Finger Finder: A Low-Energy Peak Detection Accelerator for Capacitive Touch Controllers

Kai Kristian Amundsen
Gaute Myklebust
kai.kristian.amundsen@mywo.no
gaute.myklebust@mywo.no
MyWo AS
Trondheim, Norway

Per Gunnar Kjeldsberg
Magnus Jahre
pgk@ntnu.no
magnus.jahre@ntnu.no
Norwegian University of Science and Technology
Trondheim, Norway

ABSTRACT

Capacitive touch screens are ubiquitous, and all such screens need a touch screen controller to detect user input. In this paper, we evaluate a number of software and hardware schemes for identifying the location of user touches in such systems. We conduct our study in the context of MyWo's RISC-V-based touch controller, and focus on identifying user touches — or finding the peaks in an image — with minimal energy consumption. We find that the peak detection algorithm is heavily memory-bound and has very limited computation. Thus, the complexity of even a simple RISC-V processor is almost entirely wasted, and we propose the Finger Finder peak detection accelerator to remove this overhead. Although Finger Finder consumes 7.6 times less energy than the most efficient software-based scheme, it has a limited impact on the energy consumption of the complete touch controller.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

Touch screen controllers, Peak detection, Accelerators

ACM Reference Format:
Kai Kristian Amundsen, Gaute Myklebust, Per Gunnar Kjeldsberg, and Magnus Jahre. 2019. Finger Finder: A Low-Energy Peak Detection Accelerator for Capacitive Touch Controllers. In *CARRV '19: Third Workshop on Computer Architecture Research with RISC-V*.

1 INTRODUCTION

Capacitive touch screens have become ubiquitous, and 1.7 billion screens were produced in 2016 [20]. The touch controller is responsible for applying, sensing, and analyzing the signals required to detect user input. Commonly, touch-enabled systems are put into low-power modes when not in use, and the system resumes normal operation on user input. The trend is towards systems blending more and more into the environment which results in systems where the touching the screen is the only way to wake them up. Thus, the touch controller needs to be enabled when the rest of the system sleeps — making touch controller energy consumption a key contributor to the system's energy consumption while idle. Furthermore, touch controllers need to maintain high sample rates (e.g., 100 Hz) to remain responsive, since responsiveness is critical for the user experience [18]. The key result is that touch controllers need to minimize their energy consumption to ensure that they do not adversely affect battery life.

The basic operation of capacitive touch controllers are shown in Figure 1. At a high level, controller processing consists of two steps. In the *Image Acquisition (IA)* step, the capacitances are first sensed. Then, noise is removed to produce an image with as good signal-to-noise ratio as possible. The image is then passed on to the *Image Post-Processing (IPP)* step where fingers (i.e., peaks) are detected in the image. If no fingers are detected, the system sleeps until the next time the screen is sampled. Conversely, a feature extraction step is executed and a number of touches — and their positions — are communicated to the application processor (see Section 2 for more details).

Prior work has exclusively focused on the IA step (e.g., [4, 7, 10, 17, 19]). We assume a highly efficient IA component and focus our efforts on reducing the energy consumption of the IPP step. The critical part of IPP is peak detection since it must be executed for every sample. Since no prior work has studied peak detection in detail, we systematically study both software and hardware solutions with the objective of minimizing the energy consumption of the peak detection algorithm.

The peak detection algorithm examines all pixels of the image to identify the (potential) location of the user's fingers. For each pixel, it first checks if the value is above a preset threshold (i.e., above the background noise). Then, it checks if the pixel value is larger than all neighbors. If it is, the pixel is classified as a peak and added to a list of peak pixels which is later passed on to the feature extraction step (see Figure 1). In MyWo's touch controller, the peak detection algorithm is entirely implemented in software that runs on a low-power RISC-V core. Although simple, we find that this solution leaves considerable leeway for optimization. Unfortunately, peak detection only accounts for approximately 2.5% of the touch controller's total energy consumption, but we opt to report this intermediate result as it may be relevant in other use cases. In future work, we intend to study the parts of the touch controller that have a more significant impact on the total energy consumption.

Our key observation is that peak detection approaches are fundamentally constrained by the data dependencies of the peak detection

CARRV '19, June 22, 2019, Phoenix, AZ

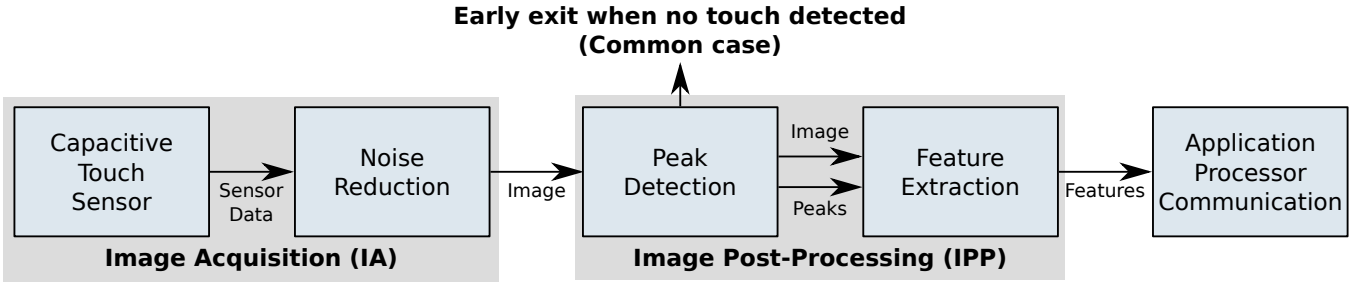


Figure 1: The image processing steps of a typical capacitive touch controller.

algorithm. However, the algorithm lends itself to hardware acceleration, and we propose an accelerator — called *Finger Finder (FF)* — for this purpose. FF simply reads each pixel of the image and compares it to the noise threshold. If the value of a pixel p is above the threshold, FF sequentially compares the value of p to the value of its neighbors until it finds a pixel with a higher value (i.e., p is not a peak) or it has compared p to all its neighbors (i.e., p is a peak). If p is found to be a peak, FF writes its coordinates to a peak list in an on-chip SRAM memory. FF reduces energy consumption by 7.6× compared to the best software algorithm, and uses only 9% more energy than a theoretically derived lower-bound for energy consumption. The root cause of FF's high efficiency is the simplicity of the peak detection algorithm — which means that the complexity of even a simple processor is entirely unnecessary.

In summary, we make the following key contributions:

- We describe the processing pipeline of MyWo's state-of-the-art RISC-V-based capacitive touch controller.
- We thoroughly evaluate a number of software and hardware peak detection approaches — including an oracle scheme that provides a lower-bound for the energy consumption of peak detection.
- We propose Finger Finder — a hardware accelerator that reduces the energy consumption of the peak detection algorithm by 7.6× compared to the best performing software algorithm. Finger Finder uses only 9% more energy than the theoretical oracle scheme.

2 THE MYWO TOUCH CONTROLLER

Figure 1 shows an overview of the processing steps of a typical capacitive touch screen controller. As aforementioned, the processing consists of two main steps: *Image Acquisition (IA)* and *Image Post-Processing (IPP)*. In this section, we discuss the key operations performed in each of these steps. We focus on peak detection since this component has so far not been studied in the literature.

The IA-step uses a system of touch screen drivers, amplifiers and ADCs to acquire screen data in a digital form. The data is then post processed by a series of filters and accumulators to reduce noise. After a complete screen acquisition, the result is a two-dimensional array representing the measured capacitance of the corresponding nodes in the touch screen.

After the image is acquired, we enter the IPP phase and the image is analyzed to find peaks. A peak is found if a node is surrounded by nodes with a lower or equal value. A simple implementation of the peak detection algorithm is shown in Algorithm 1. The feature detection step uses both the peaks found and the image. A list of previously detected touches will be matched with the peaks found to track the touches on the screen. Finger or hand gestures will also be detected and large touches — such as a full palm — will be classified and discarded. After all features have been detected, the number of touches, their positions, and other features are communicated to the application processor. This is typically done using SPI or I^2C.

Algorithm 1 Naive Peak Detection Algorithm

```
for Each line n in image do
  for Each node m in image do
    if image[n][m] >= image[n-1][m] and image[n][m] >= image[n+1][m] and
       image[n][m] >= image[n][m-1] and image[n][m] >= image[n][m+1] and
       image[n][m] >= image[n-1][m+1] and
       image[n][m] >= image[n+1][m+1] and
       image[n][m] >= image[n-1][m-1] and image[n][m] >= image[n+1][m+1]
    then
      Put image[n][m] in list of peaks
    end if
  end for
end for
```

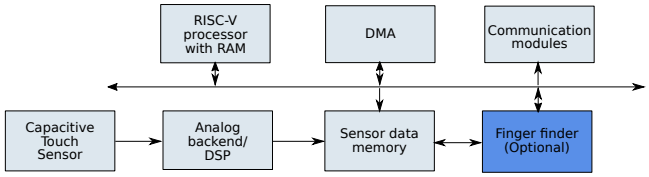


Figure 2: Architecture of the MyWo Touch Controller.

In a typical touch screen scan, the IA is on for 2 ms within the 10 ms period (i.e., 100 Hz sample rate), and IA accounts for three quarters of the total energy consumption. The IPP stage is on for 4 ms and consumes one quarter of the total energy. Within these 4 ms, IPP spends 0.2 ms doing peak detection. Thus, peak detection accounts for approximately 2.5% of the total energy consumption.

The architecture of MyWo's touch controller is outlined in Figure 2. The system consists of a 32-bit RISC-V general purpose CPU with corresponding on-chip SRAM memories and a DMA unit. In addition, the system includes a dedicated subsystem for acquiring sensor data and for initial sensor data processing. After the sensor data has been acquired and processed, it is stored in a sensor data memory block. This is a single-ported on-chip SRAM with several time-multiplexed interfaces. The sensor data is post-processed to

detect peaks and features. This post-processing is typically carried out on the RISC-V core. However, the architecture also allows for dedicated hardware accelerators — such as Finger Finder — to operate on the sensor data.

3 IMPLEMENTING PEAK DETECTION

The peak detection algorithm outlined in Algorithm 1 can be implemented in different ways. In this section, we describe a number of software and hardware implementation options.

3.1 Software-based Peak Detection Schemes

Software Naive (SW-N): This is our baseline implementation, which mostly follows Algorithm 1. However, we added a small optimization to reduce the number of recorded peaks on a plateau and a noise threshold. To reduce the number of peaks, the current node is incremented with one if one of the neighbors have the same value. This ensures that when one of the equal neighbors are processed, they will see the incremented node as a peak and will not be recorded as another peak. This optimization is particularly useful when the user rests a palm on the screen. The noise threshold reduces the number of memory loads and compares by ensuring that comparisons to neighbor node values are only carried out if the current node is over a predefined threshold.

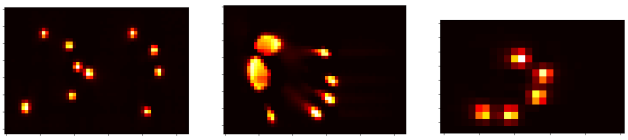
Software Caching (SW-C): In the SW-N scheme, the values of neighboring nodes are discarded when going to the next node. Another option is to keep these values in a cache to reduce the number of memory loads. Therefore, we modified the SW-N scheme to keep a cache of the neighboring nodes. When the algorithm shifts to the next node, the cache is also shifted and three new neighbors are loaded. We refer to this scheme as SW-C.

Software Combined (SW-T): When there are very few nodes over the threshold, the overhead of managing the cache may lead to unnecessary loads — which may increase execution time. We therefore implemented a variant of the SW-C scheme that only updates the cache if the value of the current node is over the threshold. We refer to this scheme as SW-T.

Software Naive 16 bit (SW-N16): In our system, the on-chip scratchpad reads and writes 32-bit values, but the image data is 16-bit. Therefore, it could be beneficial to pack two 16-bit image values in each 32-bit memory word and thereby fetch two image values on a single memory access. We refer to this scheme as SW-N16. SW-N16 unpacks the image data before processing each value as in SW-N.

3.2 Finger Finder Accelerator Variants

Finger Finder (FF-N): To reduce the energy consumption, we implemented the functionality of the SW-N scheme as a hardware accelerator with direct access to the sensor data scratchpad. For each node, a hardware finite state machine reads a data element from memory. In the next cycle, it compares the data element to the threshold to evaluate if the next node's address or one of the neighbor addresses should be loaded. If the node is over the threshold and all neighbors are equal, a table in memory will be updated with the new peak found. As in the SW-N scheme, the current node value will be incremented if it is equal to the value of one of its neighbors to reduce the number of peaks in plateaus. We refer to this scheme as FF-N.



(a) TF-10F-1 (b) TP-PALM-1 (c) PF-5F

Figure 3: Realistic data set examples (see Table 1).

Finger Finder Cached (FF-C): We also investigated an extension to FF-N that caches neighbor values. By using a register bank for the cache, we can update it in a single cycle by executing all shifts in parallel. Also, the comparison between the current node and all the cached neighbors are carried out simultaneously.

Area overhead of FF and FF-C: To establish the area overhead of FF-N and FF-C, we synthesized them together with the rest of MyWo's prototype chip, using a standard cell library and RAMs from a 180 nm commercial process (see Section 4 for details regarding our setup). The area consumed by FF-N and FF-C was 3% and 6% of the RISC-V CPU area, respectively (excluding memories). The RISC-V CPU accounts for less than one tenth of the total area of the touch controller — rendering the area overhead of the accelerators small in comparison. FF-N and FF-C are not on the critical path and therefore do not affect clock frequency.

3.3 A Lower Bound for Energy Consumption

Oracle (O): To identify the potential for energy reduction realized by the peak detection schemes, we designed an oracle scheme. The oracle knows if a node is a peak without examining any neighbors. In other words, the oracle reads every image value exactly once and performs a single comparison. This behaviour matches the behaviour of FF-N when all values in the image are below the noise threshold, and we use this configuration to determine the energy consumption and execution time of the oracle.

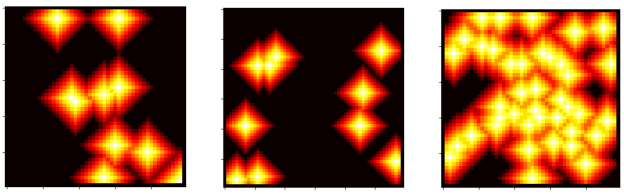
4 METHODOLOGY

To evaluate the different peak detection schemes we collected and generated a number of data sets (see Table 1). Our data sets fall into two categories: realistic and synthetic (see Figure 3 and 4 for examples). The realistic data sets were captured using an FPGA prototype with a touch screen sensor and an image acquisition pipeline. The synthetic data sets were specifically designed to investigate how the different implementations behave in corner cases and were made by a random data set generator written in Python. This enabled us to experiment with different image sizes and different numbers of peaks.

We simulated an RTL-level implementation of the digital parts of the touch screen controller in Cadence's Incisive RTL simulator [1]. The touch screen controller used the data sets as input to a program running on the controller's CPU. The program either performed the peak detection in software (i.e., using SW-N, SW-C, SW-T, or SW-N16) or started an FF accelerator (i.e., FF-N or FF-C). After the chosen peak detection scheme completed, the result was checked

Table 1: Realistic and synthetic data sets.

Abbrv.	Screen	Gesture	#Sets	Description	Figure
TF	Tablet (55x37)	Fingers	15	Three data sets from each of five cases: One finger (TF-1F), two fingers (TF-2F), five fingers (TF-5F), 10 fingers (TF-10F), and two finger pinch (TF-P).	3a
TP	Tablet (55x37)	Palm	1	Hand on the sensor (TP-PALM).	3b
PF	Phone (26x16)	Fingers	15	Five data sets from each of three cases: One finger (PF-1F), two fingers (PF-2F), and five fingers (PF-5F).	3c
S	50^2	Synthetic	12	Clustered peaks with short (S-10C) and long gradient (S-5C) and random peaks with short (S-10R) and long gradient (S-5R).	4a
SZ	$\{10^2, 20^2, \ldots, 60^2\}$	Synthetic	18	Data set for evaluating the sensitivity to image size with zero (SZ-XX-0P, where XX is the image size), one (SZ-XX-1P), or 10 peaks (SZ-XX-10P).	4b
SNB	50^2	Synthetic	9	Data set to evaluate the sensitivity to the number of peaks (SNB-XXP, where XX is the number of peaks: $\{0, 5, 10, \ldots, 40\}$).	4c



(a) S-10R-1 (b) SZ-60-10P-1 (c) SNB-40P-1

Figure 4: Synthetic data set examples (see Table 1).

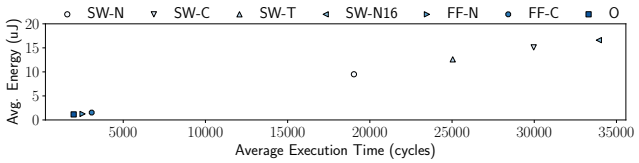


Figure 5: Execution time vs. energy consumption.

against a pre-generated list of peaks to ensure correctness. The simulator outputs the number of elapsed cycles and a simulation waveform which is input to the energy consumption analysis tool.

To find the energy consumption we used Joules [2], which is an RTL-level energy estimation tool from Cadence. Joules synthesizes the RTL code to a gate-level netlist and builds a realistic clock tree for the design. It then takes the simulation waveforms as input, and applies them to the synthesized design to determine the energy consumption of the system, including RAMs, standard cells, and analog modules. In addition to the energy consumption, Joules also estimates the area consumed by FF-N and FF-C. In order to remove the static current consumption from the current simulation, an idle simulation is also performed and subtracted from the value. In this simulation, the CPU has executed the "wait for interrupt" instruction, the rest of the system is idle, and all clocks are running.

5 RESULTS

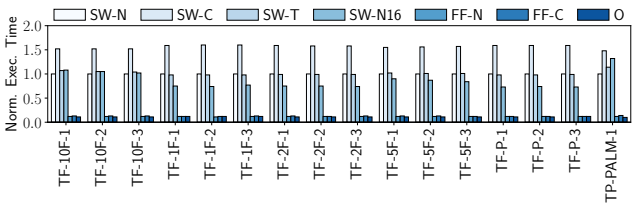
In this section, we present the performance and energy consumption results for the peak detection schemes introduced in Section 3. As aforementioned, we focus on peak detection and report the performance and energy consumption of this step. For the software-based schemes, energy measurements include the RISC-V core, scratchpad memory, and data transfers. Similarly, the energy numbers for the FF variants include the accelerator, memory, and data transfers. We name the input sets on the form dataset-subtype-ID (see Table 1). For example, TF-10F-1 is of type tablet-sized screen with fingers (TF), subtype 10 fingers (10F), and is dataset number 1.

5.1 Performance and Energy Consumption

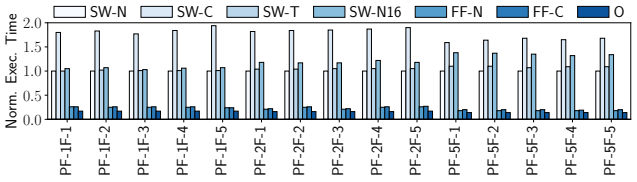
Figure 5 presents the average execution time (x-axis) and energy consumption (y-axis) of all the peak detection schemes we considered across all data sets, both realistic and synthetic. The key take-away is that the hardware accelerators substantially improve performance and reduce energy consumption compared to the software-based schemes. Also, energy consumption and performance is strongly correlated — with the evaluated schemes being placed on a diagonal line. The reason is that the average power consumption of our system was very similar for all schemes. Thus, execution time determines energy consumption since the average energy consumption is the product of the average power consumption and execution time.

FF-N and FF-C are both very close to the oracle scheme (i.e., O), but FF-N is marginally better than FF-C. The reason is that the overhead of maintaining the cache outweighs the benefit of reducing the number of memory accesses. Similarly, SW-N outperforms the other software-based schemes by a significant margin. Compared to SW-C and SW-T, the reason is again that the overhead of maintaining the cache outweighs its benefits. SW-T performs better than SW-C because it only updates the cache for values that are above the noise threshold. For SW-N16, its poor performance is due to the shifting and masking required to separate the two 16-bit words from the 32-bit word when comparing image values.

Figure 6a and 6b shows the performance of the different peak detection schemes for all realistic data sets and a large and a small touch screen, respectively. We normalize the results to SW-N — since this is the most efficient software scheme. The figures reinforce the key take-away from Figure 5: The hardware accelerators are significantly faster than the software-based schemes and perform very close to the oracle scheme across all data sets. The SW-C algorithm consistently performs 50% worse than SW-N due to cache



(a) Tablet-sized touch screen.



(b) Phone-sized touch screen.

Figure 6: Execution time normalized to SW-N for the realistic data sets.

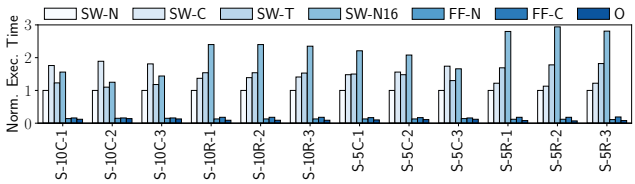


Figure 7: Execution time normalized to SW-N for the synthetic data sets.

management overheads, while SW-T performs similarly to SW-N since most nodes are below the noise threshold. SW-N16 outperforms SW-N when few nodes are over the threshold because it can discard two nodes for a single memory access in this case.

The normalized execution time for the synthetic data sets (i.e., S in Table 1) are shown in Figure 7. The key take-away from this figure is that execution time is highly data-dependent. The hardware accelerators FF-N and FF-C still outperform the software-based schemes, but the performance difference between the different software-based schemes are larger than for the realistic data sets. SW-C performs worse than SW-N in all cases. However, the performance of SW-C approaches that of SW-N when more nodes are over the threshold since caching becomes more beneficial in this case. SW-T and SW-N16 have the opposite behaviour — execution time increases when more nodes are above the threshold because this enables cache management (SW-T) and necessitates unpacking values (for SW-N16).

5.2 Simulation vs. ASIC

To validate the simulated energy consumption, we ran all SW schemes and data sets on prototype silicon of the touch controller and recorded their energy consumption (the prototype did not contain an FF accelerator). The prototype uses the same standard

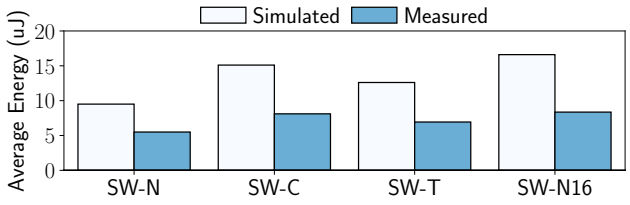


Figure 8: Simulated vs. measured energy consumption.

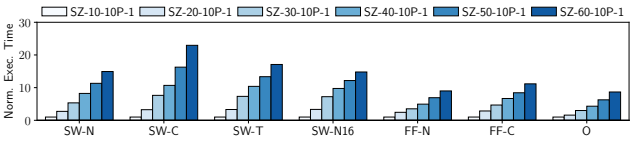


Figure 9: Image size scalability with synthetic data sets.

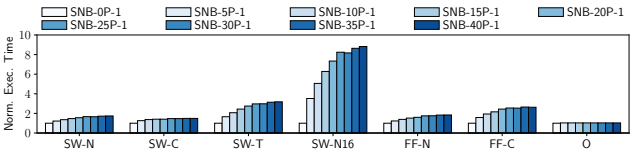


Figure 10: Scalability with respect to the number of peaks.

cells, RAMs and other building blocks as was used in the power simulation. Figure 8 shows the difference between simulation and measured energy consumption. The simulated values are almost 2× of the measured values for all algorithms. This constant scaling between the measured and simulated is to be expected when using an RTL energy estimator, as there are many factors in the measurements that do not match simulator assumptions (e.g., temperature, voltage, process corner, standard cell characterization, and simulated clock tree). That said, the prototype measurements lead to the exact same ranking as the simulator — confirming that our simulator-based approach captures the key trends.

5.3 Sensitivity Analysis

Figure 9 evaluates the scalability of the peak detection schemes with respect to image size for 10-peak images. We normalize the results to the 10^2 image size for all schemes. There are two key underlying trends. The number of data elements grows quadratically, but the number of elements above the noise threshold remain constant since we do not change the number of peaks. Thus, for the schemes where the overhead is driven by the number of elements (e.g., SW-C) there is a clear quadratic trend. For the other schemes, this effect is much less pronounced because values below the threshold are quickly discarded.

Figure 10 shows how the schemes scale with respect to the number of peaks in an image. All schemes show the same behaviour where the execution time saturates when the number of peaks increase. This is caused by almost all nodes are over the threshold and

must be evaluated when there are 30 or more peaks. The aforementioned unpacking overheads of the SW-N16 scheme causes severe performance reduction when many nodes are over the threshold.

6 RELATED WORK

The end of Dennard scaling has led to a significant interest in using accelerators to improve the performance and energy-efficiency of computing systems. Most prior work (e.g., [6, 12]) use accelerators to improve power-efficiency and thereby improve performance. This strategy is possible because high-performance systems mostly operate under a fixed power budget — meaning that improving power-efficiency enables higher performance.

In the embedded domain, the primary benefit of accelerators is to reduce energy consumption and thereby improve battery-life. For wearables, Stich [15] uses small configurable accelerators within each core of a multi-core to create large virtual accelerators on demand, while XPro [16] is an accelerator infrastructure for classification and data aggregation. Within robotics, Sacks et al. [14] and Murray et al. [11] propose accelerators for motion planning and control algorithms. Zhang et al. [22] exploit cross-frame similarities to improve the energy-efficiency of video streaming on handheld platforms, and the Galois Field processor [3] accelerates block coding and cryptography kernels in computing systems for the Internet of Things. Yazdani et al. [21] accelerate automatic speech recognition, and PULP [5] is an accelerator for embedded computer vision. These works share our motivation — using specialization to reduce energy consumption — but target different applications.

Prior work on touch screen controllers have mainly focused on improving the signal-to-noise ratio during IA and are therefore orthogonal to this work. Ragheb et al. [13] present an IA-approach based on differential sensing. Within differential sensing IA units, Won and Kim [17] propose an offset compensation technique, and Kim et al. [9] present a touch position recovery algorithm. A number of works have focused on IA units based on frequency division concurrent sensing. For instance, Kim et al. [8] present an interleaved sine wave generator while Choi et al. [4] present a hardware implementation of a fast Fourier transform.

A number of researchers have proposed complete touch screen controllers (e.g., [7]). These works focus on the IA-unit while the IPP step is implemented with a commodity microcontroller or in an FPGA. Thus, these works differ from ours since they do not evaluate different IPP implementation options.

7 CONCLUSION

In this paper, we have thoroughly evaluated software and hardware schemes for peak detection — the subsystem responsible for identifying touches in capacitive touch screen controllers. We find that peak detection lends itself to acceleration, and our Finger Finder accelerator (i.e., FF-N) reduces the energy consumption by 7.6× compared to the best software-based scheme. If adding an accelerator is not desirable, we find that the naive software implementation (i.e., SW-N) is the best choice. The reason is that the overheads of implementing caching in software are significant and outweigh the benefits. Since peak detection accounts for approximately 2.5% of the total energy consumption of our touch screen controller, we will direct our future efforts towards subsystems that have a more significant impact on overall energy consumption.

ACKNOWLEDGMENTS

This work has been supported by RFF Midt-Norge (project #272179).

REFERENCES

[1] Cadence. 2018. Incisive Enterprise Simulator. https://www.cadence.com/.
[2] Cadence. 2018. Joules RTL Power Solution. https://www.cadence.com/.
[3] Yajing Chen, Shengshuo Lu, Cheng Fu, David Blaauw, Ronald Dreslinski, Jr., Trevor Mudge, and Hun-Seok Kim. 2017. A programmable Galois field processor for the internet of things. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 55–68.
[4] G. Choi, M. G. A. Mohamed, and H. Kim. 2016. New FFT design with enhanced scan rate for frequency division concurrent sensing of mutual-capacitance touch screens. In *Int. Conf. on Electronics, Information, and Communications (ICEIC)*.
[5] Francesco Conti, Davide Rossi, Antonio Pullini, Igor Loi, and Luca Benini. 2016. PULP: A ultra-low power parallel accelerator for energy-efficient and flexible embedded vision. *Journal of Signal Processing Systems* 84, 3 (2016), 339–354.
[6] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. 2010. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the Int. Symp. on Computer Architecture (ISCA)*. 37–47.
[7] H. Kim, Y. Choi, S. Byun, S. Kim, K. Choi, H. Ahn, J. Park, D. Lee, Z. Wu, H. Kwon, Y. Choi, C. Lee, H. Cho, J. Yu, and M. Lee. 2010. A mobile-display-driver IC embedding a capacitive-touch-screen controller system. In *International Solid-State Circuits Conference (ISSCC)*. 114–115.
[8] J. Kim, M. G. A. Mohamed, and H. Kim. 2015. Design of a Frequency Division Concurrent sine wave generator for an efficient touch screen controller SoC. In *International Symposium on Consumer Electronics (ISCE)*. 1–2.
[9] Ji-Ho Kim, Dong-Min Won, and HyungWon Kim. 2016. Touch Position Recovery Algorithm for Differential Sensing Touch Screen. *Journal of information and communication convergence engineering* 14, 2 (2016), 106–114.
[10] M. G. A. Mohamed, Unyong Jang, Incheol Seo, HyungWon Kim, Tae-Won Cho, Hyeoung Kyu Chang, and Sunou Lee. 2014. Efficient algorithm for accurate touch detection of large touch screen panels. In *Int. Symp. on Consumer Electronics*.
[11] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin. 2016. The microarchitecture of a real-time robot motion planning accelerator. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 1–12.
[12] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark A. Horowitz. 2013. Convolution engine: Balancing efficiency & flexibility in specialized computing. In *Proc. of the Int. Symp. on Computer Architecture (ISCA)*.
[13] A N Ragheb, M G A Mohamed, and HyungWon Kim. 2016. Differentiator Based Sensing Circuit For Efficient Noise Suppression of Projected Mutual-Capacitance Touch Screens. In *Int. Conf. on Electronics, Information, and Communications*.
[14] J. Sacks, D. Mahajan, R. C. Lawson, and H. Esmaeilzadeh. 2018. RoboX: An end-to-end solution to accelerate autonomous control in robotics. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 479–490.
[15] Cheng Tan, Manupa Karunaratne, Tulika Mitra, and Li-Shiuan Peh. 2018. Stitch: Fusible heterogeneous accelerators enmeshed with many-core architecture for wearables. In *International Symposium on Computer Architecture (ISCA)*. 13.
[16] Aosen Wang, Lizhong Chen, and Wenyao Xu. 2017. XPro: A cross-end processing architecture for data analytics in wearables. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 69–80.
[17] Dong-Min Won and HyungWon Kim. 2018. Enhancement of touch screen sensing based on voltage shifting differential offset compensation. *Analog Integrated Circuits and Signal Processing* 94, 2 (2018), 205–215.
[18] Kaige Yan, Xingyao Zhang, and Xin Fu. 2015. Characterizing, modeling, and improving the QoE of mobile devices with low battery level. In *Proceedings of the Int. Symp. on Microarchitecture (MICRO)*. 713–724.
[19] I. Yang and O. Kwon. 2011. A touch controller using differential sensing method for on-cell capacitive touch screen panel systems. *IEEE Transactions on Consumer Electronics* 57, 3 (2011), 1027–1032.
[20] YANO Research. 2017. Capacitive touchscreen (touch panel)/components global market: Key research findings 2017. https://www.yanoresearch.com/.
[21] R. Yazdani, A. Segura, J. M. Arnau, and A. Gonzalez. 2016. An ultra low-power hardware accelerator for automatic speech recognition. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 1–12.
[22] Haibo Zhang, Prasanna Venkatesh Rengasamy, Shulin Zhao, Nachiappan Chidambaram Nachiappan, Anand Sivasubramaniam, Mahmut T. Kandemir, Ravi Iyer, and Chita R. Das. 2017. Race-to-sleep + content caching + display caching: A recipe for energy-efficient video streaming on handhelds. In *Proceedings of the Int. Symp. on Microarchitecture (MICRO)*. 517–531.