Impact of Non-Deterministic Software Execution Times in SmartGrid Applications

David Smith, Cristian Olariu, Philip Perry, John Murphy
University College Dublin, Ireland

***Abstract*—Electricity companies will only allow customers to inject power into the grid network if the customer's system is in synch with both frequency and phase as the grid itself. This is normally achieved using a specialised machine sold by the energy provider. It may be possible to achieve this via low cost, low power nodes such as a Raspberry Pi, to synchronize frequency and phase as well as voltage and current. In order to synchronize to microsecond or nanosecond precision, the hardware being deployed must in itself be able to achieve said precision when coupled with software. In this paper we evaluate the level of non-deterministic execution times in two versions of a widespread embedded compute platform, namely Raspberry Pi.**

Keywords—SmartGrid, IoT, Timing, Software

I. INTRODUCTION

The deployment of renewable power sources using the concept of microgeneration systems connected to microgrids is becoming an important tool in the diversification of energy sources, but requires considerable control [1]. The use of such systems can be somewhat enhanced through the use of low cost hardware with embedded software at the distributed nodes both for monitoring the microgeneration system and implementing adaptations requested by the central controller. In AC systems, the nodes must measure voltage, current, frequency and phase, while DC systems need to measure voltage and current.

Although measured data and control signals can be communicated between the nodes using a number of communications protocols [2], it seems likely that the increasing emphasis on the Internet of Things motivates the use of a TCP/IP approach to communications which, in turn, motivates the use of readily available platforms such as Raspberry Pi (Pi), Arduino, Beaglebone or ODROID. Until recently such systems used single core CPUs, but the use of quad-core CPUs is now a possibility and this may make such platforms more reliable for SmartGrid and microgrid control. In particular, variation in the execution time of software embedded in the node can have a significant impact on the monitoring accuracy.

The non-deterministic execution time for any given computer programme has historically been of little concern in the majority of applications. However, as software moves towards physical systems through the concept of Cyber Physical Systems (CPS), the question of how to make the software be synchronised to some real world events or clocks becomes increasingly important [3].

In this paper, the level of variability of the software execution is explored for two comparable compute platforms, one with a single core and one with a quad core, viz. the RPi B+ and the RPi 2. This will form a basis for deciding which types of compute platform are likely to be able to satisfy the requirements of a microgrid controller implementation.

II. SOURCES OF VARIABILITY

The most obvious source of variability in software execution time is the inherent drift of the CPU's clock frequency with temperature. This is caused by the small frequency drift of the crystal oscillator that provides the CPU with a reference frequency for its Phase Locked loop (PLL). This can be corrected by using a synchronisation protocol such as the Network Time Protocol (NTP) [4]. The RPi platforms investigated here typically exhibit frequency errors of the order of 100 parts per million (ppm) which translates to an error of less than 12.5ns in the period of the 8kHz signals investigated here.

Another source of variation arises mainly in single core CPUs which must do a number of housekeeping activities as well as perform the function specified in the software programme. These activities include running the Operating System (which can be either command line Unix or a Graphical User Interface (GUI)) and managing the interfaces to the network and peripheral devices. With a multicore system, these overhead activities can be managed by one core, while the software programme is executed by a second core and further programmes can be executed by other cores if they are available. Since the type of control function envisaged here is a periodic polling of a sensor or Analogue to Digital Converter (ADC), each iteration of the code will have an approximately constant execution time required for the loop that can be calibrated by adjusting the time delays in the "wait" functions of the code. However, the variation caused by slightly different timing of events at the CPU are unavoidable, so there will be some irreducible jitter associated with the execution time.

This paper presents preliminary measurements of the variability of the execution of a simple measurement trigger process to evaluate the impact of the variability in software execution time on the ability to perform such triggering reliably.

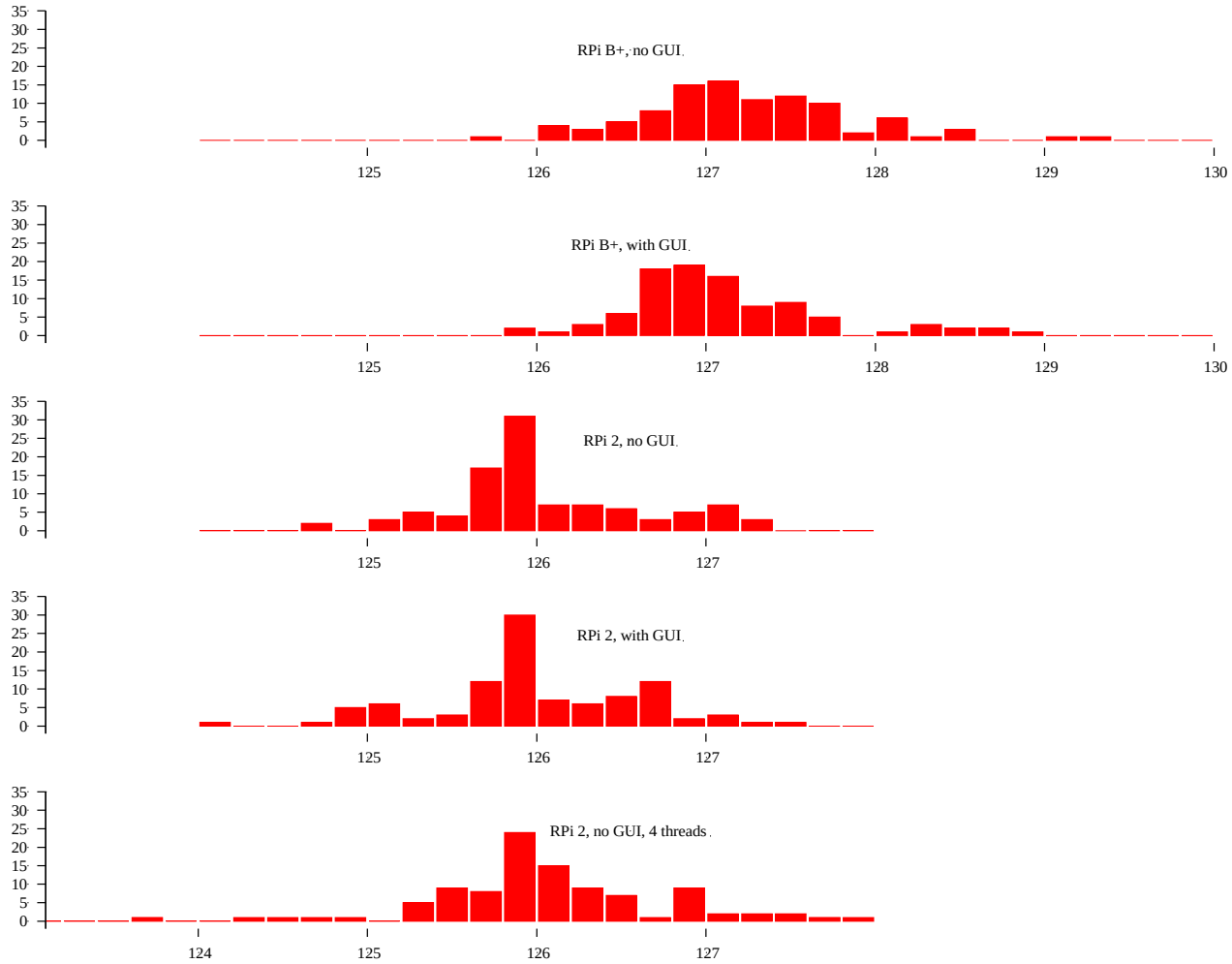

Fig. 1 Histograms from set of tests carried out on both Pi's showing frequency each period. (X-Axis scale in us)

III. EXPERIMENT

Testbed:

The testbed consisted of two Raspberry Pi's (PI's), the older B+ model, and the newer Pi 2 model. The B+ has a single 700MHz broadcom processor and 512MB of RAM. The Pi 2 has a quad core 900MHz broadcom processor and 1GB of RAM. These were connected to a 100MHz Oscilloscope via their General Purpose Input Output (GPIO) pins and they were connected via ssh on another machine.

Setup:

To assess the timing differences associated with a single core CPU vs. a quad core we utilized the GPIO pins on both Pi's. These pins allow the user to set them to High (3.3V) or Low (0V). The 8kHz frequency was chosen because it is often used in voice sampling applications so there are many low cost chips available to capture and process this sampling rate. Since the compute platforms selected are similar to the CPUs used in smartphones, the test results could also be relevant to the performance of a low cost, low power Voice over IP node. In the context of smartgrid control, this sampling rate equates to a sample every 2.25 degrees of phase at 50Hz which seems adequate for the control system envisaged.

Software:

The trigger signal uses the rising edge of an 8kHz square wave (period=125usec) which was generated on the GPIO pin by running this simple loop:

```
While()
  Assert Pin.0 "High"
  Wait 62us
  Assert Pin.0 "Low"
  Wait 63us
```

Endwhile

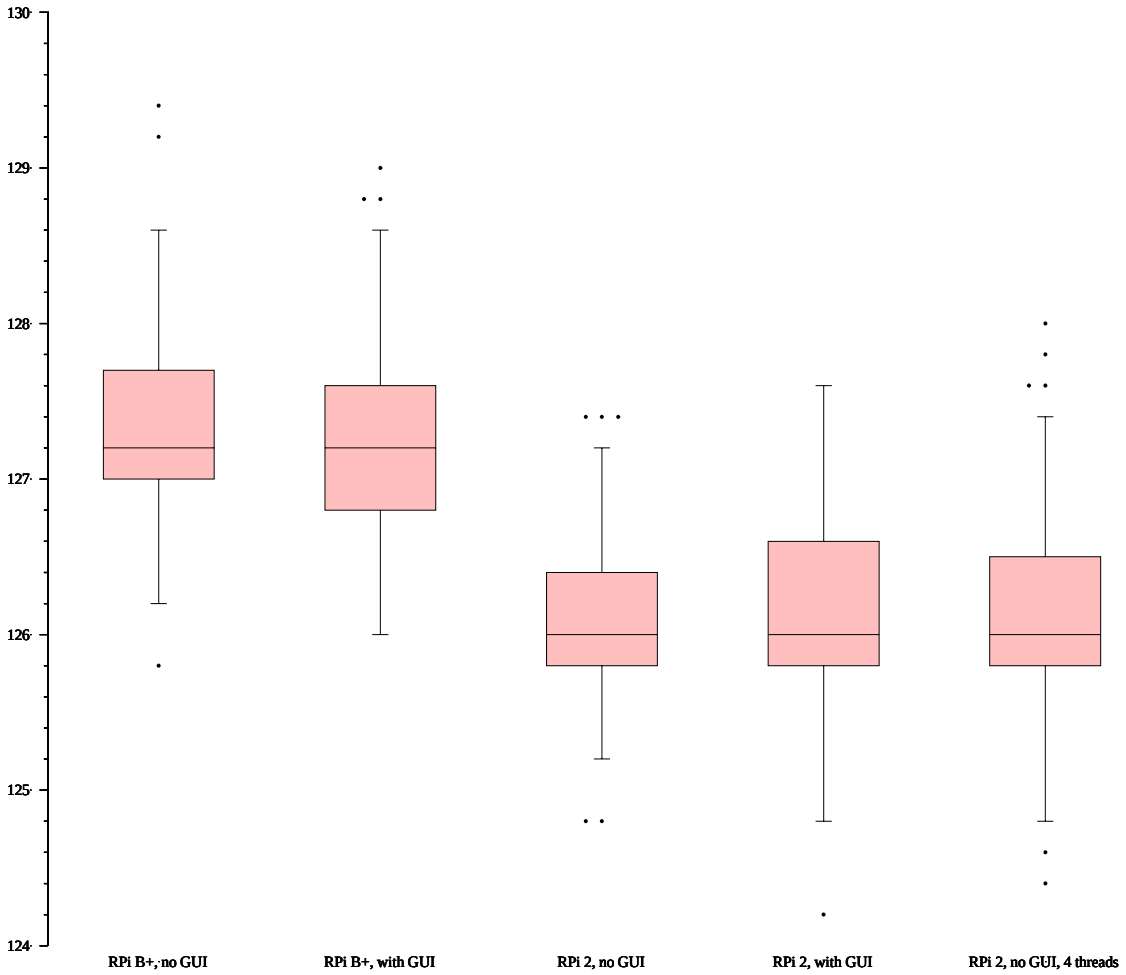

Fig. 2 Boxplot from set of tests carried out on both Pi's (Scale in us).

Although many programmers use Python to programme their RPi's, most (if not all) libraries used in Python to assert GPIO pins are not suitable for time critical applications. Therefore the tests here used the C programming language by leveraging the WiringPi Library. The period of the square wave was set to 125usec and made no attempt to correct for any inherent latency in the system execution. This was connected to a 100MHz Oscilloscope to measure the period of the switching of states.

The tests consisted of two configurations for each Pi, one with the GUI turned off and the other with the GUI left on. This allows for both cases where a Pi might not just be used as a sensor to some central system but also as a monitoring hub in itself. For each configuration the system was allowed to warm up to stabilise any thermal influence and then ran the loop constantly and took 100 measurements of the period of the square wave.

IV. RESULTS

After collecting and analysing the results it was clear that the quad core CPU was significantly more stable than the single core case. This could be clearly seen by the outliers and higher jitter which appeared in the Raspberry Pi B+ data. The outliers were so far outside of the data range that it completely skewed the rest of the plots, so they have been omitted. In the Pi B+ with no GUI running it had just one major outlier, which was 12us from the median. When the GUI was turned on with the Pi B+ it produced 6 outliers, ranging from 15us to 30us from the median. Despite only measuring 6 outliers, significant jitter was apparent on the

oscilloscope when the Pi B+ was running the GUI. These outliers occur every few seconds and lasted for milliseconds which would have severe impact on the microgrid monitoring system.

Despite the Pi B+ having a slower CPU clock speed than the Pi 2, it still managed to output the 8KHz signal with relatively high precision. The C program that generates the 8KHz signal loop runs on a single core, since the Pi B+ and Pi 2 have a similar clock speed, 700MHz vs 900MHz respectively and similar GPIO circuitry, it is unlikely that the timing variations are due to the hardware performance. The difference lies with the Raspbian OS (R Pi version of Debian) which is running in the background. In the case of the Pi B+ the same core that is running the loop is also running the OS, whereas with the Pi 2 another separate core looks after the OS and any housekeeping that needs to be done or running the ssh client. Both the Pi's were connected to another machine via ethernet and an ssh connection was established to control and monitor them. This is representative of how a Pi would be used in a smartgrid application as the monitor data will be relayed to a central hub via the Internet.

The results are presented in Figure 1 and 2. Figure 1 is the set of tests across the two Pi's plotted on the same axes. Fig. 1 shows the frequency of the specific time interval for each test. Where the y-axis shows the number of instances and the x-axis is the microsecond periods. Fig. 2 is the same results but it boxplot format. Y-axis in microseconds.

The first two plots are the Pi B+ without a GUI and with, the 3rd and 4th are the Pi 2 without a GUI and with, and the last result is the Pi 2 with 4 threads running concurrently. From the histogram it can be seen that a significant right shift has occurred with the Pi B+ both with and without GUI with respect to the Pi 2. This shows that running the OS and running the ssh client causes increased jitter and further delay in software execution time. The jitter can be seen by the spread of the histogram, with much more outliers than the Pi 2 also.

The Pi 2's results are less spread, this illustrates that the other cores were able to manage the OS and ssh client without impacting on the timing as much. There were no major outliers with the Pi 2 and all of the periods were within 2.5us of the specified period.

Despite the program setting the transition to occur at 125us intervals, it is noted that a few of the results returned less that 125us periods. This is because of the delayMicroseconds() function which is used to force the program to wait a fixed period of time. This function has a rounding function built into it, whereby the delay will be rounded either up or down to the nearest clock cycle, thus causing about 6 results with less than 125us period.

When the Pi B+ was given any other task to do, its performance deteriorated to an unacceptable level. In this test, a second thread was created which generated another clock signal on a second GPIO pin. It was clear that the Pi B+ swapped between threads at 10ms intervals which resulted in 10ms bursts of clock cycles on the pins punctuated by 10ms of inactivity while the CPU processed the other thread. When similar tests were performed on the Pi 2, up to four simultaneous threads were mapped to the four cores with no thread swapping required. The core that was running the C programme and also managing the housekeeping showed a small degradation of performance relative to the other cores. This is illustrated in the last plot of Figure 1 where the jitter is more similar the Pi B+ but that is to be expected since all four cores are fully occupied. Despite this the median remained constant with respect to the previous two tests performed on the Pi 2.

In Figure 2 the outliers are more clearly distinguished and the differences between the two Pi's is much easier to make. The boxplots show the median, the upper and lower quartiles and the min and max excluding outliers. It is to be noted that once again not all of the outliers for the Pi B+ are shown on the plots, this is because they would skew the plots to make them unreadable. From the plot it may seem that the Pi B+ with GUI is in fact performing better than the Pi B+ without GUI, but the reality is that the GUI made the number of outliers increase significantly, and they are not shown.

It is easy to see the median across the tests remained the same, the difference between the two Pi's is 1us, this 1us of extra delay is a mixture of the slightly slower clock speed, coupled with the OS and ssh client loading the single core. When the quad core test with four concurrent threads was run, it produced many outliers, but this was expected, all these outliers are in the graph below and none are omitted. However when as few as two threads were ran on the Pi B+, the results had such bad jitter that the tests couldn't be completed conclusively and to the same degree.

Further tests were carried out to validate the results produced above. Since the Pi B+ and Pi 2 run at different base clock speeds 700MHz and 900MHz respectively, the Pi 2 was locked to 700MHz. The result from this test showed a ~22% increase in the median overhead. That was an extra delay of 250ns. This result was expected as it further illustrated that the Pi B+ was slower due to its single core having to run the OS and also perform any housekeeping, and that the different clock speed only marginally affected the results because during this test, no major outliers were found.

In order to ensure the clocks of the Pi's were not skewing apart during the previous tests, they were both synchronized using NTP for at least 24 hours, and the tests were redone. These now synchronized systems produced the same results that we have shown above, so the clock skew was not impacting on the results in any significant way.

The ODROID-C1 is another embedded system which is very similar to the Raspberry Pi, however, it runs a 1.5GHz quad core. The increased clock speed offered a chance to evaluate the limitations of the lesser powered cpu's of the Pi B+ and Pi 2. Using the Odroids, the same test was performed and the median cycle time was 125.4us, versus 126.15 on the Pi 2. This result, showed clock speed was inversely proportional to overhead within these two very similar systems. The Pi 2 having a 900MHz clock and Odroid with 1.5GHz, a 66% increase in clock speed produced a 65% decrease in overhead. The Odroid C1 did not produce any significant outliers.

V. Conclusion

In this work we have shown that the newer quad core processors which have become ubiquitous among low cost, low power nodes such as the Raspberry Pi, Arduino, Beaglebone and ODROID are far superior in terms of reducing the non-deterministic software execution times. We mentioned that jitter in execution time can be caused by a number of factors, namely the temperature variations of the crystal oscillator used in the system. As well the OS and other housekeeping tasks or networking such as an ssh connection or WiFi can affect the time taken for a simple loop to execute. By paying particular attention to the outliers that appeared in the Pi B+ tests we have shown that handling the OS as well as any other mildly demanding task is just too much for the Pi B+, also when a GUI is introduced it further increases the variability. In general, any low power single core processor which has to run even a lightweight OS, such as a SmartGrid controller, would be unreliable to maintain and monitor phase and frequency when its own software has variations of microseconds. Even keeping their own time correctly would be a problem, when the variation in such a simple loop is in the tens of microseconds with a GUI running, it is to be expected the microsecond precision offered by NTP for example would not be achievable at all.

In the future we plan to adopt quad core nodes as part of a SmartGrid controller which will be used to monitor and synchronize the phase, frequency, voltage and current of itself with respect to the grid, and to possibly synchronize neighbouring nodes also. This research has been useful to establish the impact of running low cost, low power, single CPU nodes on a timing sensitive software deployment.

VI. Acknowledgements

Supported, in part, by Science Foundation Ireland grant 10/CE/I1855 and by Science Foundation Ireland grant 13/RC/2094.

References

[1] [1] Dragicevic, T.; Guerrero, J.M.; Vasquez, J.C.; Skrlec, D., "Supervisory Control of an Adaptive-Droop Regulated DC Microgrid With Battery Management Capability," Power Electronics, IEEE Transactions on , vol.29, no.2, pp.695,706, Feb. 2014.

[2] [2] Sedghisigarchi, K.; Eslami, Y.; Davari, A.; Wilkerson, S., "A real-time testbed for coordinated control of inverters in LV microgrids," Energy Conference (ENERGYCON), 2014 IEEE International , vol., no., pp.147,152, 13-16 May 2014.

[3] [3] Lee, E.A., "It's about Time: Leveraging Clock Synchronization for Distributed Real-Time Programming," Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on , vol., no., pp.213,213, 10-12 June 2014.

[4] D. Mills,'Internet Time Synchronization : The Network Time Protocol', IEEE Trans. on Comm. vol.39, no.10, :pp.1482-1493, October 1991.