologies.[2] This tracking device comes in several sizes down to a 5-by-1.75-by-1.5 cm, 15 g tag. These tags have a multi-constellation GNSS receiver, a cellular modem compatible with 2G, 3G, and 4G networks, sensors (accelerometer and temperature sensor to characterize behavior as well as to detect illness or mortality). Slightly larger models also have a satellite radio that can report the location of the tag via ARGOS satellites. Other manufacturers produce similar products. Software or firmware for these tags is quite sophisticated, allowing users to define different behaviors for different geographical locations, times of day, etc.

Tags weighing 15 g are are applicable to animals weighing around 300 g or more (the generally acceptable limit is 3%–5% of body mass). Larger animals can carry heavier tags. For reference, a mallard weighs 700–1600 g, golden jackals weigh 5–12 kg, etc.

Tags that weigh a couple of grams or less, which are used on numerous species of small birds, bats, and terrestrial animals do not share many characteristics with picosatellites; they are much smaller than picosatellites and are often powered by a primary (non rechargeable) battery, requiring extreme efficiency, short activity periods and low duty cycles; these are difficult to achieve with anything but an ultra-low-power microcontrollers.

3 REQUIREMENTS FROM COMPUTE PLATFORMS FOR PICOSATELLITES

We now present the requirements that compute platforms for picosatellites should satisfy. Some of the requirements, such as physical size, are absolutely necessary, while others, such as support for high-level programming environments, are only desirable.

3.1 Physical Size

We are interested in satellites that are significantly smaller than 1U CubeSats. For example, so-called PocketQube satellites[22,23] start at 5-by-5-by-5 cm and use PCBs that are slightly smaller than 5-by-5 cm.

There are now numerous compute boards with comparable sizes. For example, a popular range of microcontroller boards called *Feather* boards[14] are

0.9-by-2 inches (about 2.3-by-5 cm). While a Feather board might not fit a PocketQube satellite as is, it is clear that it is possible to design a compute board for a PocketQube satellite using any of the microcontrollers that are used on Feather boards, including many ARM Cortex M0 and M4 microcontrollers, ESP32 microcontrollers, and so on.

3.2 Energy Efficiency and Power Management

A 1U CubeSat has a continuous power budget of about 1 W.[13] Satellites that are significantly smaller have smaller solar panels so they must operate at lower power budgets. For example, we expect a 5-by-5-by-5 PocketQube to generate no more than 0.25 W on average, since the area of the solar panels is about a factor of four smaller.

A power budget of 250 mW is very generous for most modern microcontrollers. For example, the RP2040 microcontroller from the Raspberry Pi foundation, consumes less than 100 mW (up to about 25 mA at 3.3. V). In sleep mode, this particular microcontroller consumes 0.39 mA on average (about 1 mW), so it appears to be efficient enough for a PocketQube.

Still, it makes sense to choose for a small satellite a compute platform that is as energy-efficient as possible. Fortunately, given the prevalence of battery-operated embedded systems today, there are many such platforms to choose from. These platforms (microcontrollers and sometimes entire compute modules) achieve their efficiency through four main mechanisms:

1. Sleep modes with very low-power consumption. For example, the Texas Instruments CC13XX microcontrollers consume only about 1 μA (about 2–3 μW in sleep mode, compared with the 1 mW of the RP2040).

2. Fast wakeup from sleep mode, to allow a low-duty cycle for a device that is in sleep mode much of the time to respond quickly to external events. For example, many MSP430 microcontrollers can transition in less than 6 μs from their deepest sleep mode to being full active.

3. Alternatively, low-power microcontrollers and microprocessors with slow wakeups offer mechanisms to avoid most wakeups. These can take the form of DMA controllers that can collect sensor data while the main CPU is in sleep mode, or even entire low-power processors that can perform simple (but programmable) tasks which the main CPU is in sleep mode. For example, the CC13XX microcontrollers contain both an ARM Cortex M3 or M4 CPU and a low-power processor called a *sensor controller* whose sole function is to prevent frequent wakeups of the ARM processor, since these wakeups are relatively slow and hence power inefficient.

4. An operating system that tracks which parts of the system must be active at any given time, including the processors, and which shuts down processors and peripherals that are not needed or puts them in sleep mode.

Computer systems that aim to achieve the lowest possible power consumption in sleep modes use only static RAM (SRAM), in which leakage current is low and which does not require active refreshing. SRAM is much less dense than dynamic RAM (DRAM), so systems that use only SRAM typically have less than 1 MB of RAM and are incapable of running modern full-featured operating systems. However, we shall see below that at power budgets of around 250 mW (and even significantly less), it it possible to use DRAM, including at sizes large enough to run Linux.

3.3 Software-Development Environment

Software productivity relies on the use of standard programming languages, application-programming interfaces (APIs), and software libraries. All three promote reuse and hence simplify the software development process and make it more efficient.

Software that is developed for a laptop or server can use a huge range of programming languages, ranging from low-level ones like C all the way to high-level ones like Python and Javascript. Such software normally runs on one of three operating system families: Linux, Windows, or MacOS (a variant of Unix). Such software can use a huge range of existing libraries and modules.

Therefore, the ability to run a standard operating system, especially Linux, provides huge benefits to developers. It makes it possible to construct very complex software from modules and libraries, without the need to port them or to reimplement their functionality.

On resource-constrainted compute platforms that cannot run Linux, three options remain:

1. Running barefoot, without an operating system. Usually such software is written in C. Microcontroller vendors often make available today device drivers, but their APIs are often vendor-specific.

2. Using an embedded operating system, like TI-RTOS from Texas Instruments or the portable

FreeRTOS. These provide more mature and complete support for applications written in C, but they normally still assume that the platform runs a single application. In general, very little third-party software libraries are available for these operating systems. Their APIs are sometimes based on standard APIs (e.g., the POSIX APIs that Linux also uses), but in general they are non-standard and porting any significant software to them is a major challenge.

3. Using a portable single-application high-level software environment, like the Arduino environemnt or CircuitPython. The Arduino environment is a set of simple C++ APIs for microcontrollers that were originally designed for a particular family of easy-to-use developement boards, but which have now been ported to a wide range of microcontrollers. CircuitPython is a restricted version of the Python programming language that has been designed for similar applications. Both of these make it easy to implement simple microcontroller projects, but neither is particularly good for complex projects.

3.4 Maturity, Support, and Ecosystem

Good support facilitates effective hardware and software development. On the hardware side, microcontrollers are open documented and supported by their vendor extremely well, perhaps because they need to support large number of engineers and developers. More complex processors and more specialized integrated circuits sometimes require special relations with the vendor to access all the documentation.

Software, including operating systems, varies much more widely. Some software is mature and very well supported. It is difficult to use software libraries and operating system when this is not the case.

Hardware and software form ecosystems. Some ecosystems, like that of Ubuntu Linux (a particular popular distribution of Linux), are vibrant and extensive. What we mean by that is that a lot of existing software has been ported and is well supported under these environments. This happens when a particular combination of hardware and software has many users and many active developers. When this is the case, the ecosystem also typically contains detailed documentation, many answered questions (on forums and questions-and-answers websites like stackexachange). On the other hand, the ecosystem of niche hardware or software is often much poor: libraries must be ported by the application developer, documentation is not available, and questions remain unanswered.

3.5 Compatibility with Harsh Environments (Space)

Space is a harsh environment. On the hardware side, components and subsystems must be able to operate in near vacuum, at extreme temperatures, and sometimes under intense radiation.

However, most CubeSats are in low-Earth orbits (LEO), inside the Van Allen belts, where the radiation is not particularly intense. As a consequence, many common microcontrollers and microprocessors are usable in CubeSats and smaller satellites.[12] Many CubeSats used off-the-shelf microcontrollers without even testing them for radiation. Also, most silicon chips, as well as many supporting electronic components, can operate in vacuum and in a wide enough range of temperature for LEO orbits. As a consequence, some CubeSats carried compute platforms that were not specifically designed for space, such as Raspberrry Pi computers[25] and smart phones.[24]

Therefore, we assume that this is not a serious issue and that a LEO-capable version of any common compute platform can be designed.

Space is also remote, which implies that software must be highly resilient. Mechanisms like watchdog timers can return a system gone wild to a safe state, but such mechanisms also pose a danger . In general, the more a satellite uses software from diverse sources, the more vulnerable it is to reliability problems. In that sense, a general purpose operating system may pose a danger, not only an opportunity.

3.6 Cost and Availability

CubeSat kits are fairly expensive. For example, the OBC offered by ISIS costs more than 4000 Euros . This is not terribly limiting, since launching the satellite is expensive. But expensive hardware makes it more difficult to use small satellites as educational tools, since it is impractical to buy multiple OBCs for students. In comparison, single-board computers that can run Linux are widely available for less than 50 Euros.

3.7 Summary

Compute platforms for satellites significantly smaller than CubeSats should be physically small, ideally 4-by-4 cm or smaller. They should be energy-efficient, but do not need to be extreme; a constant power consumption of a couple of milliwatts is good enough. Ideally, they should be able to run a full-

featured, widely-used, and well supported operating system, say a popular Linux distribution.

Once selected, the platform should be tested for space conditions, or redesigned for such conditions (e.g., replacing electronic components that might explode in low pressure with robust ones).

Wide availability and low cost are desirable but non-critical features. They are desirable especially where satellites are used in education. For educational uses, a low-cost easily available version of the platform is very useful even if it is not space qualified.

4 CANDIDATE PLATFORMS

4.1 Barefoot Microcontrollers

Many small satellites used microcontrollers without operating systems, just like a wide range of other products. Guertin et al.[12] surveyed the types of processors used until 2015 in low-budget CubeSat kits and missions. They list microcontrollers from the Texas Instruments MSP430 family, from the Microchip PIC24 and dsPIC33 families, which are all 16-bit microcontrollers that are usually programmed barefoot. They also list more highly capable 32-bit processors (an Atmel ARM9 processor, an Intel Atom, and a Qualcomm Snapdradon module), but it is clear that 16-bit barefoot microcontrollers have been used in CubeSats.

4.2 Microcontrollers with Specialized Operating Systems

Microcontrollers today are often too complex to program barefoot; at least a set of drivers or a hardware abstraction layer implemented by the vendor is used. Often, the silicon vendor also supports a minimal operating system. For example, Texas Instruments offers developers an operating system (including a full set of drivers) called TI-RTOS to many of its microcontrollers, including the CC13XX line, the MSP432, and more. TI-RTOS is a multi-tasking but non-preemptive operating system without hardware memory protection. Originally it used a specialized proprietary API, but in recent versions much of the functionality is accessible via a POSIX API (e.g., the POSIX threads API).

Similarly, Silicon Labs offers an operating system called Micrium OS for many of its microcontrollers. Micrium OS was a portable real-time operating system that was acquired by Silicon Labs.

Limited-functionality specialized operating systems are also used today on some MCU boards designed for cube sats. For example, the cube sat onboard computers made by AAC Clyde Space and by GomSpace both use a 32-bit MCU with a small amount of RAM (see Table 2). Both run specialized real-time operating systems.

4.3 Limited-Capability Linux Platforms (OpenWRT)

Physically-small computers capable of running Linux have been widely available for at least 15 years, but until the introduction of the Raspberry Pi, they were available almost exclusively as part of specialized products, initially small routers, and today also TV streamers and automotive infotainment systems. Over time, independent developers developed standardized and well-supported Linux distributions for these devices, the most popular one being OpenWRT.[5] This Linux distribution is supported on more than 1700 different hardware products (mostly small routers). It is memory efficient, requiring only 8 MB of flash and 64 MB of RAM (earlier versions required even less).

Complete products such as routers or streamers are difficult to repurpose as compute modules for satellites or other specialized uses.

Very few compute modules designed to serve as a part of more complex products and capable of running OpenWRT are available; in fact, we were able to find only one, named VoCore2.[3] However, it appears that designing custom printed circuit boards for embedded Linux systems is not terribly challenging.[8]

Some compute modules designed specifically for cube sats, like those produced by ISIS and Pumpkin (see Table 2) use CPUs in the same class as the VoCore2 module. The ISIS and Pumpkin modules can indeed run a Linux-based operating system called Kubos. It requires at least 64 MB of RAM, similar to OpenWRT.

4.4 Full-Featured Linux Platforms

There are now small single-board computers (SBCs) that can run a full-featured operating system. The most widely available and widely known of these is the Raspberry Pi family of SBCs, introduced in 2012. Current offerings include

- Raspberry Pi 4 SBC with a 4-core CPU and a range of physical ports (USB, Ethernet, HDMI, etc).
- Raspberry Pi 3 SBC with a weaker CPU and similar range of physical ports.
- Raspberry Pi Zero, a version of the Raspberry Pi 3 but lacking most of the physical ports, for use as a module within products. The Raspberry Pi Zero W adds Bluetooth and WiFi con-

nectivity and the recent Raspberry Pi Zero 2 W, with a stronger processor.

- Raspberry Pi 3 and 4 *Compute Modules*, which are versions of the 3 and 4 lacking completely physical ports but with board-to-board connectors. These are intended to be used as modules within products, like the Pi Zero, but with simpler manufacturing and stronger CPUs.

All versions are similar in terms of firmware and operating systems, but are obviously different in terms of power consumption.

The manufacturer, the Raspberry Pi Foundation, also maintains a Linux distribution for the computers, based on the Debian distribution. The hardware design is not open and its details are not published. However, the Foundation does provide detailed documentation of the platforms.

Raspberry Pis are inexpensive. The Pi Zero costs only 5 USD and the Pi Zero W costs 10 USD. Raspberry Pi Compute Modules start at about 25 USD.

Reliability, good support, and low prices made Raspberry Pis immensely popular. In March 2021 it was reported that 38 million units have been sold since 2012.[19] The large user base, which includes many hackers and makers, translates into excellent community support via on-line forums, web sites, and ported software.

Another platform with fairly similar capabilities and market is the BeagleBone board. It was originally offered by Texas Instruments as an educational tool, but is now available as a compute module. It has 512 MB of RAM and runs Linux. It is used as the on-board computer on cube sat kits offered by GOMSpace. It can run versions of Ubuntu or Debian Linux, and is also supported by Kubos.

A company called Compulab[4] offers a fairly large number of modules capable of running full-featured Linux distributions. The smallest one is 3-by-3 cm, has a dual- or quad-core ARM processor and up to 4 GB of RAM. The company provides a Debian Linux distribution for the board as well as support for the Yocto Linux project.

4.5 Platforms for Wearables (Android Wear)

The original iPhone, introduced in 2007, was the first Unix computer in the form factor of a smart phone. Earlier Unix or Linux computers came as servers, desktops, laptops, and specialized (but not physically small) SBCs. Android smart phones, introduced a year later, run Linux.

Wearable computers in form factors smaller than a phone initially used microcontrollers, but more recent wearables, and in particular smart watches, run Unix (Apple watches) or Linux (Android watches).

The computer in a smart watch is obviously physically smaller than that in a phone, so the existence of smart watches running Linux (and Android above the Linux operating system) led us to search for wearable platforms running Android.

It turns out that such platforms do exist and are available for purchase, but there is not a wide selection of them. We found only one vendor selling Android Wear modules (System on a Module or SOM) and development kits for them, Intrinsyc. During our research, Intrinsyc was acquired by Lantronix. Intrinsyc/Lantronics offers several SOMs and development kits for them. We chose the Open-Q 2500 SOM, which uses silicon made by Qualcomm, for our investigation. This platform is described more fully below.

5 EVALUATION

In this section we describe our evaluation of four candidate compute platforms for moderns pico satellites:

- A Raspberry Pi module, and more specifically, the Raspberry Pi Zero W.
- An Android Wear module, the Intrisic Open-Q 2500 SOM.
- A compute module intended for home WiFi routers and IoT devices, the VoCore2 module.
- A compute module intended for industrial computers and IoT devices, the Compulab MCM-iMX8M-Mini.

Our main findings appear to be applicable to other similar modules. In particular, the deficiencies we found in the Raspberry Pi Zero W are also present in other Raspberry Pis and the deficiencies we found in the Open-Q 2500 are also present in other Android Wear module by the same (and only) vendor.

The evaluation mainly focused on two aspects: power consumption and ease of development. However, we start with a detailed technical overview of the four platforms.

5.1 Technical Details of the Evaluation Platforms

The Raspberry Pi Zero W is a 65-by-30 mm SBC containing a Broadcom BCM2835 system-on-a-chip (also used in the Raspberry Pis 0, 1, 2, and 3). The chip contains a 1 GHz single-core 32-bit ARMv6 CPU, a GPU, and 512 MB of DRAM. The operating system and user data are stored on a MicroSDHC card. The board requires 5 V power at 1.2 A. The board also contains a wireless chip supporting

WiFi and Bluetooth (we did not use these), mini HDMI and USB ports (we did not use these), and a standard Raspberry Pi 40-pin expansion connector supporting GPIO, I2C, and SPI, all at 3.3 V.

The Open-Q 2500 is a 31.5-by-15 mm SOM containing a Quad-Core ARM Cortex A7 (32-bit) at 1.094GHz, a Qualcomm® Adreno™ 304 GPU, and a DSP processor, as well as 1 GB of LPDDR3 RAM and 8 GB of eMMC flash. It contains WiFi, Bluetooth, and GNSS radios but no antennas. It runs on 3.6 to 4.2 V. Two 100-pin connectors provide access to GPIO, UART, I2C, I2S (audio), and USB interfaces, as well as to a high-resolution display and a camera. It runs Android for Wearables version 8. The GPIO, UART, and I2C interfaces operate at 1.8 V.

We tested the module using the vendors Development Kit, a carrier board that provides power and battery management, a debug interface, antennas for the radios, and convenient physical interfaces (header pins) to many of the peripheral interfaces of the SOM. The carrier board also supports optional display and camera, which we did not use; we ran the SOM headless. The carrier board also provides a current-sensing resistor, to measure power consumption.

The VoCore2 is a 26-by-26 mm printed circuit board with components on both sides (so it cannot be soldered directly on top of another PCB). It contains a MediaTek MT7628 system-on-a-chip with a 580 MHz MIPS 32-bit single-core CPU and a WiFi radio. The board also contains a 128 MB low-power DDR2 DRAM chip and a 16 MB NOR flash chip. It supports OpenWRT versions 18 and 19. The MT7628 SOC is designed specifically for home-routers and for WiFi-based IoT devices. The module exposes through pads on its periphery SPI, I2C, I2S, USB (host), and PCI-Expresses buses, as well as JTAG pins, two UARTs, an SDXC card interface, and several Ethernet interfaces. It is relatively easy to integrate on a carrier PCB, easier than a Raspberry Pi compute module.

The MCM-iMX8M-Mini is a 30-by-30 mm printed circuit board. Like the VoCore2 is has components on both sides and is intended to be soldered to a carrier board with a square hole, to accomodate the components on the bottom of the SOM. It is based on a processor from the NXP i.MX 8M family. The board is available with 3 differnent dual- or quad-core processors and with 1, 2, or 4 GB of RAM and with a 4–64 GB eMMC flash memory. The module provides a wide range of high-speed interfaces (wired Ethernet, camera and display, USB, SDIO, etc) and low-speed interfaces (I2C, SPI, etc). It is available in standard, extended, or industrial temperature specifications.

The Raspberry Pi Zero W costs 10 USD. The Raspberry Pi Compute Modules, which are more suitable for integration in satellites and other products, are a bit more expensive, starting at 25 USD, but are still relatively inexpensive. The VoCore2 is also inexpensive, costing 18 USD in small quantities. The Open-Q SOM costs about 140 USD alone or about 700 USD with the carrier board. The cost of the MCM-iMX8M-Mini starts at 123 USD for a minimal configuration at small quantities (with large discounts for large quantities, up to a factor of 2.5); loaded versions tested for industrial temperatures reach prices of 448 USD for small quantities. The wide range of options and prices is a significant benefit.

5.2 Ease (or Difficulty) of Development

As stated above, Raspberry Pis are very well supported by their manufacturer and have a huge user base, including many hackers and developers. The level of support and documentation, the stability of the hardware and operating system, the quality of the Linux distribution, the availability of third-party software, and community support are all as good as they get. They are all fabulous. Developing software on Raspberry Pis is easy, can be done in a wide range of programming languages and environment. Access to low-level interfaces (GPIO, I2C, and so on) is standardized and easy.

Developing on the Intrinsyc SOMs is much more challenging. There is virtually no community support for these modules. There are no code samples. Documentation is often partial or difficult to understand. Updates to the Linux distribution and the Android system are infrequent.

For example, communicating with an I2C sensor proved to be a major undertaking on the Open-Q SOM. We tried to use the SOM as an I2C slave in order to let it communicate with a radio chip that could only be configured as an I2C master. It turns out to be impossible, but there is no way to know that from the documentation. We eventually managed to replace the I2C connection by a UART connection, but that too was very challenging. We did, however, develop several programs that run on the SOM and that read data from an accelerometer, gyroscope and temperature sensors over I2C, as well as a daemon that communicates with the radio chip over UART. So developing on the SOM is challenging, but certainly possible, including programs that access the hardware interfaces.

We eventually resorted to buying a 3000 USD support package, which allowed us to obtain answers from Intrisic's support engineers. This allowed us to resolve many of the outstanding issues that pre-

vented us from evaluating the platform, but development remained challenging.

While the Intrinsyc SOMs and the Qualcomm silicon are certainly usable, our evaluation is that they are not suitable for small development teams and for prototyping, even when the developers are experienced.

Developing on the VoCore2 is relatively easy. The board is easy to bring up, requiring only a 5V, 500mA supply (any USB port or power supply works) and a 3.3V USB-to-UART bridge, to connect to the board's Linux console. The board functions by default as a WiFi access point, so it is also possible to connect the network that it advertises and to connect to the board using SSH (so even the UART connection is optional, but convenient). C and C++ programs can be cross compiled on a desktop Linux computer on which the custom version of OpenWRT has been installed. Compiled programs can be transferred to the board using SCP and once there can be stored in a nonvolatile file system, along with any scripts or modified configuration files. The small amount of on-board non-volatile memory make it unlikely that the board can support full versions of high-level programming environments like Python or Java. Adding a micro-SD socket and a high-capacity memory card or an eMMC module might rectify this issue but we have not tested this. The relatively small amount of RAM might also make running high-level software challenging.

We have not tried to bring up the MCM-iMX8M-Mini, but since the vendor provides a standard Debian distribution for it, development should be easy. The module can boot from both the built-in eMMC storage, and from an external SD card, like a Raspberry Pi.

5.3 Power-Consumption Measurements: Methodology and Results

We measured the power consumption of the Raspberry Pi by feeding it with power throught a 0.1 Ohm resistor and measuring the voltage across the resistor with a PicoScope 3406D digital oscilloscope connected to a laptop running Windows. The resolution of the PicoScope is about 1.85 mV, which translates to about 19 mA or around 100 mW. We used this setting to measure power consumption in active mode, idle mode, and during boot. We used another setup described below to measure power consumption during sleep mode.

During these experiments, we disabled USB, Bluetooth WiFi, and HDMI. We disabled HDMI using the `tvservice` utility. We disabled WiFi and Bluetooth via `/boot/config.txt` and via `systemctl`. We disabled USB through `sysfs`.

We measured consumption in idle and active mode for about 50 s each. In idle mode the Raspberry Pi only ran background tasks. In active mode, we ran a program called `stress-ng` with its default parameters, which computes FFTs on the ARM CPU.

To measure boot time, we wrote a program that sets a GPIO pin which the Pi ran automatically after startup. This allowed us to determine on the oscilloscope when the boot process ended.

The operating system of the Rasberry Pi does not support sleep mode, but the hardware does. We tested this mode using a community-contributed firmware called `rpi-open-firmware`,[6] which replaces Broadcom's standard firmware (bootloader). We modified the firmware so that the CPU and the VPU (a SOC management CPU) enter sleep mode almost immediately. To gain more resolution in power measurements in this mode, we used a 2 Ohm sense resistor instead of the 0.1 Ohm resistor we used to measure power in active and idle modes and during the boot process.

Table 3: Power consumption of the Rasberry Pi Zero W (with its wireless and display interfaces turned off).

Mode	Mean±Std (mW)	Peak (mW)	
Sleep	33 ± 16	127	
Boot	3024 ± 1116	9879	39 s
Idle	836 ± 121	2233	
Active	3138 ± 1068	7974	

The results of the power measurements on the Raspberry Pi are presented in Figure 2 and table 3.

The SOM has a lower power consumption, which required a higher resolution power measurement. We measured current consumption (at the 4.0 V supplied by the carrier board) using a 0.5 Ohm resistor and a 100x voltage amplifier whose output was measured by the same PicoScope. The amplifier used a Texas Instruments INA169 amplifier on a Sparkfun breakout board.

We disabled the WiFi, Bluetooth, NFC, GNSS, on the SOM using ADB (Android debugger) commands. The display interface is active but since no application uses it, it turns the display off and presumably consumes little or no power.

We used the same benchmark, `stress-ng` (running under Linux, not under Android), to test active mode, using either 2 or 4 cores. We also used a similar mechanism as on the Raspberry Pi to determine when the boot process ended. The results are shown in Figure 2 and Table 4.

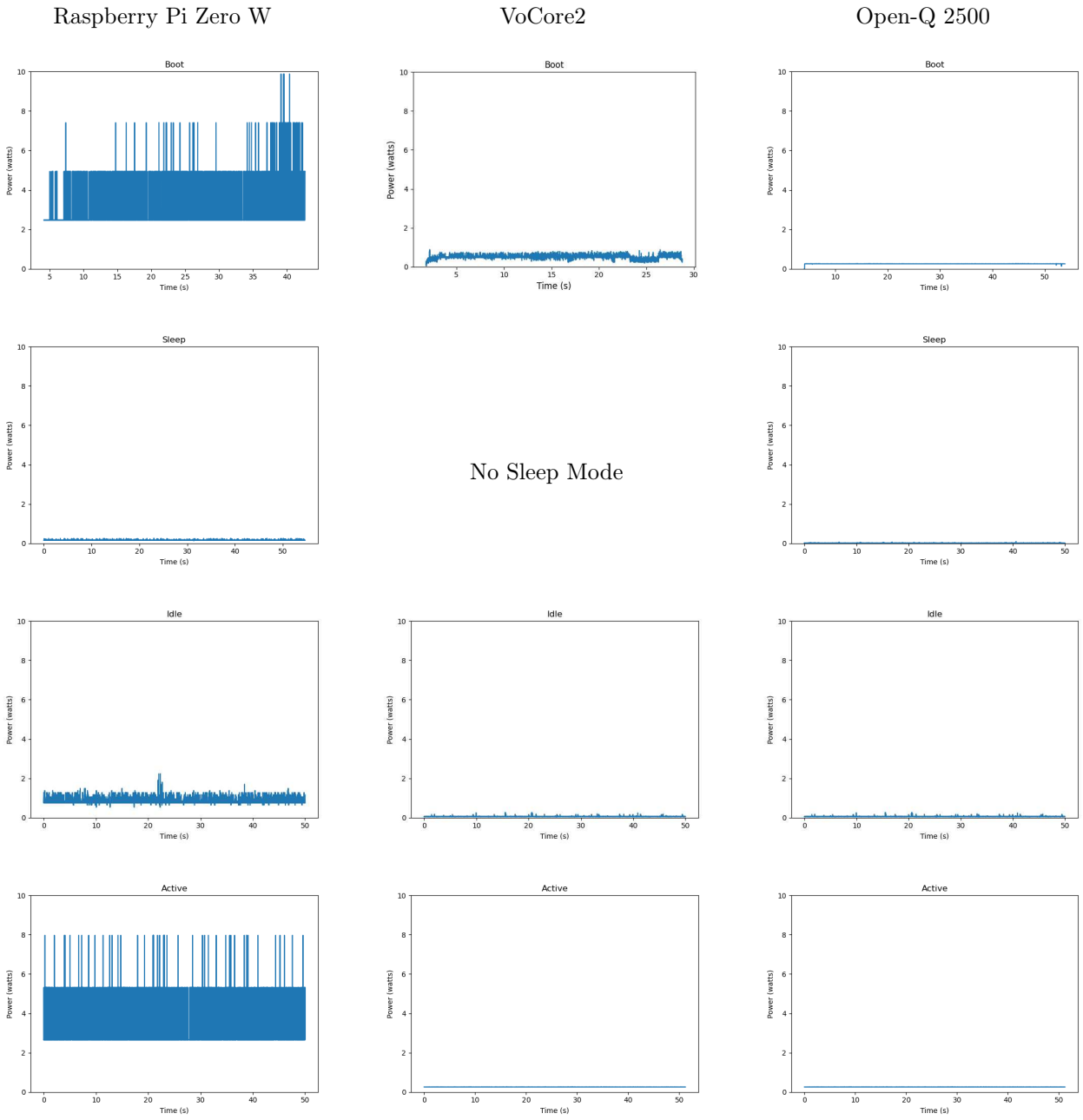

Figure 2: Instantaneous power consumption of a Raspberry Pi Zero W, an Open-Q 2500 SOM, and VoCore2 in different activity modes. The The SOM was tested in active mode using all four cores.

Table 4: Power consumption of the Open-Q 2500 SOM (with its wireless and display interfaces turned off).

Mode	Mean±Std (mW)	Peak (mW)	
Sleep	4 ± 7	90	
Boot	255 ± 4	258	50 s
Idle	54 ± 16	258	
Active	258 ± 1	262	

The VoCore2 has a high power consumption. We measured current consumption (at the 5.0 V supply) using an 0.2 Ohm resistor and a 25x voltage amplifier whose output was measured by the same PicoScope. The amplifier was again a Texas Instruments INA169 amplifier on a Sparkfun breakout board.

We found no way to disable power-hungry peripherals (USB, Ethernet, and WiFi) that are not needed for our intended applications and we found no way to put the module into low-power sleep mode. We verified with the manufacturer that they indeed

do not support disabling peripherals and sleep mode.

We used the same benchmark, `stress-ng`, to test active mode. The results are shown in Figure 2 and Table 5.

Table 5: Power consumption of the VoCore2 module.

Mode	Mean±Std (mW)	Peak (mW)	
Sleep	N/A	N/A	
Boot	505 ± 109	881	27 s
Idle	605 ± 175	2550	
Active	844 ± 200	2690	

We did not test the current consumption of the MCM-iMX8M-Mini, but the manufacturer provides detailed current consumption data for several operational modes, listed in Table 6.

Table 6: Power consumption of the MCM-iMX8M-Mini SOM.

Mode	Manufacturer Data (mW)
Sleep (DRAM refresh)	15
Idle, Ethernet & display disabled	240
GPU load	1760

6 CONCLUSIONS

Did we find a compute platform with a full-featured modern operating system supporting modern programming environments that is suitable for pico satellites? No.

Our results, summarized in Table 7, indicate that from the power perspective, Android Wear SOMs are terrific. They have a low sleep power consumption (around 4 mW peak) and consume only about 250 mW in active mode. This level of power efficiency should allow them to stay active all or almost all the time even in 5-by-5-by-5 cm pocket cubes, and to effectively exploit low duty-cycle work loads. The modules run Android on top of Linux, so in principle they support high-level programming (e.g., Android apps and many programming languages running directly on top of the Linux operating system). They are also tiny.

On the other hand, support for these modules is poor. Developing with them presents many obstacles that are difficult to overcome given the poor support and the essentially non-existent developer community. Also, software support on these modules targets Android exclusively, so other popular programming environments, such as Python or Node.JS are probably not supported out of the box.

The MCM-iMX8M-Mini SOM also has effective power management. It can probably be used in a nanosatellite or picosatellite, given the 15mW power consumption in sleep mode, but under heavy computational loads it consumes more than 1.7 W, requiring careful consideration of energy consumption even on a cube sat. However, this module uses an NXP processor from the i.MX 8M family that is not the best match for battery-powered devices. NXP has announced an ultra-low power sub-family, the i.MX 8ULP family, with lower power consumption. We expect that SOMs based on it will have power consumption similar to that of the Open-Q 2500.

We have not tested the MCM-iMX8M-Mini SOM ourselves, but given that the manufacturer supports both Debian and Yocto Linux on it, software development should not be too challenging.

The relevant characteristics of the Raspberry Pi are the exact opposites. The Pi excels where the SOM struggles, and in particular in ease of software development, a huge spectrum of sofware and programming environments, and excellent vendor and community support. Simply put, developing software on the Raspberry Pi is easy. On the other hand, the Raspberry Pi consumes way too much power and lacks even minimal support for sleep modes or low-duty cycle operations. It consumes between 3 and 10 W in active mode. This might have been acceptable for a 250 mW satellite if it was possible to put the Pi in sleep mode most of the time, but this is not possible. In idle mode, when the Pi runs only background tasks (the lower useful power mode available) it still consumes almost 1 W.

Both power and support issues render the VoCore2 module unsuitable for our purposes. It is power hungry. Not as much as a Raspberry Pi, with peak power consumption about 2.7 W, not 8 W, but it also lacks effective power management, so it consumes around 0.6 W even when it needs to do nothing. We found support from the vendor weak (e.g., we were able to get no assistance on the power management question), and we did not find any active community of users and developers.

It appears that we must wait for somebody to produce a low-power, tiny (3-by-1 cm or smaller) compute module running Linux that is well supported. Our results show that this is certainly possible. But we have not found one yet.

APPENDIX
DETAILS OF A SOM+RF MCU PLATFORM

We evaluated the ease of development with the compute platforms, and in particular on the OpenQ 2500 that we initially perceived as the most

Table 7: Relevant characteristics of potential compute platforms for picosatellites. The first two characteristics are absolutely essential, the rest are not.

	ARM Cortex M4 + Arduino/CircuitPython	ARM Cortex M4 + Proprietary OS	Raspeberry Pi Compute Module (3 or 4)	Android Wear SOM	OpenWRT on VoCore2	MCM-iMX8M-Mini
small physical size	+	+	±	+	+	+
ultra low power	+	+	−	±	−	±
COTS hardware	+	+	+	±	+	±
COTS software	+	−	+	+	+	+
technical support	+	+	+	−	+	?
ease of HW integration	+	+	±	+	+	+
resources (RAM, CPU)	−	−	+	+	±	+
high-level language	+	−	+	+	±	+
high-capability software	−	+	+	+	+	+
(high-speed interfaces)	−	−	+	−	+	+
cost	$	$	$$	$$$	$	$$$

suitable for picosatellites, by interfacing the SOM to a radio platform suitable for a picosatellite. The SOM, like any module designed for wearables, only has short-range radios, namely Bluetooth and WiFi. A picosatellite needs a long-range radio. We chose for the radio a modern MCU with a data radio transceiver, the CC1352P. This chip from TI has an ARM Cortex-M4 processor and a data radio capable of transmitting at up to 100mW at both sub-GHz frequencies (including the VHF and UHF bands typically used for CubeSats) and at 2.4 GHz. The radio can operate at both high-throughput short-range modes and at low-throughput long range modes. The long-range modes use a combination of narrowband modulation (to improve the signal-to-noise ratio), spreading to improve symbol recognition, and error-correcting codes.

We designed a software architecture to pair the SOM with a CC1352P and tested it using a CC1352P evaluation board. The architecture used the CC1352P as a simple radio peripheral, since we aimed to assign sophisticated processing to the SOM, which in principle is much easier to program (supports high-level languages, libraries, etc.).

Architecturally, the firmware of the CC1352P can perform exactly two tasks that are controlled by the SOM:

- Transmit a radio packet, and
- Enter receive mode for a given amount of time. If a packet is received within that time, it is sent back to the CC1352P for processing.

When the CC1352P is doing neither of these things, it enters a deep sleep mode. The SOM can wake it up to perform one of its tasks. Packets are opaque to the CC1352P; it does not interpret them in any way, only passes along valid packets. Treating packets as opaque implies that the high-level communication protocol is implemented by the SOM, not by the RF MCU.

We implemented the firmware on the CC1352P using C and the vendor's proprietary operating system called TI-RTOS. The implementation uses four threads that communicate through semaphores:

- A main thread that sets up the hardware, creates the other threads, and exits.
- A radio transmit thread, which waits for a packet from the SOM and tries to transmit it.
- A radio receive mode that receives a command to start receiving from the SOM, executes it, and sends back the received packet, if any.
- A service task that controls two LEDs on the evaluation board for diagnostics.

The SOM performs all other tasks, including collecting data from sensors (we connected an I2C

temperature sensor to the SOM's carrier board to test this functionality), waking up periodically using a real-time-clock (in the satellite, wakeups would be scheduled for sensing, attitude control, and for communication with ground stations), and attempting to communicate with ground stations using the CC1352P.

We explored two possible mechanisms for implementing the communication channel between the SOM and the CC1352P. We first tried to implement the channel using an I2C bus, which both devices support. However, the CC1352P does not support well slave mode and the SOM does not support slave mode at all. Since one of the two would have to be the slave in the channel, I2C proved to be a nonviable option. (The hardware of the CC1352P does support slave mode, but not in low-power sleep modes, and not in the high-level driver.)

We therefore switched to a UART implementation. This proved easier and we managed to get it to work. The UART on the CC1352P also does not work in low-power modes, so the prototype implementation only uses the UART for communication when both modules are awake. To wake the CC1352P, the SOM uses a general-purpose input-output (GPIO) pin. This was relatively easy to implement and it works well. Waiting for the edge on that pin was implemented using a software interrupt routine.

The software that we wrote for the SOM side of the prototype includes one Android system application written mostly in Java. Because we configured it as a system application, as opposed to a user application, it runs nonstop. The application includes three components, a scheduler, a handler for sensor measurements, and a radio-message controller. The sensor handler uses a built-in Android service called SensorManager to configure sensors and collect data from them.

Our Android application also contained C++ code, accessible to the Java parts through JNI (the standard Java-native interface). This code communicates with the UART, which is exposed in Android as a Linux character device (`/dev/tty`). On the other hand, we were able to control the GPIO pin directly from Java.

Unfortunately, user-mode Android code cannot use the I2C buses on the SOM. Therefore, to use an I2C sensor (I2C is very common today in sophisticated sensors) we had to write a Linux kernel module that implemented the low-level communication with the sensors and that generates standard sensor-measurement messages that eventually reach the SensorManager running in user space.

We tested the completed prototype and verified that it can communicate with another CC1352P evaluation board.

EVALUATION

We believe that the overall architecture is sound. There are now many more radio data transceivers that are part of RF MCUs than standalone transceivers, so it is likely that future picosatellites will need to use RF MCUs as radios. Our prototype also verified the feasibility of keeping most of the sophisticated functionality on the SOM, not on the harder-to-program MCU.

Unfortunately, the version of Android used on the SOM (Android 8.1.0 for Wearables) does not allow high-level Android applications to access all the hardware interfaces, and in particular the I2C interface. This implies that I2C communication must be implemented in a kernel module, one of the least hospitable and most complex programming environments in existence. In contrast, on Raspberry Pis I2C and other buses are easily accessible from user space code written in a variety of programming languages (including, for example, Python or Java).

But this was not the most difficult aspect of implementing the prototype. That was the lack of documentation and examples for all the unique aspects of the SOM (that is, all the hardware and software aspects that are not part of the base Android system). This made development difficult and required purchasing an expensive support package from the manufacturer (3000 USD, much more than the cost of the evaluation platform). Even with the paid support package, development remained challenging.

Acknowledgements

This study was supported by a grant from the Israeli Space Agency (part of the Israeli Ministry of Science) and by grant 1919/19 from the Israel Science Foundation.

References

[1] Alba orbital. PocketQube platform vendor at `http://www.albaorbital.com`, accessed March 2022.

[2] Cellular tracking technologies. Wildlife tracking vendor at `https://celltracktech.com`, accessed March 2022.

[3] Coin-sized Linux computer with WiFi and Ethernet. `https://vocore.io`, accessed March 2022.

[4] Compulab. System-on-a-module (SOM) vendor at `https://www.compulab.com`, accessed March 2022.

[5] OpenWrt project. `https://openwrt.org`, accessed March 2022.

[6] Kristina Brooks. Minimal Raspberry Pi VPU firmware. Source code and documentation at `https://github.com/christinaa/rpi-open-firmware`, accessed March 2022, January 2019.

[7] C. Cappelletti. Femto, pico, nano: overview of new satellite standards and applications. In *Advances in Astronautical Sciences: Proceedings of the 4th IAA Conference on University Satellite Missions and CubeSat Workshop*, volume 163, pages 503–510, 2018.

[8] Jay Carlson. So you want to build an embedded Linux system? Blog post at `https://jaycarlson.net/embedded-linux/`, accessed 4 January 2022; date infered from comments, October 2020.

[9] Frank .C. Craighead, Jr., John J. Craighead, Charles E. Cote, and Helmut K. Buechner. Satellite and ground radio tracking of elk. In S. R. Galler, K. Schmidt-Koenig, G. J. Jacobs, and R. E. Belleville, editors, *Animal Orientation and Navigation*, number 262 in NASA Special Publication, pages 99–111. 1972.

[10] Z. D. Deng, T. J. Carlson, H. Li, J. Xiao, M. J. Myjak, J. Lu, J. J. Martinez, C. M. Woodley, M. A. Weiland, and M. B. Eppard. An injectable acoustic transmitter for juvenile salmon. *Scientific Reports*, 5(8111), 2015.

[11] Dirk Geeroms, Sabine Bertho, Michel De Roeve, Rik Lempens, Michiel Ordies, and Jeroen Prooth. ArduSat, an Arduino-based CubeSat providing students with the opportunity to create their own satellite experiment and collect real-world space data.

[12] Steven M. Guertin, Mehran Amrbar, and Sergeh Vartanian. Radiation test results for common CubeSat microcontrollers and microprocessors. In *IEEE Radiation Effects Data Workshop (REDW)*, pages 1–9, 2015.

[13] Hank Heidt, Jordi Puig-Suari, Augustus S. Moore, and Shinichi Nakasuka. CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation. In *Proceedings of 14TH Annual Conference on Small Satellites*. 19 pages.

[14] Adafruit Industries. Feather specification. Retrieved September 9th, 2021.

[15] Herbert J. Kramer and Arthur P. Cracknell. An overview of small satellites in remote sensing. *International Journal of Remote Sensing*, 29:4285–4337, 2008.

[16] Erik Kulu. Nanosats database. `https://www.nanosats.eu`, accessed on August 5, 2021.

[17] B. Naef-Daenzer, D. Früh, M. Stalder, P. Wetli, and E. Weise. Miniaturization (0.2g) and evaluation of attachment techniques of telemetry transmitters. *The Journal of Experimental Biology*, 208, 4063–4068.

[18] Alba Orbital, Delft University, and GAUSS. The pocketqube standard, issue 1, June 2018.

[19] Avran Pitch. Raspberry Pi's ninth birthday: 9 things you might not know. Tom's Hardware, `https://www.tomshardware.com/news/raspberry-pi-9th-birthday`, March 2021.

[20] Armen Poghosyan and Alessandro Golkar. CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions. *Progress in Aerospace Sciences*, 88:59–83, 2017.

[21] The CubeSat Program. CubeSat design specification. Technical Report Revision 13, California Polytechnic State University, 2014.

[22] S. Radu, M.S. Uludag, S. Speretta, J. Bouwmeester, A. Menicucci, A. Cervone, A. Dunn, T. Walkinshaw, P.L. Kaled Da Cas, C. Cappelleti, and F. Graziani. PocketQube Mechanical Interface Standard, 2018.

[23] Silvana Radu, Sevket Uludag, Stefano Speretta, Jasper Bouwmeester, Eberhard Gill, and Nikitas Chronas Foteinakis. Delfi-PQ: The first pocketqube of Delft University of Technology. In *Proceedings of the 69th International Astronautical Congress*, pages 1–10, Bremen, Germany, 2018. International Astronautical Federation.

[24] Alberto Guillen Salas, Watson Attai, Ken Y. Oyadomari, Cedric Priscal, Rogan S. Shimmin, Oriol Tintore Gazulla, and Jasper L. Wolfe. PhoneSat in-flight experience results. Conference Paper ARC-E-DAA-TN14625, NASA, 2014. Presented at the Small Satellite Systems and Services (4S) Conference.

[25] Jeremy Straub, Christoffer Korvald, Anders Nervold, Atif Mohammad, Noah Root, Nicholas Long, and Donovan Torgerson. OpenOrbiter: A low-cost, educational prototype cubesat mission architecture. *Machines*, 1(1):1–32, 2013.

[26] Michael Swartwout. The first one hundred CubeSats: A statistical look. *Journal of Small Satellites*, 2(2):2013–233, 2013.

[27] Wikipedia. Pocketqube. `https://en.wikipedia.org/wiki/PocketQube`, accessed March 2022.