Applicability of commodity, low cost, single board computers for Internet of Things devices

Steven J. Johnston*, Mihaela Apetroaie-Cristea, Mark Scott and Simon J. Cox,
Faculty of Engineering and the Environment,
University of Southampton, UK.
Email: *sjj698@zepler.org

***Abstract*—We can expect the number of Internet of Things (IoT) devices to rapidly increase, in part due to the availability of low cost powerful hardware. These advances in hardware introduce a new class of computer – the commodity low-cost Single Board Computer (SBC); for example the Raspberry Pi. These devices are capable of running a full operating system and are accessible to non-technical users.**

In this paper we demonstrate the feasibility of using this class of computer to construct IoT devices without deep technical knowledge of embedded systems. Our IoT device is targeted at real world data collection and is the basis of a deployed device.

We discover that although the existing SBC hardware on the market varies dramatically, much of it is very applicable to IoT scenarios. Generally their default out of the box configuration supports one or more Linux variants, high-level programming languages and standard hardware libraries. They are very configurable and attractively priced – we predict a growth in using this class of computer particularly for IoT prototypes; disposable compute or low hardware volume scenarios.

In the future we can expect hardware improvements and enhanced operating systems taking into account the issues surrounding IoT devices, such as SD card storage corruptions, power consumption and improved failure detection.

***Index Terms*—Weather; IoT; low-cost; commodity; SBC;**

1. Introduction

Ubiquitous Internet enabled embedded systems, often referred to as the Internet of Things (IoT) is experiencing rapid growth. Some sources predict the number of IoT devices will exceed 26 billion by the year 2020 [1]; this number excludes PCs, smart phones and tablets. Many of these devices will be mass produced, however, there will be a subset of the IoT devices, designed by non-technical users, partly due to the availability of low cost hardware. The emergence of the low cost Single Board Computer (SBC) has been accelerated by a series of Raspberry Pi devices, currently sold in their millions. There are many similar boards on the market, for example Odroid, Udoo Neo, PINE64 [2] [3] [4], and newer hardware is being released continually. These low cost devices have a rich set of features and offer a powerful compute capability. The popularity of these devices has effectively introduced a new class of computer- the commodity low-cost SBC. The SBC is a complete, miniature computer,for example with sockets for an external display, network, expansion capabilities via USB and PCB headers for external circuit boards or sensors. This can be compared to other IoT devices which are often custom-built for the task they are performing.

The aim of this paper is to investigate the applicability of SBC's for IoT devices, focusing on the build complexity and technical skill-set required. We will use an exemplar weather monitoring IoT device, capable of gathering data from multiple sensors and transmitting it to an online cloud-based application; the Met Office WOW citizen science project[5].

The proposed device is built from currently available, commodity, low-cost hardware, suitable for mass deployment, and is programmable in a mainstream high level programming language.

2. Background

The concept of having all the peripherals on a single board, resulting in a SBC is not new [6] and many microcontrollers are embedded into SBCs, often referred to as development boards. For example the Arduino UNO is a development board for the ATmega328 [7]. The Arduino platform offers a great deal of libraries for a wide range of hardware but ultimately the CPU and processing capabilities combined with the lack of a full TCP/IP stack make it unsuitable for this particular IoT exemplar.

More powerful microcontrollers exist and are often referred to as a System on a Chip (SoC) although the distinction between them is blurred. The term microcontroller often refers to low memory, low performance single chip devices whereas a SoC usually supports an operating system such as Linux or Windows; although this is not a definition.

The plummeting cost of powerful SoC devices make them a popular alternative choice to a basic microcontroller. Table 1 shows a selection of common SoC processors.

In this paper we focus only on SBCs that are capable of running a fully featured operating system (OS). The main

TABLE 1. EXAMPLE SYSTEM ON A CHIP (SOC) HARDWARE

SoC	Cores / Clock	GPU	I/O
S805	4 x 1.5 GHz	ARM Mali-450	ADC, GPIO, SPI, I2C, UART, USB, DDIO/SD/MMC, PWM, SDXC/SDHC/SD, I2S, SPDIF, HDMI, PCM, Ethernet
BCM2837	4 x 1.2 GHz	Broadcom Video-Core IV	GPIO, SPI, I2C, UART, USB, CSI, DSI, PWM, PCM/I2S, DMA, Timers, Interrupt Controller
AM3358	1 x 1 GHz	SGX530	ADC, GPIO, SPI, I2C, UART, CAN, USB, SDIO/SD/MMC, McASP RTC.
i.MX 6Solo	1 x 1 GHz	Vivante GC880 for 3D and Vivante GC320 for 2D	ADC, GPIO, SPI, I2C, UART, USB, ESAI, I2S/SSI, Ethernet, FlexCAN, NAND Cntrl, PCIe, MIPI HSI, S/PDIF Tx/Rx, HDMI, LVDS
Exynos 5422	4 x 2.1 GHz and 4 x 1.5 GHz	ARM Mali-T628 MP6	ADC, GPIO, I2C, SPI, UART, USB, HDMI, SDIO/SD/MMC, PWM, LCD, MIPI CSI2, eDP, PCM, I2S, S/PDIF, PMIC, DMA Controller, MCT

reason is to remove the requirement for low-level programming languages and encourage portability as new hardware becomes available. Popular operating systems usually have good documentation, code examples and support forums to assist with device development. By definition an IoT device will need Internet connectivity; using a full OS ensures access to industry standard secure protocols and vulnerability updates. Other platforms such as MBed [8] support SSL and are excellent for IoT devices but are considered out of scope for this work due requirement for a low level programming language.

The SBC has to support electronic interfaces to attach hardware, for example sensors and actuators. For flexibility we consider the following industry standard interfaces as a requirement, but this is application specific. Devices that support these interfaces and features will easily integrate with the majority of off-the-shelf sensors.

Inter-Integrated Circuit (I^2C) or Two Wire Interface (TWI) is a serial bus capable of hosting multiple master and multiple slave devices using just two connections.

Serial Peripheral Interface (SPI) is a serial bus capable of hosting a single master with multiple slave devices per bus and has a lower power consumption than I^2C.

Pulse-Width Modulation (PWM) is not an interface, but is rather a technique to encode a message by varying the power supplied by a digital pin and is often implemented in hardware.

Universal Asynchronous Receiver/Transmitter (UART) refers to the hardware that converts parallel to serial communications and is often simply called *serial*, it is actually the communication standards over UART to which people are generally referring; the most common are RS-232 and RS-485.

TABLE 2. SUMMARY OF WEATHER STATION SENSORS

Sensor	Interface	Hardware
Wind vane	Analog	Switched resistors
Anemometer	GPIO Interrupt	Reed switch
Rain gauge	GPIO Interrupt	Reed switch
UV intensity	Analog	ML8511
Humidity & temperature	2-wire serial	SHT15
Barometric pressure	I^2C	BMP180
Luminosity	I^2C	TSL2561

Controller Area Network (CAN) is a message-based protocol defined by ISO 11898-1 that allows multiple master device communication and was designed over the last 30 years for in-vehicle electronic networking.

An *Analog to Digital-converter* (ADC) is a device that converts a continuous analog voltage to a digital number. It is important to ensure that the analog signal is sampled at a high enough resolution to ensure that analog variations are not lost.

General Purpose Input Output (GPIO) is a general-purpose pin whose behaviour can be controlled by the user at the run time. Some of these GPIO pins can be configured to function as Interrupt Request (IRQ) inputs –rising, falling or both edges of the incoming signal.

3. Weather station

An exemplar IoT weather station must be capable of monitoring weather conditions outdoors. It will be roof mounted to achieve a clear airflow and solar powered for ease of installation.

A range of common sensors with a variety of capabilities and electronic interfaces are included on the weather station. Table 2 provides a summary of the sensors, their electronic interface and hardware specifics.

The wind vane requires an ADC and has eight magnetic reed switches arranged in a dial, each reed switch is connected to a different sized resistor. As the vane changes direction a magnet moves, making and breaking the switches, therefore changing the resistance of the circuit. The magnet is sized such that when halfway between two reed switches both will be connected. This will give a total of 16 distinct circuit resistances This is a crude sensor and with a resolution of 22.5 degrees, but is an excellent example of a simple analog circuit. A 5 V supply is connected to the wind vane and the enabled resistors act as a voltage divider. Reading the voltage using an analog pin indicates which resistors are enabled in the circuit and hence establish the wind direction.

The three hemispherical cup anemometer is used to calculate the windspeed and it requires GPIO pins with interrupt capability. As the anemometer cups rotate, a single magnet on the device closes a reed switch completing the circuit. This particular device has two reed switches resulting in the circuit closing twice per revolution. Connecting

one end of the circuit to ground and the other to a GPIO pin, pulled high we will get an interrupt every time the magnet passes. For example a wind of 2.4 km/h will result in a single rotation of the cups and two interrupts every second.

The rain gauge is a self-emptying tipping bucket and requires GPIO pins with interrupt capability, each time the bucket empties it moves a magnet past a reed switch which will close the circuit. In the same way as the anemometer, these interrupts are counted over time and used to calculate the rainfall. In this particular gauge the bucket tips every 0.28 mm of rain.

The Ultraviolet (UV) intensity sensor uses an ML8511 detector which is most effective between 280-390nm and will detect both UVA and UVB. The output is a linear voltage relating to the UV intensity (mW/cm2), which can be read using an analog pin and converted to a UV index.

The SHT15 temperature and humidity sensor is a very reliable low cost sensor with an accuracy of $\pm 0.3\,^{\circ}\mathrm{C}$ and $\pm 2.0\%$ relative humidity. The easiest way to communicate with this sensor is to use two GPIO pins one for the clock and one for the data. By manipulating the GPIO pins we can clock commands into the sensor and read the data back.

The BMP180 is a digital pressure sensor which can measure atmospheric pressure between 950 hPa and 1050 hPa with an accuracy of 0.12 hPa. An accurate temperature reading is needed to measure the pressure so the BMP180 can also be used as a temperature sensor. The BMP180 has I^2C serial output and a fixed address and it can be connected to an I^2C bus together with other sensors providing there is not an address clash.

The TSL2561 luminosity sensor detects infrared and full-spectrum light and has an I^2C serial output. The sensor can be used for lux calculations and has a configurable slave address (a total of 3 different address combinations are possible).

4. Single board computer selection

Selecting a SBC is partly application specific as it requires the appropriate industry standard features and interfaces to support the selected sensors shown in Table 2. Table 3 shows a range of single board computers compatible with Linux variants as well as their supported electronic interfaces; note newer boards are a regular occurrence.

The most obvious choice of SCB is the Raspberry Pi 3 Model B+ which is shipped in large numbers and has a huge userbase, offering online support and ensuring bugs are patched. It lacks an Analogue to Digital converter (ADC) and so are unsuitable for this particular example, but worth considering for other IoT applications.

Securing these devices is notoriously difficult [9], [10], [11]. Secure boot is only supported by newer SBC's and it allows a measure of defence against some attacks by only permitting booting from code that has been signed by an authorised key [12], making it more difficult for attackers to introduce malicious code.

The range of features, on-board peripherals, support availability and physical size all need to be considered when selecting a SBC. We selected the Odroid C1+ because of the availability, cost, size, compute power and its supported features. In particular, ADC support, a battery backed up Real Time Clock (RTC) and the same 40 pin header as the Raspberry Pi 2 & 3 Model B+.

The Odroid C1+ has support for Android and Ubuntu as well as a range of unofficially supported operating systems such as ArchLinux Ubuntu Snappy and Debian. This work uses the official recommended OS; currently Ubuntu 14.04, Odroid release v1.6. Ubuntu has a package manager so updates to OS software as well as Odroid specific software will both come through as distribution updates.

5. Implementation

The sensor hardware in Table 2 was connected to an Odroid C1+ running Ubuntu 14.04, Odroid release v1.6 with all updates installed. The sensors are physically mounted in a metal enclosure, protected from water ingress and separated from each other to protect against interference; for example thermal pollution. Internet connectivity is provided via a RT3070 USB module which has an external 5 dB omni-directional detachable antenna; this was replaced with a high gain 24 dB directional antenna. The implementation has been running for over 12 months and has been updated with the latest OS updates as they have become available.

The power is supplied by a single south facing Photo Voltaic (PV) panel, tilted for optimal winter sun. The power is stored in a deep cycle 20 A h lead acid battery and managed by an EPsolar Tracer1215BN v2.05 power regulator [13]. The Tracer has a Modbus [14] connection which provides information about the equipment state, power generation, power consumption and has on/off timers. Being able to query the power circuitry means that it is possible to predict power outages/shortages and either reduce computational load or power off the Odroid. The PV panel can also be used as a sensor, the daytime instantaneous power generation indicates sun intensity and cloud cover.

All the sensor data is gathered using Python [15] scripts. Python is a very popular high-level programming language, is available for most operating systems and has an extensive number of libraries. The ML8511, SHT15, BMP180, TSL2561 all have existing Python libraries making them very simple to integrate. The analog wind vane, GPIO interrupt anemometer and rain guage are supported by a modified version of the Raspberry Pi GPIO library. The selected hardware is well supported and does not need an in-depth knowledge of electronics or low level programming, making prototyping fast.

The device implementation takes advantage of the Odroid C1+ hardware watchdog timers to monitor free memory and system load (indication of a crashed OS). The watchdog timer will reboot the system if the timer expires, the watchdog daemon constantly resets the timer on a healthy system, keeping it alive. The watchdog daemon is very customisable and can look for network outages, hardware temperature and even execute repair scripts.

TABLE 3. COMPARISON OF LOW COST, SINGLE BOARD COMPUTER DEVELOPMENT BOARDS

Mainboard (SoC)	RAM	Storage	USB	Interfaces	Size (mm)	Cost (≈ USD2016)
ODROID-C1+ (S805)	1GB DDR3 SDRAM	MicroSD Card Slot eMMC module socket	4 x USB 2.0 Host, 1 x USB 2.0 OTG	ADC, 40 GPIO, SPI, I2C, UART, HDMI, RTC, IR Receiver, DMC, PLL/OSC.	85 x 56	35
Raspberry Pi 3 Model B+ (BCM2837)	1 GB	MicroSD Card Slot	4 x USB	ADC, 17 GPIO pins, UART, SPI, I2C.	85.6 x 56.5	35
Beaglebone Black (AM3358)	512 MB DDR3	MicroSD Card Slot 4GB 8-bit eMMC on-board flash storage	1 x USB host, 1 x USB miniB	ADC, 66 GPIO, SPI, I2C, UART, CAN, PWM, LCD, GPMC, MMC1, 4 Timers.	86.4 x 53.3	50
Udoo Neo (i.MX 6SoloX)	512 MB or 1 GB DDR3	SPI Flash onboard, MicroSD, 8-bit SDIO interface	1x USB 2.0 A Host 1x USB OTG	ADC, 36 x GPIO pins, UART, 2X CAN Bus, PWM, I2C, SPI	85 x 59.3	50 - 65
ODROID-XU4 (Exynos 5422)	2 Gbyte 933MHz	MicroSD, eMMC	1 x USB 2.0, 2 x USB 3.0	ADC, 42 GPIO, SPI, I2C, UART, PWM, RTC I2S, HDMI, PMIC.	82 x 58 x 22	74

The sensor data is collected, timestamped and written to files on the SD card ready for transmission to backend services. This means that sensor polling can be performed at a different rate to the data upload and any network outages do not result in data loss. Initially the sensor data collection script was executed at boot and operated in an infinite loop. The watchdog timer checked to ensure that the script was working by checking the process ID; this was later changed to use cron [16] to allow for more scripts with different execution frequencies.

The sensor data is uploaded to two different online backend systems. First, the Met Office WOW citizen science project which receives sensor data as HTTP posts with parameters. The WOW project is under development and currently accepts a limited selection of sensor data, it is a good example of how IoT device data can be used when deployed at scale.

The second destination is a cloud hosted Apache Hadoop [17] cluster, which is a software framework capable of using a compute cluster as distributed storage and compute. There is a strong ecosystem of supporting modules for data analysis but more importantly it is supported by multiple cloud providers (Amazon Elastic MapReduce and Microsoft HDInsight) as well as an on-site solution, for example HortonWorks. All the sensor data is transmitted to the Hadoop cluster via a Windows Server Service bus [18] which is identical to the cloud based Windows Azure Service Bus [19]; a hosted commercial service. The data is transmitted using the Python Apache Qpid Proton library [20]. This reduces transmission bandwidth requirements and demonstrates scalability to support large numbers of IoT devices.

6. Discussion

The boards shown in Table 3 offer a wide range of interfaces, peripherals and plenty of compute power. Not all boards are created the same and it is important to select carefully, for example if an ADC is required. All the peripherals included on a board have an impact on the power requirements, for example the Raspberry Pi 3 Model B+ recommends 2.5 A @ 5 V and the ODROID-C1+ requires a minimum 0.7 A @ 5 V. This does not mean that the Raspberry Pi 3 Model B+ continuously draws 12.5 W, but rather is an indication of the expected peak current draw; these recommendations also assume that peripherals such as WiFi dongles and a keyboard will be connected – all requiring power. A device that has a heavy power requirement will have limited application as an IoT device. The Odroid C1+ without any peripherals connected, consumes ≈ 1.6 W once booted with the stock Ubuntu 14.04 (v1.6). The first two minutes of boot consume ≈ 80 mA h. By comparison the Raspberry Pi 2 Model B+ consumes ≈ 1.8 W once booted with the stock Raspbian Jessie. The first two minutes of boot consume ≈ 60 mA h. Connecting a RT3070 chipset USB WiFi dongle consumes a further 750 mW.

The Odroid supports CPU frequency scaling, changing the CPU frequency it is possible to reduce the power requirements when the CPU load is not high. The Linux Kernel has implemented cpufreq [21] since Kernel v3.4 and it is enabled by default. The default OS install configuration keeps the CPU frequency at its maximum. Changing the governor to `ondemand` keeps the device responsive but keeps the frequency low when not required. For example the idle CPU frequency can be reduced to 312 MHz in our weather station, without an impact on performance. Profiling the CPU scaling shows that even with sensor polling scripts and data uploads via WiFi the CPU remains at 312 MHz for over 80% of the device uptime. This could be an indication that the SoC is overly specified for the IoT task. Applying CPU frequency scaling to the Odroid C1+ reduced the power consumption by ≈ 10%.

It is possible to disable unwanted peripherals to reduce the power consumption. The Odroid's four USB ports run off a USB hub which is potentially inefficient, particularly when running only one USB peripheral. When debugging

the device in situ it is convenient to connect a USB keyboard and mouse but disabling the ports when not using them can reduce the power requirements. Disabling on-board peripherals can have some unexpected side effects, for example on the Raspberry Pi 2 & 3 Model B+, the on-board Ethernet connector is also connected as a USB device to the USB hub; disabling the USB Hub will disable Ethernet capability.

The weather station is powered by a single Photo Voltaic panel but this requires lots of physical space and a favourable exposure to the sky. For fixed installations, another option is to consider Power over Ethernet (PoE); 802.3af supports up to 15.4 W and 802.3at supports up to 30 W . This provides reliable power and networking over the same cable and is well suited to external installations. The Intel Galileo Gen 2 board supports PoE but the other boards require a PoE splitter which adds further costs. Many network switches provide PoE (injectors) and have the ability to power cycle individual ports, making a remote reboot of failed devices possible.

The most common way to add storage to this class of SBC is via SD card; the Odroid C1+ supports both SD and removable eMMC storage. Inherently using an SD card for IoT devices is a risk as it is not tolerant to power outages; some efforts have been made to improve reliability [22]. Filesystem corruptions will eventually result in the need for physical intervention which can be difficult for remote IoT devices. We force a file system check at every boot on the weather station but using the supported eMMC would have been a better solution; the eMMC uses a non-standard connector, whereas SD cards are compatible with all the tested boards.

The Odroid C1+ has a battery backed up Real Time Clock (RTC) which means that the device always has the correct date and time. Other SBC devices require Network Time Protocol (NTP) servers to synchronise the time; this requires an Internet connection. This is problematic since some WiFi configurations require client devices to have the correct time in order to connect; WiFi WPA-Enterprise security needs to check the certificate validity period. The correct date and time is important on IoT devices, e.g. to synchronise logs and for timestamped data. This makes a battery backed up RTC important; the Global Positioning System (GPS) is another source of date and time.

The Raspberry Pi 3 Model B+ has a WiFi module on-board, but it does not have an external antenna, the other boards require a WiFi module. Our implementation is stored in a metal weather proof box, forcing the need for an external high-gain directional antenna. Not all WiFi modules are supported by Linux and some have poor performance. All our work is performed using the RT3070 USB module using the `rt2800usb` Linux kernel module.

Having a collection of peripherals built into an SBC has its advantages, but becomes an burden if they are not required for the IoT device. For example; powering a USB hub for one peripheral, powering WiFi and Bluetooth when not used, running a Graphics Processing Unit (GPU) when a display is not connected. Depending on the SBC implementation, it can be possible to disable peripherals in software to save power.

The level of maturity of an SBC is important – on-board peripherals and features are not always accessible from the OS and bugs need to be addressed. For example, the Odroid C1+ had issues when first released, the GPIO pin interrupts were not triggered so the anemometer and raingauge failed to work. Issues are common across new hardware and having a huge community such as the Raspberry Pi is an advantage; there is a higher probability that bugs will be addressed.

The emergence of newer hardware raises the issue of portability. Running Linux with standard libraries and programming in a high level language will make upgrading the hardware easier. One way to simplify hardware upgrades is to pick a Linux distribution which can support a wide range of hardware, for example OpenWRT [23] The stock OS installs often include software that may not be desirable for IoT devices such as windowing environments and are not configured with IoT devices in mind. Overlay FS [24] has been built into the Linux kernel since v3.18 and provides a mechanism to create a read-only root file system and then overlay a read write file system on top. This technique can be used to make SD devices more tolerant to unexpected power outages.

Customising the SBC's default OS can take some work so another approach is to build your own *distribution*. The Yocto [25] project is a build environment which can target many types of hardware. We built an image for both the Raspberry Pi 2 Model B+ and the Odroid C1+ using the Yocto project build system. There are two key advantage of using such a system: i) the base images are very similar across hardware – we experienced a kernel version difference, and ii) it is very easy to remove any unwanted software, drastically reducing the image size and improving boot time. By adding `read-only-rootfs` to the Yocto configuration, both OS image root file systems are configured as read only, without any additional effort. The OS images can also easily be configured to get updates from a custom server, this provides greater control over which packages get updated and it best suited to large numbers of IoT devices.

Operating systems, other than Linux variants are worth consideration, for example Microsoft Windows IoT Core supports multiple different SBC platforms and is easily programmed in languages such as C#, but it is unclear how extensive the hardware libraries will become.

Many IoT situations will have limited or costly Internet connectivity; for example satellite connections. This implementation sends data to the WOW service via HTTP which requires a large data bandwidth, but other data transmission protocols need to be considered such as REpresentational State Transfer (REST) and frameworks such as Apache Thrift [26]

Both SOAP and REST were not created with IoT devices in mind. By looking at these and other technologies we can list the top IoT data transmission requirements: *Efficient data transmission packet size* - Remote IoT devices using mobile phone or satellite internet connections need to pre-

serve bandwidth and data transmission costs. *Secure reliable data transmission* - Messages need to be resent or batched for example, where internet connectivity is intermittent. *IoT message persistence* - Unsent messages need to be stored on the IoT device to survive crashes and power outages. *Supported by a wide range of programming languages* - The message needs to be easy to build using a wide range of languages and hardware used for IoT devices.

Based on these requirements it is clear that we need more than just a messaging protocol, we need a message framework, for example, Message Queue Telemetry Transport (MQTT), Messaging and Presence Protocol (XMPP), Data Distribution Service (DDS), Constrained Application Protocol (CoAP) and Advanced Message Queuing Protocol (AMQP) [27]. Our implementation uses AMQP because it is supported by the Window Service Bus, however further work is required to select a framework that is both lightweight, well supported and offers an adequate level of security.

7. Conclusion

In this paper we investigated the feasibility of building an IoT weather station device using commodity off the shelf hardware and a low-cost single board computer. The work was undertaken over a period of 12 months and the final implementation operated reliably, supporting a range of sensors.

We conclude that many single board computers are suitable for IoT devices, particularly for IoT prototyping and development. Throughout this work it is evident that some effort is required to harden OS distributions specifically for this class of device and that not all SBCs are created the same; more hardware is not always better.

The low cost, extensive hardware libraries, community support and hardware availability make this class of devices applicable for designing, building and testing the next generation of IoT device. They will have higher power requirements, suffer from hardware vendor issues, SD card corruptions and include un-utilised hardware, amongst other issues which must be taken into consideration.

With hardware prices starting at $15 USD for a powerful, fully featured computer, it is clear to see that development in this area will be big. We predict a growth in using this class of computer particularly for IoT prototypes; disposable compute or low hardware volume scenarios. This class of device brings the capabilities of embedded systems to non-technical users and we can expect much development in this area.

References

[1] Peter Middleton, Peter Kjeldsen, and Jim Tully. Forecast: The internet of things, worldwide. 2013.

[2] Hardkernel: Odroid. http://www.hardkernel.com/, 2013. Accessed: 2016-03-11.

[3] Udoo Neo. http://www.udoo.org/, 2015. Accessed: 2016-03-11.

[4] Pine 64. https://www.pine64.com/, 2015. Accessed: 2016-03-11.

[5] Met Office WOW. http://wow.metoffice.gov.uk/, 2015. Accessed: 2016-03-11.

[6] Winn L Rosch. *Hardware Bible*. Que Publishing, 5th edition, 1999. ISBN 0-7897-1743-3.

[7] Alessandro DAusilio. Arduino: A low-cost multipurpose lab equipment. *Behavior research methods*, 44(2):305–313, 2012.

[8] ARM MBed. https://www.mbed.com/en/, 2016. Accessed: 03/2016.

[9] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013. Towards a Science of Cyber Security: Security and Identity Architecture for the Future Internet.

[10] H. Suo, J. Wan, C. Zou, and J. Liu. Security in the internet of things: A review. In *2012 International Conference on Computer Science and Electronics Engineering (ICCSEE)*, volume 3, pages 648–651, Mar 2012.

[11] R. Khan, S. U. Khan, R. Zaheer, and S. Khan. Future internet: The internet of things architecture, possible applications and key challenges. In *2012 10th International Conference on Frontiers of Information Technology (FIT)*, pages 257–260, Dec 2012.

[12] J. E. Ekberg, K. Kostiainen, and N. Asokan. The untapped potential of trusted execution environments on mobile devices. *IEEE Security & Privacy*, 12(4):29–37, Jul 2014.

[13] EPsolar. *Tracer-1206RN / 1210RN / 1215RN Instruction Manual*. Beijing Epsolar Technology co., LTD. Utility model patent no. 201120064092.1.

[14] Modicon. *Modbus Protocol*. Trexon Inc, 3-1750 The Queensway Suite 1298 Toronto ON Canada M9C 4H5, 2000.

[15] Michel F Sanner et al. Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1):57–61, 1999.

[16] Daniel P Bovet and Marco Cesati. *Understanding the Linux kernel*. O'Reilly Media, Inc., 2005.

[17] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[18] Microsoft. Service Bus for Windows Server (Service Bus 1.1). https://msdn.microsoft.com/en-us/library/dn282144.aspx, 2013. Accessed: 2016-03-11.

[19] Tejaswi Redkar, Tony Guidici, and Todd Meister. *Windows Azure Platform*, volume 1. Springer, 2011.

[20] Service Bus for Windows Server (Service Bus 1.1). https://qpid.apache.org/proton/, 2015. Accessed: 2016-03-11.

[21] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230. sn, 2006.

[22] Kim Jaegeuk. The F2FS system. http://git.kernel.org/cgit/linux/kernel/git/jaegeuk/f2fs.git/. Accessed: 2016-03-11.

[23] Florian Fainelli. The openwrt embedded development framework. In *Proceedings of the Free and Open Source Software Developers European Meeting*, 2008.

[24] Andy Whitcroft. OverlayFS documentation. https://git.kernel.org/cgit/linux/kernel/git/apw/overlayfs.git/. Accessed: 2016-03-11.

[25] Elizabeth Flanagan. The yocto project. *The Architecture of Open Source Applications*, 2:347–358, 2012.

[26] Apache Thrift. https://thrift.apache.org/, 2016. Accessed: 2016-03-11.

[27] International Organization for Standardization. ISO/IEC 19464:2014 information technology Advanced Message Queuing Protocol (AMQP) v1.0 specification. ISO/IEC, 2014.