

UNIVERSITY OF CALIFORNIA

Los Angeles

Time in Wireless Embedded Systems

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical Engineering

by

Thomas Schmid

2009

UMI Number: 3424179

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 3424179

Copyright 2010 by ProQuest LLC.

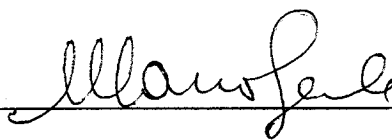
All rights reserved. This edition of the work is protected against unauthorized copying under Title 17, United States Code.



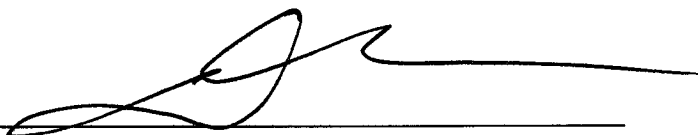
ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

© Copyright by
Thomas Schmid
2009

The dissertation of Thomas Schmid is approved.



Mario Gerla



Deborah Estrin



William J. Kaiser



Mani B. Srivastava, Committee Chair

University of California, Los Angeles

2009

*To my wife ...
to my mother and father ...
to my sister and brother ...
for their endless encouragement and patience.*

TABLE OF CONTENTS

1	Introduction	1
1.1	A Short History of Keeping Time	1
1.2	Accurate Time in Modern Embedded Systems	3
1.2.1	LC/RC Circuits	6
1.2.2	Ring Oscillator	8
1.2.3	Quartz Crystal	8
1.2.4	MEMS Resonators	13
1.2.5	Other Types of Resonators	13
1.3	Contributions	14
2	Dual Crystal Designs for Temperature Induced Frequency Error Compensation	16
2.1	Introduction	16
2.1.1	Related Work	17
2.1.2	Novel Approach	19
2.2	Introducing Differential Drift	21
2.3	The Crystal Compensated Crystal-Based Timer (XCXT)	23
2.3.1	Calibration Phase	24
2.3.2	Compensation Phase	26
2.3.3	Corrected Timestamping	28
2.3.4	Performance Evaluation	29

2.3.5	Model Fitting Effects	32
2.3.6	Quantization Effects	32
2.3.7	Other Effects	35
2.4	Algorithm Evaluation	36
2.4.1	Environment Emulator	37
2.4.2	Testbed	37
2.4.3	Interrupt Latency Considerations	39
2.4.4	Limitations	39
2.4.5	Calibration	40
2.4.6	Compensation	40
2.4.7	Experimental Results	43
2.5	XCXT Hardware Implementation	47
2.5.1	XCXT Power Measurements	49
2.5.2	Microprocessor Power Consumption	49
2.5.3	Oscillator and Timer Power Consumption	51
2.6	Discussion	54
2.6.1	Crystal Compensated Crystal Oscillator (XCXO)	54
2.6.2	Cost – Performance Comparison	54
2.6.3	The Smart Timer Unit	56
2.7	Summary	59
3	The Effect of Temperature on Time Synchronization	60

3.1	Introduction	60
3.1.1	Contribution	62
3.2	Understanding Power Consumption in Duty-Cycled Devices	62
3.2.1	Clock Stability and Duty Cycling	63
3.3	Clock-Characteristic Impacts on Resynchronization Rate	69
3.3.1	Temperature Effects on Frequency Error	70
3.3.2	Time Synchronization Error	70
3.3.3	A More Realistic Bound on the Time Synchronization Error	75
3.3.4	Verification Through Simulation	77
3.4	Bringing it All Together: Synchronization, Duty-Cycling, and Power	78
3.5	Summary	80
4	Temperature Compensated Time Synchronization	82
4.1	Introduction	82
4.2	The Cost of Measuring Temperature	85
4.3	Frequency Error Estimation	87
4.4	TCTS Algorithm	90
4.4.1	Calibration	91
4.4.2	Compensation	91
4.5	Simulation Results	92
4.5.1	Robustness Towards Communication Loss	94
4.6	Summary	98
5	Architectural Support for Increased Time Accuracy	99

5.1	Introduction	99
5.1.1	Architectural Requirements	100
5.2	The Cost of Time Synchronization	102
5.2.1	The Flooding Time Synchronization Protocol	103
5.2.2	Time Synchronization Power Model	106
5.2.3	Power – Frequency Stability Equilibrium	108
5.2.4	Concrete Example using the Epic Sensor Network Platform	109
5.3	Exploiting Dual-Clock Hardware Designs for Increased Time Resolution	110
5.3.1	Sub- μ Second Time Synchronization Using the Dual-Clock Approach	112
5.4	A Power-Aware Timer Module, the HLTimer	114
5.5	Summary	118
6	Discussion & Conclusion	120
6.1	Low-Power, High-Accuracy Time in Embedded Systems	120
6.2	Looking Ahead	123
	References	124

LIST OF FIGURES

1.1	Basic block diagram of a clock circuit and associated timer hardware.	3
1.2	Relative Frequency Drift vs Cost of commercially available clock sources. Data gathered from different vendors in Fall 2007.	5
1.3	Picture of a tuning fork 32kHz crystal commonly found in embedded systems to keep wall time. They are small, and consume $< 15\mu W$. . .	9
1.4	Frequency Drift vs Temperature for two AT-cut crystals. The crystals are cut at an angle of $35^{\circ}20' + \theta$	11
2.1	Frequency Drift vs Temperature for an uncompensated AT-cut quartz crystal oscillator and temperature compensated crystal oscillator (TCXO).	18
2.2	Differential Frequency Drift vs Temperature for multiple pairs of differently AT-cut oscillators. Note that the steeper the slope, the better it is for our compensation algorithm.	20
2.3	Frequency Drift versus Difference of Frequency Drift for several pairs of AT-cut oscillators. The larger the full span of δf_{12} the better our algorithm can compensate.	22
2.4	Frequency Drift versus Temperature characteristics for Uncompensated, TCXO, Cubic Fit XCXT and Lookup Table XCXT	30
2.5	Frequency Drift versus Temperature characteristics for Uncompensated, Temperature Compensated, Cubic Fit based Compensation and Lookup Table based Compensation	31
2.6	Frequency Drift versus Temperature characteristics for Clock Source 2 showing minimal effects due to quantization when $F_0/F_s = 50 \times 10^6$.	33

2.7	Frequency Drift versus Temperature characteristics for Clock Source 2 showing detrimental effects of quantization when $F_0/F_s = 50 \times 10^3$	34
2.8	Block diagram of the implementation of the XCXT.	37
2.9	Experimental setup with the two Agilent waveform generators, the 2Hz reference clock, and the MSP430 microcontroller with its two timer units.	38
2.10	Measured calibration and its cubic curve fit. Note the quantization effect which comes from measuring frequency difference with digital counters. The effect on error of this is discussed in Section 2.3.6.	41
2.11	Block diagram of our compensation experiment.	42
2.12	Compensated clock drift over the full temperature range of -40° to 75° Celsius. We also show the drift of the two uncompensated crystals used in the compensation algorithm.	43
2.13	Frequency stability expressed by the Allan Variance for the compensated and uncompensated clock. We can see that we trade short term vs. long term stability by employing our compensation algorithm.	45
2.14	Difference between global time and the estimated time at node 1 over 2.5 hour and a change in temperature from -40° to 75° Celsius. The different phases come from the inaccuracy introduced by the frequency generators, which not always set the output frequency to the desired frequency (see Figure 2.12).	46
2.15	Using the LUT from the calibration data, we can successfully compensate one of the crystals for its temperature drift. Over the full temperature range of -10°C to 60°C we measured a standard deviation of 0.31ppm.	48

2.16	This is a typical current consumption plot of the XCXT. The measurements were taken at 3V. The spikes represent instances when the microprocessor woke up because of a timer overflow which has to be treated.	50
2.17	These are the typical current consumptions of the clock and timer subsystem of the MSP430. The clock source was an 8MHz crystal, and the different clock speeds were achieved by dividing that signal using an internal clock divider. As comparison, we also measured the MSP430 power consumption when it uses a 32kHz crystal, one timer active, CPU turned off.	52
2.18	Block diagram of an ideal hardware base Crystal Compensated Crystal Oscillator (XCXO).	55
2.19	This is the timeline of the smart timer unit. We can find the phase of the slower timer (red, dashed) since the time instance t_s by counting the number of ticks of the fast clock (blue, solid)	57
3.1	Duty-Cycling performance of SCP-MAC using different clock source stabilities. Note that with a 1ppm clock source, the difference between piggybacking the sync messages on regular data messages or not virtually disappears.	65
3.2	Worst case time synchronization error for four different clock speeds. The two low frequency clocks are tuning fork crystals with a maximum frequency error of 120 ppm, and the high frequency clocks are AT-cut crystals with a maximum frequency error of ± 50 ppm.	72

3.3	This graph shows the effect of using the maximum drift change calculated from a temperature traces on the maximum synchronization error. In general, the smaller the temperature changes, the better the synchronization accuracy for longer resynchronization intervals. . . .	76
3.4	This graph compares the theoretical upper bound to the 95% confidence interval found through simulating a time synchronization protocol. . .	77
3.5	This graphs shows the average power consumption for different synchronization techniques in a duty cycled system. With ideal clocks, no overhead is required, and thus a minimum power consumption is achieved. We only show the sun exposed data set because it is a worst case for the three temperature traces we collected.	79
4.1	Current profile of a TMote Sky while sampling the internal temperature sensor. It takes 35ms to wakeup the CPU, stabilize the internal voltage reference, take the sample, and shut down the CPU again. During that time, the node consumes a total of 66.5 μ J, or about 10% of what a radio message consumes (~600 μ J).	85
4.2	This is the current consumption of a TMote Sky while sampling the external SHT11 temperature sensor. It takes a total of 220ms or about 287 μ J.	86
4.3	The error in the frequency error estimation of a sensor node is hampered by two different phenomenas: (1) quantization due to the digital nature of a clock, and (2) temperature induced frequency drift. Simulation shows that there is an optimal resynchronization period at which estimation error is minimized.	88

4.4	This graph shows the hourly average resynchronization time. In the first 24h, TCTS calibrates the local clock, and thus uses a very short synchronization period. Once calibrated, the resynchronization time rapidly increase and thus saves power and communication overhead.	93
4.5	If one ignores the one-time calibration overhead, TCTS quickly outperforms FTSP in terms of accuracy with long resynchronization intervals. This is for a 32kHz tuning fork crystal.	95
4.6	Illustration of what happens if time synchronization is stopped. We can clearly see how FTSP's performance worsens once the temperature starts to change, whereas TCTS keeps the synchronization accuracy at a much higher level.	96
4.7	TCTS estimates the frequency drift function $g(\kappa)$ for the current temperature, and stores it in a calibration table. In our simulation, $g(\kappa)$ is a quadratic curve, as found in the tuning fork crystals. Even with only a 0.25C temperature resolution, the performance of TCTS is extremely high.	97
5.1	Time synchronized Quanto Activity log of three nodes participating in time synchronization. Node 1 sent out a timestamped broadcast message. Node 2 and 3 receive the message and process it, since Node 1 is their synchronization root node.	105
5.2	Time synchronized Quanto Activity log of three nodes participating in time synchronization. Node 3 sent out a timestamped broadcast message. However, since 3 is not the synchronization parent of node 1 or 2, they both just receive and then discard it.	106

5.3 Result of Equation 5.6 using the Epic platform as an example. We can observe that for dense networks ($N > 75$ nodes), a TCXO with a power consumption of around $200\mu W$ starts to be a viable solution to improve the power consumption. 109

5.4 Power consumption of a dual-clock hardware design given a specific duty-cycle of the radio, assuming the slow clock consumes $56\mu W$ and the fast clock $660\mu W$ 111

5.5 Performance of a dual-clock enhanced FTSP implementation. The hardware contains one 8MHz and one 32kHz crystal. The radio chip is a TI CC2420, using the start of frame delimiter (SFD) of the 802.15.4 messages for time stamping. On average, the accuracy is well below $1\mu s$ 113

5.6 This is a simplified block diagram of the main HLTimer components. The main counter is a 16-bit counter that is fed by the high frequency clock signal HCLK. The low frequency clock LCLK drives the rest of the logic and increments the different counting registers LTC and HTC based on the HLTimer’s algorithm. 115

5.7 The power consumption of the HLTimer abruptly increases if the resolution is set to high. The effect is that the counting in the HTC register becomes much finer, and thus high resolution time becomes available. 117

LIST OF TABLES

1.1	Comparison of different Resonating Elements	7
2.1	Quantization Effects on Compensation Gain	35
5.1	Different Time Constants	107
5.2	Different Average Power Consumptions	110

ACKNOWLEDGMENTS

The title page of this doctoral dissertation contains only one name. However, this dissertation would not have been successful without the help and guidance of many others to whom I am endlessly indebted. I am most especially grateful to my parents, Martha and Heinrich Schmid, for their guidance, support and encouragement to this great challenge. My wife, Sieglinde, for always being there in the good and bad times. My brother and sister, Marcel and Barbara, who's curiousness in my work encourages me to strive further in my research.

During my work at UCLA, I had the fortunate opportunity to work together with some brilliant people and many lifelong friendships have developed in the process. I want to specially thank Young Cho for bringing me on the path of investigating time in embedded systems. I also want to thank Roy Shea, Jonathan Friedman, and Zainul Charbiwala, without who's innumerable discussion, both academic and otherwise, this work would never have achieved the level it did.

I had the privilege of collaborating in many situations with numerous people from other Universities, and even continents. In particular, I would like to thank Martin Vetterli for his encouragement during my time at EPFL to explore another University campus, and his guidance throughout the numerous internships in his laboratory. I am also thankful to Prabal Dutta for the innumerable discussions we had on the importance of time.

Last, but not least, I would like to thank my advisor and mentor, Mani Srivastava. His guidance, breadth of knowledge, and patience with my work never ended to amaze me. He built an extraordinary environment for creativity and innovation at the Networked and Embedded Systems Lab, and I am thankful that he gave me the chance to participate and further the development of my own skills and knowledge within that environment.

VITA

- 1980 Born, Arbon, TG, Switzerland.
- 1985-1987 Kindergarten, Horn, TG, Switzerland.
- 1987-1995 Elementary and Secondary School, Horn, TG, Switzerland.
- 2000 Matura, Kantonsschule Romanshorn, TG, Switzerland.
- 2002-2003 Exchange Year McGill University, Montreal, Quebec, Canada.
- 2004-2005 Student Researcher at Audiovisual Communications Laboratory (LCAV), EPFL, Lausanne, Switzerland
- 2005 M.Sc. Communication Systems Engineering, Ecole Polytechnique Fédéral de Lausanne (EPFL), Switzerland
- 2005-present Graduate Research Assistant, Electrical Engineering Department, UCLA.
Chancellor's Prize Fellowship
- 2007 Summer Research Intern, LCAV, EPFL, Lausanne, Switzerland.
- 2008-2009 Dissertation Year Fellowship, UCLA

PUBLICATIONS

- Dustin Torres, Jonathan Friedman, Thomas Schmid, Mani B Srivastava, *Software-Defined Underwater Acoustic Networking Testbed*, In Proceedings of WUWNET, 2009
- Thomas Schmid, Dustin Torres, Mani B Srivastava, *Demo Abstract: Low-power High-precision Timing Hardware for Sensor Networks*, (Demo Abstract), In Proceedings of SenSys, 2009
- Thomas Schmid, Zainul M Charbiwala, Roy S Shea, Mani B Srivastava, *Temperature Driven Time Synchronization*, IEEE Embedded Systems Letters, 2009
- Younghun Kim, Thomas Schmid, Zainul M Charbiwala, Mani B Srivastava, *ViridiScope: Design and Implementation of a Fine Grained Power Monitoring System for Homes*, In Proceedings of the 11th international conference on Ubiquitous Computing (UbiComp), 2009
- Jonathan Friedman, Thomas Schmid, Zainul M Charbiwala, Mani B Srivastava, Young H Cho, *Multistatic Pulse-Wave Angle-of-Arrival-Assisted Relative Interferometric RADAR*, In Proceedings of IEEE Radar Conference, 2009
- Thomas Schmid, Zainul M Charbiwala, Jonathan Friedman, Young H Cho, Mani B Srivastava, *Exploiting Manufacturing Variations for Compensating Environment-induced Clock Drift in Time Synchronization*, In Proceedings of ACM Sigmetrics, 2008
- Thomas Schmid, Jonathan Friedman, Zainul M Charbiwala, Young H Cho, Mani B Srivastava, *Low-Power High-Accuracy Timing Systems for Efficient Duty Cycling*, In Proceedings of ISLPED, 2008

- Thomas Schmid, Zainul M Charbiwala, Jonathan Friedman, Young H Cho, Mani B Srivastava, *The True Cost of Accurate Time*, In Proceedings of HotPower, 2008
- Thomas Schmid, Jonathan Friedman, Zainul M Charbiwala, Young H Cho, Mani B Srivastava, *XCXO: An Ultra-low Cost Ultra-high Accuracy Clock System for Wireless Sensor Networks in Harsh Remote Outdoor Environments*, In Proceedings of ISSCC/DAC, 2008
- Younghun Kim, Zainul M Charbiwala, Akhilesh Singhanian, Thomas Schmid, Mani B Srivastava, *SPOTLIGHT: Personal Natural Resource Consumption Profiler*, In Proceedings of HotEmNets, 2008
- Younghun Kim, Thomas Schmid, Zainul M Charbiwala, Jonathan Friedman, Mani B Srivastava, *NAWMS: Nonintrusive Autonomous Water Monitoring System*, In Proceedings of The 6th ACM Conference on Embedded Networked Sensor Systems (SenSys), 2008
- David Jea, Jason Liu, Thomas Schmid, Mani B Srivastava, *Hassle Free Fitness Monitoring*, In Proceedings of The 2nd International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments (HealthNet), 2008
- Jonathan Friedman, Zainul M Charbiwala, Thomas Schmid, Young H Cho, Mani B Srivastava, *Angle-of-Arrival Assisted Radio Interferometry (ARI) Target Localization*, In Proceedings of MILCOM, 2008
- G. Troxel, E. Blossom, S. Boswell, A. Caro, I. Castineyra, A. Colvin, T. Dreier, J. Evans, N. Goffee, K. Haigh, T. Hussain, V. Kawadia, D. Lapsley, C. Livadas, A. Medina, J. Mikkelson, G. Minden, R. Morris, C. Partridge, V. Raghunathan, R. Ramanathan, C. Santivanez, T. Schmid, D. Sumorok, M. Srivastava, B. Vincent,

- D. Wiggan, A. Wyglinski, and S. Zahedi, *Enabling Open-source Cognitively-controlled Collaboration among Software-Defined Radio Nodes*, Computer Networks: The International Journal of Computer and Telecommunications Networking (Special Issue on Cognitive Wireless Networks), 2008
- Kartik Ariyur, Thomas Schmid, Yunjung Yi, Zainul M Charbiwala, Mani B Srivastava, *On the Impact of Time Synchronization on Quality of Information and Network Performance*, In Proceedings of The Second Annual Conference of the International Technology Alliance (ACITA), 2008
 - Thomas Schmid, Roy S Shea, Jonathan Friedman, Mani B Srivastava, *Movement Analysis in Rock-Climbers*, (Demo Abstract), IPSN 2007 , 2007
 - Thomas Schmid, Oussama Sekkat, Mani B Srivastava, *An experimental study of network performance impact of increased latency in software defined radios*, In Proceedings of the the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization (ACM WINTeCH), 2007
 - Andrew Parker, Sasank Reddy, Thomas Schmid, Kevin Chang, Saurabh Ganerival, Mani Srivastava, Mark Hansen, Jeff Burke, Deborah Estrin, Mark Allman and Vern Paxson, *Network System Challenges in Selective Sharing and Verification for Personal, Social, and Urban-Scale Sensing Applications*, Record of the Fifth Workshop on Hot Topics in Networks (HotNets-V), 2006
 - Dustin McIntire, Kei Ho, Bernie Yip, Sasank Reddy, Thomas Schmid, Amarjeet Singh, Winston Wu, and William J. Kaiser, *Demonstration of The Low Power Energy Aware Processing (LEAP) Embedded Networked Sensor System*, (Demo Abstract), In Proceedings of Information Processing in Sensor Networks (IPSN), 2006

- S. Reddy, T. Schmid, A. Parker, J. Porway, G. Chen, A. Joki, J. Burke, M. Hansen, D. Estrin, and M. Srivastava, *UrbanCENS: Sensing with the Urban Context in Mind*, Proceedings of The Eight International Conference on Ubiquitous Computing (UbiComp), 2006
- G. Troxel, E. Blossom, S. Boswell, A. Caro, I. Castineyra, A. Colvin, T. Dreier, J. Evans, N. Goffee, K. Haigh, T. Hussain, V. Kawadia, D. Lapsley, C. Livadas, A. Medina, J. Mikkelson, G. Minden, R. Morris, C. Partridge, V. Raghunathan, R. Ramanathan, C. Santivanez, T. Schmid, D. Sumorok, M. Srivastava, B. Vincent, D. Wiggin, A. Wyglinski, and S. Zahedi, *Adaptive Dynamic Radio Open-source Intelligent Team (ADROIT): Cognitively-controlled Collaboration among SDR Nodes*, Proceedings of the First IEEE Workshop on Networking Technologies for Software Defined Radio (SDR) Networks, 2006
- Thomas Schmid, Ted Dreier, Mani B. Srivastava, *Software Radio Implementation of Short-range Wireless Standards for Sensor Networking*, (Demo Abstract), Proceedings of the 4th international Conference on Embedded Networked Sensor Systems (SenSys), 2006
- Thomas Schmid, Ted Dreier, Mani B. Srivastava, *Software Radio Implementation of Short-range Wireless Standards for Sensor Networking*, (Demo Abstract), Proceedings of the First ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (ACM WiNTECH), 2006
- T. Schmid, H. Dubois-Ferrere, M. Vetterli. *SensorScope: Experiences withh a Wireless Building Monitoring Sensor Network*, In The Workshop on Real-World Wireless Sensor Networks (REALWSN), 2005

ABSTRACT OF THE DISSERTATION

Time in Wireless Embedded Systems

by

Thomas Schmid

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2009

Professor Mani B. Srivastava, Chair

Wireless embedded networks have matured beyond academic research as industry now considers the advantages of using wireless sensors. With this growth, reliability and real-time demands increase, thus timing becomes more and more relevant. In this dissertation, we focus on the development of highly stable, low-power clock systems for wireless embedded systems. Wireless embedded networks, due to their wire-free nature, present one of the most extreme power budget design challenges in the field of electronics. Improvements in timing can reduce the energy required to operate an embedded network. However, the more accurate a time source is, the more power it consumes. To comprehensively address the time and power problems in wireless embedded systems, this dissertation studies the exploitation of dual-crystal clock architectures to combat effects of temperature induced frequency error and high power consumption of high-frequency clocks. Combining these architectures with the inherent communication capabilities of wireless embedded systems, this dissertation proposes two new technologies; (1) a new time synchronization service that automatically calibrates a local clock to changes in temperature; (2) a high-low frequency timer that allows a duty-cycled embedded system to achieve ultra low-power sleep, while keeping fine granularity time resolution offered only by high power, high frequency clocks.

CHAPTER 1

Introduction

1.1 A Short History of Keeping Time

Modern society relies on the ubiquitous availability of accurate time. Scheduling meeting appointments, lunch breaks, office opening/closing hours, or the availability of certain resources are based on the notion of time. We even teach the youngest members of our society how to read clocks and understand the concept of time [Lle92].

Time-telling has a long history. Around 3500 BC, the Egyptians relied on the sun to tell time. Obelisks and sundials were used in order to tell the approximate time of day. For the longest time, the sun and stars were the only measure of time available to humans.

The first mechanical time keepers did not appear until 1500 BC. Water clocks, some of the first mechanical time keepers, relied on a very simple principle. A tiny hole at the bottom of a big bucket lets water escape from it in a near constant stream of drops. Markings on the inside of the bucket indicate the passing of time. Priests used these primitive time keepers at night to schedule the temple rites and sacrifices at the correct hour.

It was not until 1583 AD that Galileo Galilei discovered that a pendulum period denotes constant time. This discovery revolutionized the accuracy of time keeping, and a new generation of clocks and watches was soon to follow.

At about the same time, 15th Century naval exploration drove time accuracy research. Finding the latitude of a ship out in the ocean was easily accomplished using a sextant, by measuring the position of the sun at midday, or the stars at night. However, longitude is much more difficult. You need a sextant, and the knowledge of accurate time. Several large accidents with many dead sailors happened back in those days, because captains miscalculated their position, and thus drove their boats onto reefs.

For that reason, in 1714, the British government established "The Board of Longitude", and offered a prize of £20,000, equivalent to about \$4,000,000 today, for the person who could localize a ship within 30 nautical miles. To achieve such a high accuracy, you needed a clock that could keep time to within 3 seconds per day. Something not possible with the primitive clocks available thus far.

It wasn't until 1736, when John Harrison, a self-educated English clockmaker, tested his marine chronometer "H1" at sea. The H1 successfully calculated the landfall of a ship, though the British Board of Longitude required a trip to America, in order to get the full prize money. Harrison didn't stop after the H1. With each iteration, Harrison improved the accuracy, and especially the size, of the previous model. However, the Board of Longitude never accepted his clock as "accurate enough", and thus never awarded the prize money to Harrison. As a matter of fact, the Board never awarded the prize to anyone, before it was dissolved in 1828, when the significant problem of determining the longitude was considered solved.

Two more radical inventions in the technology of time keeping had a similar effect on the precision of time. In 1918, Alexander M. Nicholson at Bell Telephone Laboratories patented the first crystal controlled oscillator, and in 1955, Louis Essen built the first caesium-133 atom based atomic clock at the National Physical Laboratory in the UK. Both these developments consisted of huge gains in time accuracy, and we wouldn't have the plethora of different devices today without these two achievements.

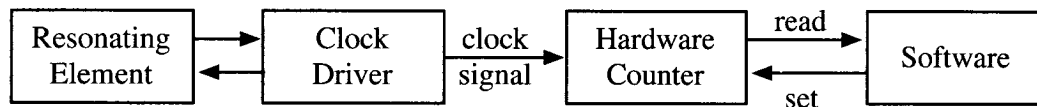


Figure 1.1: Basic block diagram of a clock circuit and associated timer hardware.

For a more detailed early history of time, please refer to the excellent book from Kristen Lippincott, "The Story of Time" [Lip99] and David Allans "The Science of Timekeeping" [AAH97].

1.2 Accurate Time in Modern Embedded Systems

Time in embedded systems is usually kept by a specialized sub-system illustrated in Figure 1.1. A periodic *clock signal* lies at the core of the system. The clock signal increments a hardware counter every $1/f$ seconds, where f is called the frequency of the clock signal. Therefore, at any time t since the n -bit counter was reset, the counter reads $c(t) = \lfloor f \cdot t \rfloor \bmod 2^n$. The floor operator $\lfloor \cdot \rfloor$ comes into play due to the digital nature of the hardware counter. The $1/f$ rate at which the counter is incremented is called the *resolution*. However, high resolution is not useful if the software cannot read the counter at that speed. Therefore, the smallest increment at which an application can read the counter is called *precision*. Finally, the counter is typically set to an international time calendar, e.g. UTC. The *accuracy* determines how true the counter holds to that calendar.

The clock signal is a periodic signal with some nominal frequency f_0 . Every clock signal will deviate from its intended nominal frequency for both dynamic (environmental changes, like pressure, temperature, acceleration) and static (imprecision in its manufacture) reasons. This deviation is termed *frequency error* (or inversely *frequency*

stability) defined as $f_e(t) = f_0 - f(t)$, where $f(t)$ is the frequency of the clock signal at time t . In general, this error is very small and is commonly expressed in a unitless quantity parts per million (ppm) derived from Equation 1.1.

$$f_e(t) = \frac{f(t) - f_0}{f_0} \cdot 10^6 \quad (1.1)$$

The frequency error of a clock signal changes over time. This change, called *frequency drift*, can be classified into two categories: short term (seconds to days) and long term (weeks to years). The observable short term changes are due to changes in the environment. Rapid acceleration, like in rockets, or changes of temperature and pressure on the circuit induces frequency change in the order of tens to hundreds of ppm. Long term changes occur as the clock ages. The physical stress induced on components over time can change their electrical properties, and thus introduces a frequency error on the order of a few ppm every year.

Figure 1.2 shows how the stability of a clock source is related to its price. The x -axis could represent other cost parameters as well, such as power consumption and/or size without appreciable change. To give a perspective on the values of drift, a 10ppm clock source would introduce a measurement error of over 5 minutes over a year. A 0.1ppm clock source on the other hand would be off by only about 3 seconds over a year.

The behavior of the frequency error largely depends on the underlying technology used to generate the clock signal itself. In general, two components are necessary to create a clock signal, a *resonating element*, and a *clock driver*. The resonating element is responsible to create an oscillation. The element by itself however does not sustain the oscillation and a clock drive circuit is necessary to initiate and sustain that oscillation. More specifically, it provides both feedback (must meet the Barkhausen [HC04] criteria with $gain \geq 1$, total loop phase $\geq 2\pi$) and isolation to the resonator. Although the

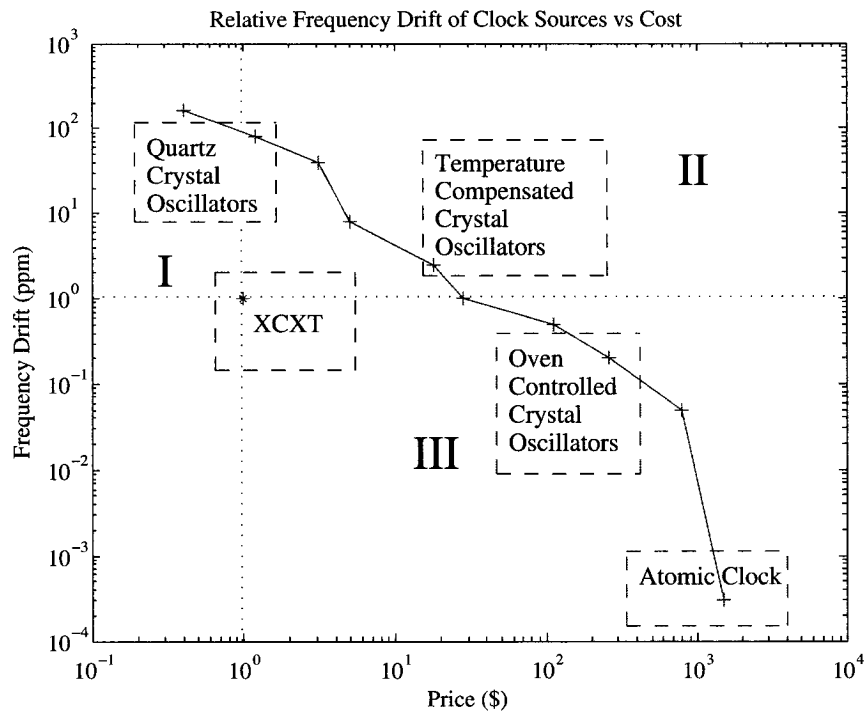


Figure 1.2: Relative Frequency Drift vs Cost of commercially available clock sources. Data gathered from different vendors in Fall 2007.

ideal driver varies by oscillator technology, CMOS buffers or inverters are generally well-suited since they have high input impedance (except w.r.t. the low impedance of quartz crystals excited into series resonance), high gain, and high bandwidth.

The choice of inverter, buffered or unbuffered, and the number of inverters in the feedback loop influences the characteristics of the clock signal, its drive strength, and the total power consumption of the clocking circuit. Often times, the clock driver is either integrated with the resonating element (usually called an oscillator) or it resides within the microprocessor. In the first case, the driver will have high drive strength in order to deliver a high quality clock signal to many different digital components, though it will consume a lot of power. In the second case, the clock driver is matched to the internal clock signal network of the microprocessor, and thus is optimized for power consumption. For example, a clock oscillator from Abracon at 16MHz consumes about 8.0mW while clocking a Texas Instrument MSP430 Microprocessor. This is about 20% more energy than if we clock the MSP430 directly using a resonating element (Citizen 16MHz crystal) and its internal clock driver, which we measured at 6.1mW.

The frequency error introduced by changes in the clock driver are usually not significant. It is in large part due to physical changes in the resonating element that the frequency of the clock signal changes over time. Therefore it is important to know the different technologies that can be used as resonating elements, since each of them has different performance characteristics. Table 1.1 summarizes the characteristics of the most common resonating elements, and the next subsections will briefly introduce each one of them.

1.2.1 LC/RC Circuits

One of the simplest resonating elements is an LC-circuit, consisting of one inductor (L) and one capacitor (C). An electric current can resonate between the two elements

Type	Stability	Power	Cost
LC/RC	1000's of ppm	Low	Cents to free
Ring Oscillator	1000's of ppm	Low	Cents to free
Crystal	10's of ppm <1ppm if controlled	Low - High Freq. Dependent	10's of cents to Dollars
Crystal Oscillator	<1ppm to 10's of ppm	Med - High	Dollars to 10's of Dollars
MEMS Resonator	10's to 100's of ppm	High	10's of Cents to Dollars

Table 1.1: Comparison of different Resonating Elements

at the circuit's resonant frequency $f = \frac{1}{2\pi\sqrt{LC}}$. LC oscillators offer comparatively low production cost and can be produced as an integrated circuit on a chip. However, the precision of the value of L and C and the effect of temperature on these values make this resonating element highly frequency unstable. Frequency errors in the order of thousands of ppm can be easily observed. LC circuits are now rarely used for modern digital applications.

A similar circuit can be produced using a resistor (R) and a capacitor (C). Its advantage over an LC resonator is a cheaper on-chip integration, because an inductor is large to produce compared to a resistor, and area on a chip costs money. Many modern microprocessors integrate such RC-type oscillators as a cheap alternative to external resonators. The Texas Instruments MSP430 families or the Atmel SAM3U are two such example. Like the LC circuit, the RC circuit uses passive components that are subject to similar degrees of inaccuracies. Thus, both resonators are only used in digital circuits where the frequency stability is not critical, like clocking the main CPU. One advantage of RC oscillators is their fast startup times of only a few μ seconds. This fast startup can lead to considerable power savings in duty-cycled systems.

1.2.2 Ring Oscillator

A ring oscillator is a device consisting of an uneven number of logical NOT gates. Each NOT gate has a specific transition time. Connecting an uneven number of them into a loop generates a clock signal with a frequency of $\frac{1}{n\tau}$ where τ is the transition time of one inverter. Thus, the frequency of a ring oscillator can be changed by revising the number of NOT gates in the circuit.

An advantage of ring oscillators is their very low price. Ring oscillators can even be synthesized in a FPGA. However, the frequency accuracy of a ring oscillator is very bad because the transition time of each NOT gate depends heavily on the applied voltage and temperature of the circuit. Frequency error is typically on the order of thousands of ppm. While the frequency drift of a ring oscillator is comparable to that observed in LC/RC circuits, a ring oscillator is typically smaller, cheaper to implement for integrated devices, and has a better output signal. Hybrid designs of ring oscillators and some sort of RC-oscillator can be found in some applications where the actual frequency isn't critical and where a cheap oscillator is necessary or desired.

1.2.3 Quartz Crystal

Quartz crystals are probably the most common resonating elements in today's digital systems. They are cheap and have a very stable frequency. The main working of the quartz crystal is a piezo-electric effect. Applying a voltage to a quartz crystal causes the crystal to mechanically deform, and vice-versa, if the quartz crystal is mechanically deformed then one can measure a voltage. By cutting a quartz crystal at a specific angle from its blank, very selective resonance frequencies can be obtained. There are many different ways of cutting these blanks, ranging from AT-cut, the doubly rotated SC-cut, to Y-cut. Each one of these cuts has specific properties, reacts differently to changes in

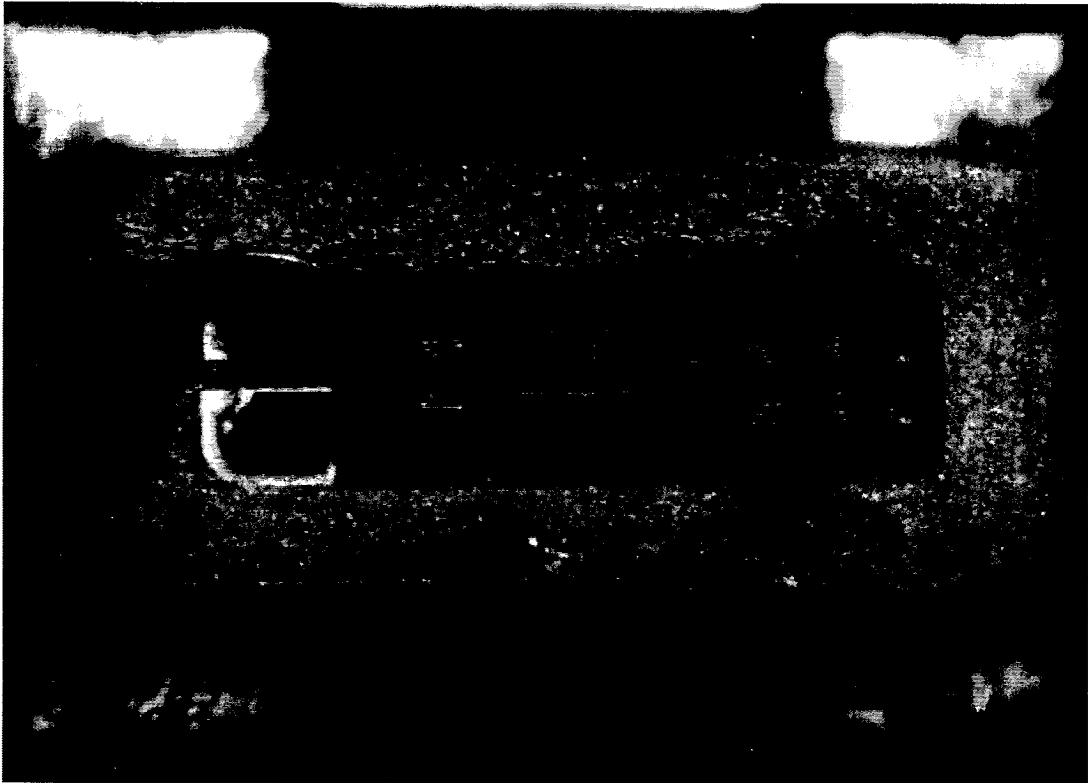


Figure 1.3: Picture of a tuning fork 32kHz crystal commonly found in embedded systems to keep wall time. They are small, and consume $< 15\mu W$.

the environment, and has different aging characteristics.

The primary causes of aging are mass transfer to or from the resonator's surfaces due to absorption or desorption of contamination, changes in the oscillator circuitry, and atomic level changes in the quartz material. Therefore, in order to achieve low aging, crystal units must be fabricated and hermetically sealed in an ultra-clean, ultra-high-vacuum environment. The aging rates of typical commercially available crystal oscillators range from 5ppm-10ppm per year for an inexpensive crystal, to 0.5ppm-2ppm per year for a temperature-compensated crystal, to 0.05ppm-0.1ppm per year for a temperature-controlled crystal [Vig92].

The most common low-frequency quartz crystals, up to about 100kHz, are the so called tuning fork crystals. They are called tuning fork because they have the same Y-shape and vibrate similar to a musical tuning fork. See Figure 1.3 for an example. The tuning fork crystal has a quadratic frequency error curve reaction to changes in temperature that ranges from about -120ppm to 10ppm, reaching the maximum at about room temperature. These crystals are commonly used in real time clocks to keep wall time during system sleep, because they consume very little power ($< 15\mu W$).

For higher frequencies, AT-cut crystals account for up to 75% of all quartz resonators made, due to their excellent frequency-temperature ($f - T$) characteristics. They come in frequencies ranging from 1MHz up to several hundreds of MHz. The AT-cut crystal exhibits a cubic frequency error curve reaction to changes in temperature, that ranges from $\pm 100ppm$ down to $\pm 20ppm$, depending on the quality.

To produce a quartz resonator, manufacturers cut out a tiny sheet from a quartz crystal at a specific shear angle. Different angles of cut produce vastly different crystal characteristics. The AT-cut crystal is cut at a nominal angle of $35^{\circ}20'$. The $f - T$ characteristics of AT-cut crystals is well studied in the literature and is found to follow

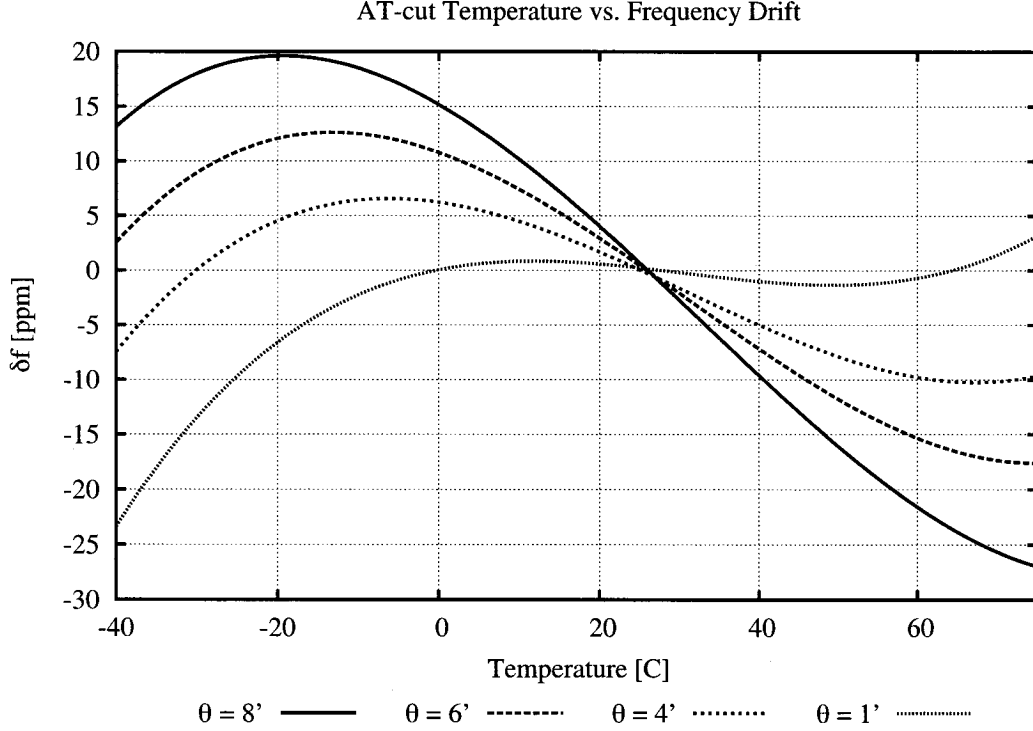


Figure 1.4: Frequency Drift vs Temperature for two AT-cut crystals. The crystals are cut at an angle of $35^{\circ}20' + \theta$

a third order polynomial using:

$$\delta f_{stability}(T) = A(T - T_0)^3 + B(T - T_0) + C \quad (1.2)$$

where A, B, C, T_0 are unique to each device. Interestingly, this is the key observation behind the compensation techniques detailed in Chapter 2, the value of B is extremely sensitive to any imprecision in the angle of the cut itself. Figure 1.4 shows how the $f - T$ characteristics vary for AT-cut crystals sheared with a slightly different angle of cut. This slight difference could be either deliberate or due to manufacturing variation. The key idea of Chapter 2 behind improving the stability of the clock source is to exploit this difference in $f - T$ characteristics for two sources to compensate one of them.

1.2.3.1 Compensation Techniques

Different techniques have been developed to make quartz crystals more stable with respect to environmental changes. The biggest effect on frequency stability are changes in temperature. Thus, many compensation techniques reverse this effect by measuring the temperature, and then tuning the crystal to the right frequency. These types of crystals are called Temperature Compensated Crystal Oscillators (TCXO), and there are many possible ways in how to do that [ZZX05]. The most advanced and precise crystals use a microcontroller and a doubly rotate SC-cut crystal. SC-cut crystals have the interesting behavior that they can be excited at two different frequencies at the same time. Since both frequencies have a different temperature stability curve, by measuring both frequencies in the microcontroller the output frequency can be tuned without directly measuring the temperature using a thermistor. This type of crystal is called Microcomputer Compensated Crystal Oscillator (MCXO [BMH89]). The major limiting factor on the attainable frequency stability of the TCXO or MCXO is hysteresis [KV90]. The lowest observed stability using the 10MHz/3.3MHz dual-mode MCXO was 1 part per billion (ppb). However, more typical TCXO's will have a limit of around 0.1 ppm. TCXOs can often be found in GPS applications, however their prohibitive high cost of \$10 to \$100 and the non-negligible power consumption makes them less appealing to general usage in wireless sensor networks.

Another possibility for temperature compensation is keeping the crystal at a stable temperature using an oven, instead of measuring the temperature. This type of crystal is called an Oven Controlled Crystal Oscillator (OCXO), and can achieve a stability of 1 to 5 ppb. However, that stability comes at a huge cost of energy, which typically ranges between 1 to 5 Watt. Such a high energy profile is clearly not suited for sensor networks. OCXOs are usually found in cellular phone basestations, where the high accuracy is needed for wireless communication purposes.

1.2.4 MEMS Resonators

One of the latest developments are Microelectromechanical System (MEMS) Resonators. The first MEMS resonators were built in 1967 [NNW67], though only recently became available to the general public. An advantage of the MEMS resonator is the possibility to directly integrate them into the CMOS process and thus allow a smaller PCB layout and better integrated systems. However, their power consumption is considerably higher than the power used for regular crystal oscillators. For example, we measured the power consumption of the SiTime 8002AI 16MHz MEMS Oscillator at 41mW without any digital circuit connected to it. This power consumption is outside of the realm of wireless sensor networks, though we can expect these power numbers to fall over the next few years, while MEMS resonators evolve further in research and are tailored towards low-power applications [RKM05].

1.2.5 Other Types of Resonators

There are a multitude of resonating elements that we have not yet discussed. Ceramic resonators, bulk acoustic wave (BAW) resonators, rubidium oscillator, atomic clocks, or opto-electronic oscillators can all be used to generate clock signals. These resonating elements are not a good fit for wireless sensor network applications because of their bad frequency stability, high price, or prohibitive power consumptions.

Research in resonators is still very active, and maybe someday every embedded device may contain an atomic clock itself. For Example, the National Institute of Standards and Technology (NIST) is working on a chip-scale atomic clock [KSS05]. It still consumes too much power (195 mW) to be a viable solution for sensor networks though one day this might be low enough to be put in each and every sensor device.

1.3 Contributions

The focus of this dissertation is on the investigation of a key source of timing error – the local clock (in)stability. Traditionally, this has required the use of expensive clock sources that are not cost or energy effective in low-end wireless sensor nodes. Besides developing novel clock sources, we can't forget about the inherent networked nature of wireless sensor networks, and thinking about ways of exploiting these aspects.

Our objective is to develop a new timing source for sensor networks using a novel cross-layer approach. The first contribution of this dissertation is a new way of temperature compensating a crystal oscillator using a technique called *Differential Drift*. As we will show in Chapter 2, this algorithm can outperform regular temperature compensated crystal oscillators by exploiting manufacturer variations in crystal oscillators. However, the Crystal Compensated Crystal based Timer (XCXT) does not exploit any communication capabilities of a sensor network platform, and thus still has to be factory calibrated during production.

The second contribution of this dissertation is a thorough investigation of the effects of temperature on networked time synchronization. In Chapter 3 we show why the common assumption of static drift during resynchronization intervals doesn't always hold true, and that time synchronization performance can drastically be impacted by the change of environmental temperature. Through the lessons learned from this investigation, Chapter 3 describes a new time synchronization protocol called Temperature Compensated Time Synchronization, which exploits the often present on-chip temperature sensor. With this additional knowledge, a node can elongate its resynchronization interval, and thus save energy and communication overhead.

The last contribution of this dissertation is the development of hardware architectural support for ultra low-power and high time-resolution time synchronization. A prototype

implementation of the device showed great promise, pushing wireless embedded time synchronization into a new dimension of sub-microsecond accuracies, with power consumptions close to a regular 32kHz tuning fork crystal. We do hope that this device will augment future sensor network architectures to provide them with unprecedented high accuracy of time, enabling researchers to improve localization, beam-forming, distributed logging mechanisms, and other algorithms that rely on the availability of highly accurate time.

CHAPTER 2

Dual Crystal Designs for Temperature Induced Frequency Error Compensation

2.1 Introduction

The single biggest impediment to a node's battery-powered lifetime is the energy spent during radio communication, and secondary to that, the time spent in its "awake" (as opposed to its low-power shutdown "sleep" state). Reduce these times, and lifetime improves substantially. However, as soon as nodes in the network begin sleeping and are correspondingly offline, other nodes in the network that are still awake can no longer use them as a communications hub to route sensor data back to a command-and-control station (referred to as a "gateway" node). To optimize sleep time and network performance simultaneously all of the nodes must synchronize their internal clocks and sleep and wake at the same time (we are aware that this statement is somewhat of a generality and numerous works on WSN scheduling exist, but our axiom to follow – that better synchrony yields better lifetime – still holds in these cases).

Commodity time references in WSN's, and indeed consumer and industrial products in general, typically consist of a quartz crystal driven by a Colpitts oscillator. This configuration is popular because it offers substantially better performance than switched resistive-capacitive networks and ceramic-based mechanical resonators at a moderate and tolerable marginal cost (see Figure 1.2). However, crystals drift in frequency as

their temperature changes. A survey of available HC-49S packaged crystals reveals that the majority of inexpensive parts experience a drift over the commercial temperature range (-20°C, +70°C) of ± 50 ppm. With this much deviation, nodes must spend at least 0.01% of their lifetimes awake [DCS07] in-order to guarantee that when they transmit, their intended receiver (whose clock has drifted differently) is awake to receive the broadcast. While this may sound acceptable, consider that a node that wants to sample water quality once per hour may only require a total of 100ms to wake, take the measurement, communicate any findings, and return to sleep. This corresponds to spending as little as 0.0028% of its lifetime awake. Repairing the temperature-imposed drift in the crystal-based clocks could improve network lifetime more than 3.5 times!

This chapter describes a novel clock system (hardware and software) that is composed of two crystal oscillators running in parallel at each node. It exploits the subtle manufacturing differences in each crystal that produce different drift-vs.-frequency behaviors. By measuring this difference it compensates for the drifting clock. In effect, it uses one crystal to compensate the other. We call this system as a whole the XCXT (Crystal Compensated Crystal based Timer). The notion of a timer comes from the subtle difference to regular oscillators or clocks. It provides not a corrected frequency (like a TCXO would), but rather it provides a corrected timer to a microcontroller. However, the XCXT architecture could easily be extended such that it provides a stable 1pps signal to an other device, which it already uses internally for corrections.

2.1.1 Related Work

Before we describe the software based compensation technique, it is noteworthy to mention the state-of-the-art in oscillator design. For mid-performance applications, like GPS receivers, designers prefer to use a temperature compensated crystal oscillator or TCXO [NB63, NH68b, ZZX05]. The approach followed by a TCXO manufacturer is to

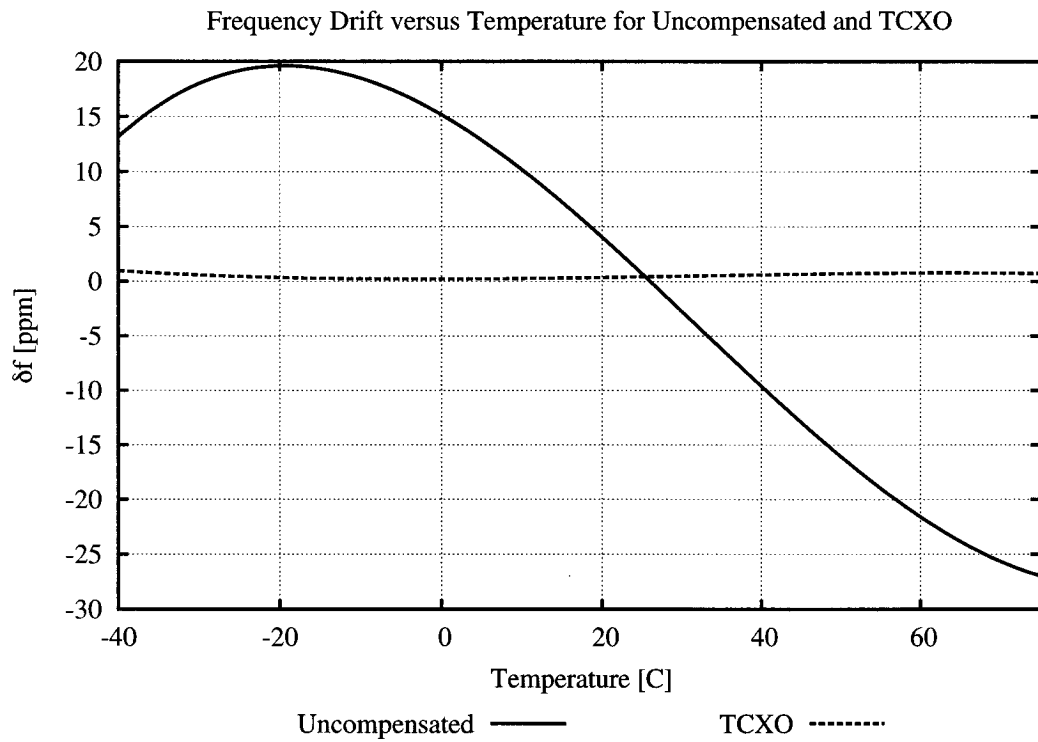


Figure 2.1: Frequency Drift vs Temperature for an uncompensated AT-cut quartz crystal oscillator and temperature compensated crystal oscillator (TCXO).

characterize each device in the factory to obtain its $f - T$ curve [SC01]. Then, by using a custom analog matching circuit [KS96] or a digital tuning circuit with a temperature sensor [CCD89, LHH00, LHT05], the $f - T$ characteristics are corrected by minuscule frequency adjustments to negate the effects of drift. Figure 2.1 shows the improvement in the drift performance of a commercially available digital TCXO [Max08] over an uncompensated oscillator. For high-performance applications, like GSM or CDMA cell phone basestation towers, designers employ an oven controlled crystal oscillator or OCXO which has an active mechanism of maintaining the temperature of the crystal structure, allowing much higher frequency stability over external temperature variations at the cost of an active heating element.

2.1.2 Novel Approach

The basic idea of Differential Drift (see Section 2.2), i.e., exploiting two components to compensate its frequency drift for each other, isn't entirely new. In [Sch] Schodowski introduces a temperature sensing device using a dual-harmonic-mode crystal (SC cut crystal). The two harmonics of the SC cut crystal have different temperature behavior. Mixing these two frequencies results in a beat frequency that is proportional to the temperature. Subsequently, Bloch et al. [BMH89] develop the Microcontroller Compensated Crystal Oscillator (MCXO) based on said SC cut crystal. This MCXO achieves the precision of an Oven Controlled Crystal Oscillator (OCXO) although it consumes only a fraction of its power ($\sim 70\text{mW}$ instead of $\sim 1.5\text{W}$) since it doesn't need an active heating element.

Measuring the temperature using two AT-cut crystals, which is similar to our basic approach of measuring drift, has been done by Satou et al. [SH93]. Nevertheless, all the prior work focuses on ways to generate a stable frequency for the purpose of building accurate oscillators. This research on the other hand does not begin with the sole interest

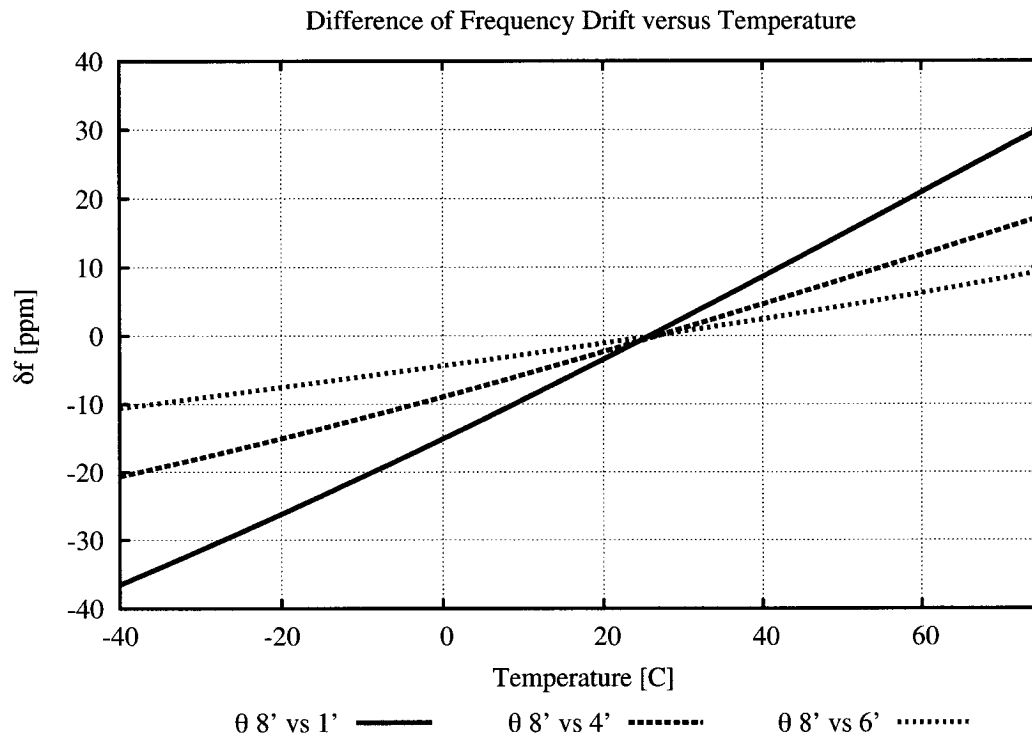


Figure 2.2: Differential Frequency Drift vs Temperature for multiple pairs of differently AT-cut oscillators. Note that the steeper the slope, the better it is for our compensation algorithm.

of building a low drift oscillator. Instead, we sought to find a way to provide accurate time when an application requests it.

The contribution of this chapter are the low-power timer algorithms for wireless sensor network applications, and a full working prototype that achieves a $5\times$ better energy performance compared to commercially available systems, with the potential of higher gains in future hardware iterations.

2.2 Introducing Differential Drift

In order to explain how the software based compensation technique works, we first describe the mechanism intuitively. Assume that the system under consideration has two AT-cut quartz crystal oscillators with slightly different shearing angles. For now, we pick the crystals with the top and bottom curves from Figure 1.4 representing the $35^{\circ}21'$ cut and the $35^{\circ}28'$ cut. Measuring the “difference of drift” between the two oscillators (or differential drift) over the entire temperature range and plotting it against temperature, we obtain Figure 2.2. The reason that Figure 2.2 is almost a straight line is due to the fact that between the two crystals, it is the B parameter from Equation 1.2 that dominates. Now, if instead we plot the frequency drift of one of the oscillators against the differential drift, we obtain Figure 2.3, which is similar to the $f - T$ curve of that oscillator. This leads us to believe that if the system measures the differential drift at run time, it can estimate the relative drift of one of its oscillators. Using this information, the system can make a correction to its oscillator when it deviates and thus gain higher frequency stability.

This approach is analogous to traditional temperature compensation techniques except that there are two significant advantages to compensating using differential drift. On the one hand, temperature sensing is itself error-prone, requiring its own calibration and compensation system to provide an appreciable accuracy in the reading. Further, temperature sensing displays non-linear dynamic behavior causing hysteresis effects during temperature variations. On the other hand, the measurement of differential drift is completely done in digital logic and software, and the accuracy of measurement can be set arbitrarily high. There are no dynamic behaviors that affect the reading and the speed of acquiring a reading scales with the speeds of deep sub-micron process technology. Additionally, the potential saving in hardware and thus production cost is tremendous since the logic circuitry could directly be integrated into microcontrollers or

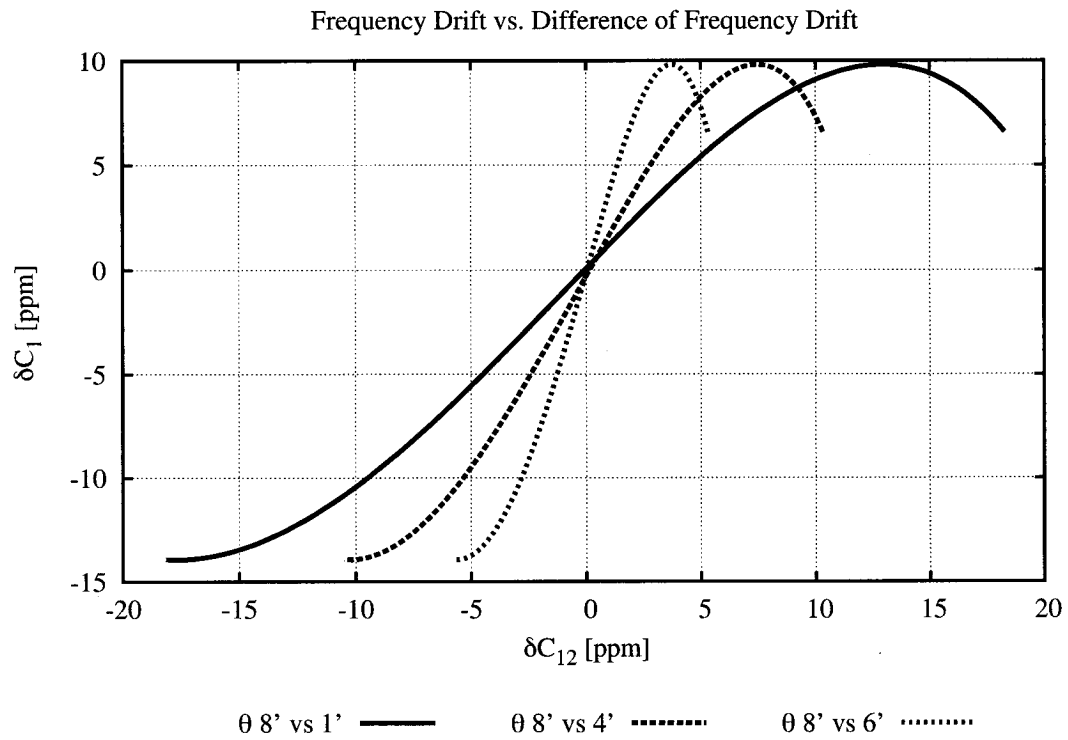


Figure 2.3: Frequency Drift versus Difference of Frequency Drift for several pairs of AT-cut oscillators. The larger the full span of δf_{12} the better our algorithm can compensate.

systems on a chip (SoC) at virtually no additional cost. To put it briefly, using simpler hardware with smarter algorithms can benefit in lower production cost and performance advantages, as will be show later on.

2.3 The Crystal Compensated Crystal-Based Timer (XCXT)

The crystal compensated crystal-based timer uses measurements of differential drift introduced in Section 2.2 to compensate for the frequency deviation of the clock source. The following illustrates how this is done. Let the frequency of the two oscillators in the system be denoted as f_1 and f_2 . These frequencies deviate due to various factors from the nominal oscillation frequency of F_0 Hz. The relative frequency drift for each oscillator is given by Equation 1.1. The differential drift, defined in Section 2.2 and denoted as δf_{12} , is given by $\delta f_{12} = \delta f_1 - \delta f_2$. To measure the differential drift, observe that δf_{12} can be simplified using Equation 1.1 to:

$$\delta f_{12} = \frac{f_1 - F_0}{F_0} - \frac{f_2 - F_0}{F_0} = \frac{f_1 - f_2}{F_0} \quad (2.1)$$

Thus, measuring differential drift could be implemented by measuring the difference in frequencies between the oscillators. Measuring frequencies in software is achieved by counting the number of clock pulses within some fixed time interval and dividing by that interval. The interval must be large enough so that the $1/f$ temporal quantization error does not affect the accuracy of the measurement. On the other hand, the interval must remain small to ensure short measurement acquisition times and quick response to dynamic environments. The sampling interval could be derived from a faster clock source in the system by generating a low frequency sampling signal F_s .

The frequency correction algorithm is executed in two parts, a one time calibration phase and a run time compensation phase. The simplest implementation of the algorithm requires the software to have low level access to a hardware timer and two hardware

counters. However, it is very likely that this restriction can be relaxed to one counter in future implementations.

2.3.1 Calibration Phase

In the calibration phase, the system develops an equivalent of the δf_{12} vs. δf_1 characteristics by measurement against a known reference of the sampling clock, F_s . The calibration phase is performed for a pair of oscillators at the factory. This is analogous to the calibration phase performed for TCXOs described earlier, except that the calibration reference required is simply a stable externally applied sampling clock, F_s . The requirement to accurately measure temperatures is eliminated. The calibration phase requires the use of two free running hardware counters C_1 and C_2 that are being fed from the two oscillators with frequencies f_1 and f_2 respectively. The system then captures the values of both counters at the positive edge of F_s via an interrupt service routine and resets them to zero for the next sample. The collection of counter samples is performed over the largest temperature variation possible, ideally over the entire range from -40 to $+85^\circ\text{C}$. Let the collection of counter samples, sampled at F_s , be denoted as C_1 and C_2 .

Define two new quantities δC_i and δC_{ij} as follows:

$$\delta C_i = \frac{C_i}{F_0/F_s} - 1 \quad (2.2)$$

$$\delta C_{ij} = \delta C_i - \delta C_j \quad (2.3)$$

Before the sampling routine exits, the routine computes δC_1 , δC_2 and δC_{12} and publishes it for subsequent storage. This procedure is shown in the pseudo-code in Procedure 2.1.

It will now be shown that $\delta f_1 = \delta C_1$, $\delta f_2 = \delta C_2$ and $\delta f_{12} = \delta C_{12}$, so that a model of δC_{12} vs. δC_1 is equivalent to a model of differential drift δf_{12} versus frequency drift δf_1 .

Procedure 2.1 Calibration:On_Fs_Interrupt

```
C1 ← Counter1.value
C2 ← Counter2.value
Counter1.reset()
Counter2.reset()
dC1 ← [C1 / (F0/Fs)] - 1
dC2 ← [C2 / (F0/Fs)] - 1
dC12 ← dC1 - dC2
print dC1, dC2, dC12
```

The value of the counter in each interval can be given by:

$$C_i = \frac{f_i}{F_s} \quad (2.4)$$

where f_i is the *mean* frequency of the oscillator over the sampling interval. δf_i is given by Equation 1.1 as:

$$\delta f_i = \frac{f_i - F_0}{F_0} = \frac{f_i}{F_0} - 1 \quad (2.5)$$

$$= \frac{C_i \cdot F_s}{F_0} - 1 = \frac{C_i}{F_0/F_s} - 1 \quad (2.6)$$

$$= \delta C_i \quad (2.7)$$

A similar relationship can be shown for δf_{12} and δC_{12} .

At the end of the calibration phase, the system has collected a set of $\langle \delta C_{12}, \delta C_1 \rangle$ and $\langle \delta C_{12}, \delta C_2 \rangle$ tuples. We found that these tuples fit well to a third order polynomial function because C_{12} is linearly proportional to the temperature (see Figure 2.2). Thus, only the values of A , B , C , and D in the following equation are required to be stored:

$$\delta C_1 = A \cdot (\delta C_{12})^3 + B \cdot (\delta C_{12})^2 + C \cdot \delta C_{12} + D \quad (2.8)$$

The entire set of tuples can also be stored as a lookup table, subject to memory availability. Section 2.3.5 describes how this choice affects the performance of the compensation

algorithm.

2.3.2 Compensation Phase

The compensation phase performs a continuous frequency correction at run time to provide a higher stability clock. Instead of stabilizing f_1 or f_2 using hardware based tuning circuits, we focus on regenerating a replica of the stable sampling clock, F_s from the unstable clock sources. It can be shown that the fact that F_s is at fairly low frequency does not affect the accuracy of the clock as long as it captures dynamic variations in the environment adequately.

The key idea of the compensation algorithm is to estimate the frequency of the sampling clock as closely as possible. Thus, at every pulse of this generated clock, we know that (close to) $1/F_s$ seconds of *real time* has elapsed. This value is accumulated in a register that then provides the real time at the pulse edge. To timestamp an event or read time between two pulses, an intermediary correction is required as described in Section 2.3.3.

If the timer is fed the clock source f_1 and loaded with a value γ , the timer expires after a time interval γ/f_1 seconds asserting an interrupt. If the timer reloads itself on expiry with the same value γ , it generates a periodic sampling signal with frequency denoted as f_γ . Again, two counters C_1 and C_2 are used, fed by the two oscillators running at frequencies f_1 and f_2 respectively.

As shown in the initialization routine in Procedure 2.2, the timer is loaded with a value $\gamma = F_0/F_s$ since this provides the best initial estimate of the sampling signal. On every timer interrupt, a sampling and compensation routine is executed as shown in Procedure 2.3.

First, the values of the two counters are captured and the counters reset to zero for

Procedure 2.2 Compensation: Initialization

$\gamma \leftarrow F_0/F_s$
RealTime $\leftarrow 0$
Timer1.value $\leftarrow \gamma$
Timer1.reset()
Counter1.reset()
Counter2.reset()

Procedure 2.3 Compensation: On_Timer Interrupt

//Read and reset counters
C1' \leftarrow Counter1.value
Counter1.reset()
C2' \leftarrow Counter2.value
Counter2.reset()
// Calculate the normalized difference
dC12' $\leftarrow (\gamma - C2') / (F_0/F_s)$
//Calculate the correction term
dC1' $\leftarrow A \cdot (dC12')^3 + B \cdot (dC12')^2 + C \cdot dC12' + D$
 $\gamma \leftarrow (F_0/F_s) \cdot (1+dC1')$
//Increment the corrected counter
RealTime \leftarrow RealTime + $1/F_s$
//Update the timer
Timer1.value $\leftarrow \gamma$

the next sample. Note that since *Counter1* is incremented with the same clock source as the timer, when the interrupt is asserted, *Counter1* would have incremented γ times, i.e., $C'_1 = \gamma$. Then, an estimate of δC_{12} is computed using Equation 2.3 as follows:

$$\delta C'_{12} = \frac{\gamma - C'_2}{F_0/F_s} \quad (2.9)$$

This is only an estimate of the original δC_{12} since f_γ is not necessarily equal to F_s . Note that since f_γ is generated using f_1 :

$$f_\gamma = \frac{f_1}{\gamma} \quad (2.10)$$

and $C'_1 = \gamma$. Using the values of A , B , C and D from the calibration phase, $\delta C'_1$ is computed as:

$$\delta C'_1 = A \cdot (\delta C'_{12})^3 + B \cdot (\delta C'_{12})^2 + C \cdot \delta C'_{12} + D \quad (2.11)$$

$\delta C'_1$ represents an estimate of the mean relative drift of f_1 with respect to F_0 over the sampling period. Compensating for this drift would require retuning the oscillator such that f_1 is reduced by an amount $-F_0 \cdot \delta C'_1$. Instead, we compensate by continuously adjusting the value of γ such that $f_\gamma \approx F_s$ as follows:

$$\gamma = (1 + \delta C'_1) \frac{F_0}{F_s} \quad (2.12)$$

γ is then loaded into the timer for the next sampling period. By adjusting the value of γ in this way, we create as close a replica of the reference sampling signal as possible, accumulating *real time* epochs in the *RealTime* register as described below.

2.3.3 Corrected Timestamping

In order to use the above technique to estimate real time, a memory register *RealTime* is used to keep track of the accumulated counts. At every sampling interrupt, this register is incremented by a fixed value $1/F_s$ as shown in Procedure 2.3. An accurate

estimate of the real-time t is then given by reading $RealTime$ and C_1 and applying an inter-sampling period correction.

$$t = RealTime + \frac{C_1(t)}{\gamma \cdot F_s}$$

where $C_1(t)$ is the captured value of the counter at the instant the timestamp is required. This process is described in Procedure 2.4.

Procedure 2.4 Compensation:On_Get_Time

// Return RealTime's value and add the corrected

// elapsed time since the last increment

Correction = Counter1.value · (1/ F_s) · (1/gamma)

return RealTime + Correction

2.3.4 Performance Evaluation

To evaluate the effects of implementation choices on performance, four oscillator architectures are considered: an uncompensated crystal, a commercial TCXO, the XCXT implementing the cubic fit Equation 2.8, and the XCXT implemented via a lookup table. Due to the intrinsically low level nature of the compensation technique, we found it illustrative to evaluate its performance through simulation. The simulator models the drift characteristics of the oscillators and estimates their performance based on the implementation of the algorithms described previously. Figure 2.4 illustrates the $f - T$ characteristics for an oscillator with various schemes of compensation. The $f - T$ characteristics of the TCXO [Max08] is also plotted to serve as a reference. It is observed that the stability of the XCXT is significantly better than the uncompensated version but shows large variations around the $0ppm$ line – an effect examined below. Note that these characteristics correspond to $F_0 = 1 MHz$ and $F_s = 2 Hz$.

The performance of a particular compensation technique is quantified in *com-*

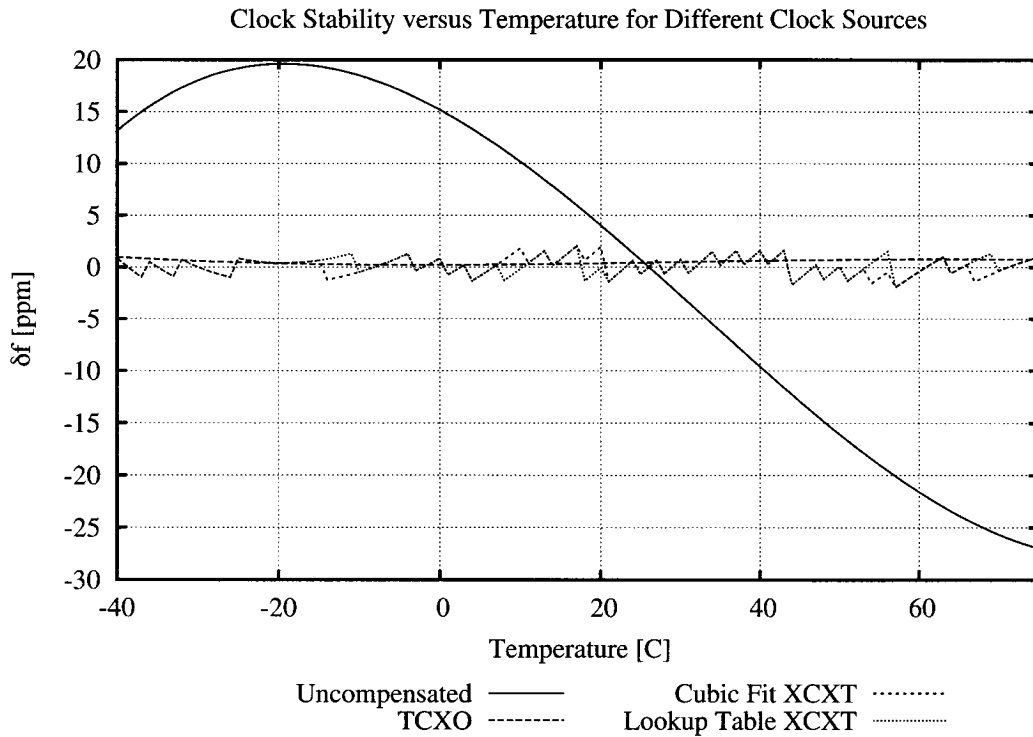


Figure 2.4: Frequency Drift versus Temperature characteristics for Uncompensated, TCXO, Cubic Fit XCXT and Lookup Table XCXT

compensation gain, defined as the ratio of the frequency drift of the oscillator without compensation to the drift with compensation at a specific temperature. A log plot of this compensation gain over temperature for the TCXO and XCXT is shown in Figure 2.5. The plot reveals that even though there is large variation between the compensation schemes, the mean value of gain over all temperatures for both the TCXO and XCXT is about the same, between 14 and 14.5 dB. The dip in the TCXO curve is a surprising artifact. We attribute this to an uncharacteristically high stability in the uncompensated oscillator at 25°C at which the TCXO seems to have a slightly higher frequency drift.

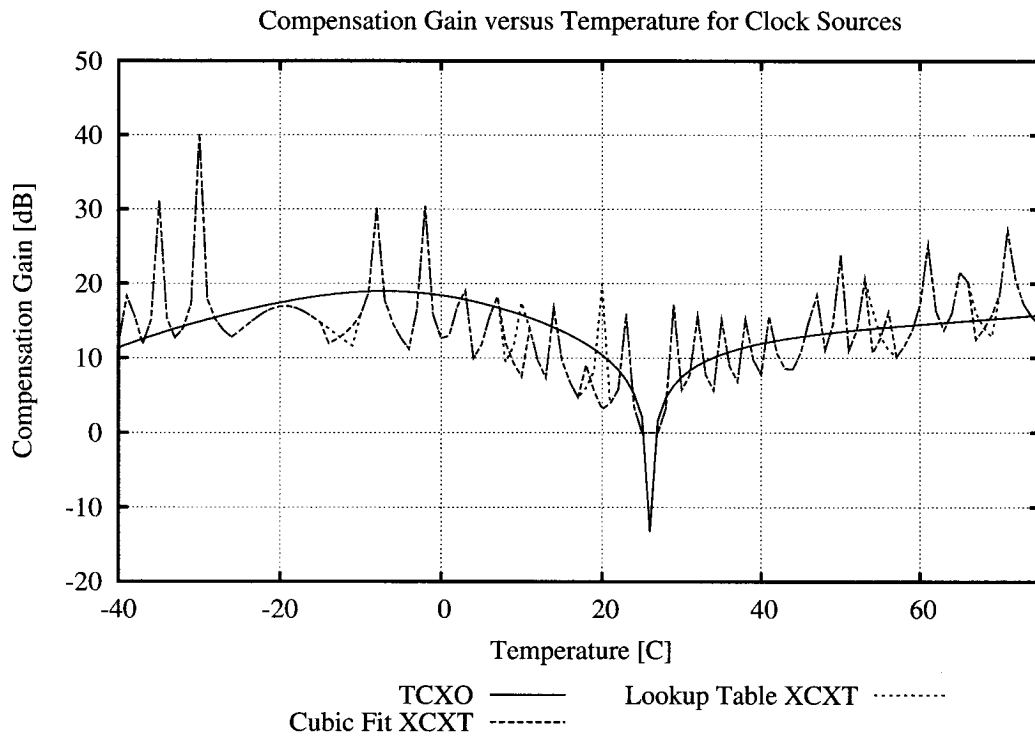


Figure 2.5: Frequency Drift versus Temperature characteristics for Uncompensated, Temperature Compensated, Cubic Fit based Compensation and Lookup Table based Compensation

2.3.5 Model Fitting Effects

During the calibration phase described in Section 2.3.1, the system collects a set of tuples given by $\langle \delta C_{12}, \delta C_1 \rangle$ and $\langle \delta C_{12}, \delta C_2 \rangle$. The cubic fit XCXT employs a third order polynomial fit to the tuples to form the relation given by Equation 2.8. It is found that the cubic fit fares quite well with root mean square error consistently below $10^{-1} ppm$.

Alternatively, the XCXT lookup table implementation compiles the calibration data in a one-to-one mapping between δC_{12} and δC_i . Since the values of δC_{12} are quantized and we have a calibration data point for each possible value of δC_{12} stored in the table, no interpolation is required.

A comparison of the cubic fit XCXT and the lookup table XCXT is shown in Figure 2.4, which illustrates the $f - T$ characteristics for $F_0 = 1 MHz$ and $F_s = 2 Hz$. It is observed that the lookup table characteristics generally follow that of the cubic fit. For this set of oscillator characteristics and choice of F_0/F_s , the lookup table size is only 37 entries which covers all possible values of δC_{12} .

Figure 2.5 depicts the difference in compensation gain, as defined in Section 2.3.4. As the δC_{12} and δC_i values are themselves quantized and the calibration data points cover all values, the mean compensation gain for the lookup table implementation is slightly higher than that of the cubic fit.

2.3.6 Quantization Effects

Due to the discrete-time nature of the compensation algorithm, quantization errors are introduced in the measurement of δC_{12} and in the adjustment of γ . Recall from Equation 2.3 that δC_{12} is computed from the captured values of the counters. In each sampling period, each counter increments by a nominal value of F_0/F_s . Therefore,

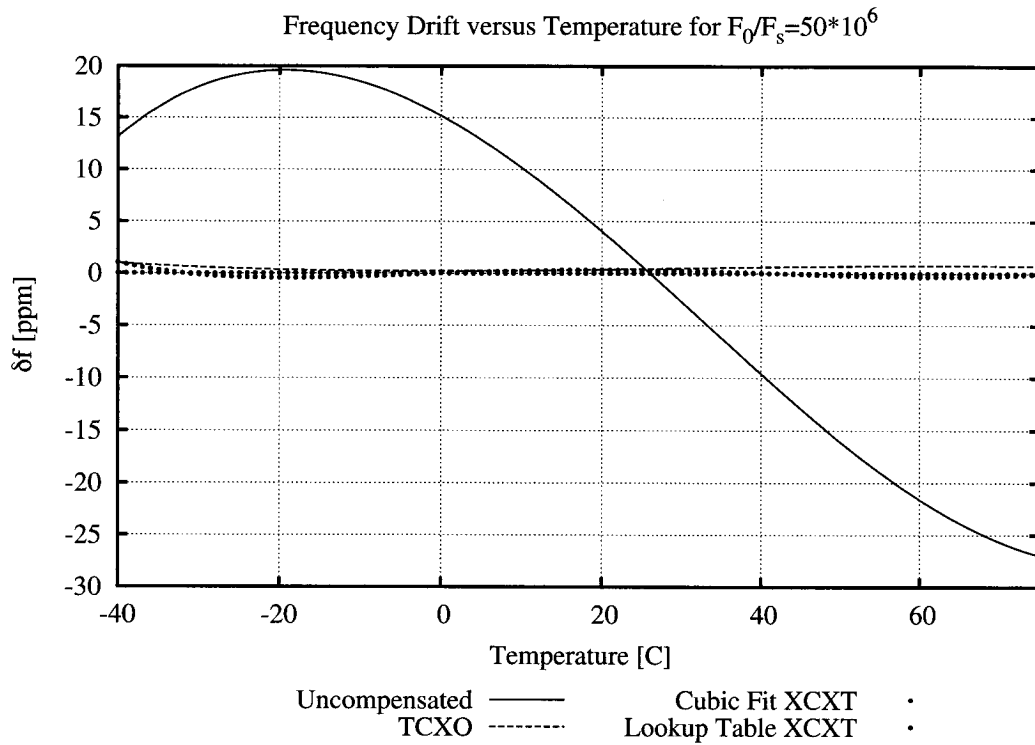


Figure 2.6: Frequency Drift versus Temperature characteristics for Clock Source 2 showing minimal effects due to quantization when $F_0/F_s = 50 \times 10^6$

the maximum error in the measurement of δC_{12} due to both counters is $2F_s/F_0$. This relation shows that if the F_0 is made higher or F_s is lowered, the performance of the compensation scheme would improve.

Further, during the compensation phase the value of γ that is computed is approximated to the closest integer value in order to adjust the counter that generates f_γ . Due to this quantization, an error is introduced in the value of f_γ . This in turn affects the accuracy with which the measurement of δC_{12} is taken, since the sampling period in the compensation phase should match $1/F_s$ as closely as possible. It can be shown that the error in f_γ due to this is approximated by F_s^2/F_0 . This relation further supports the intuition that if the nominal frequency was increased or the sampling interval was

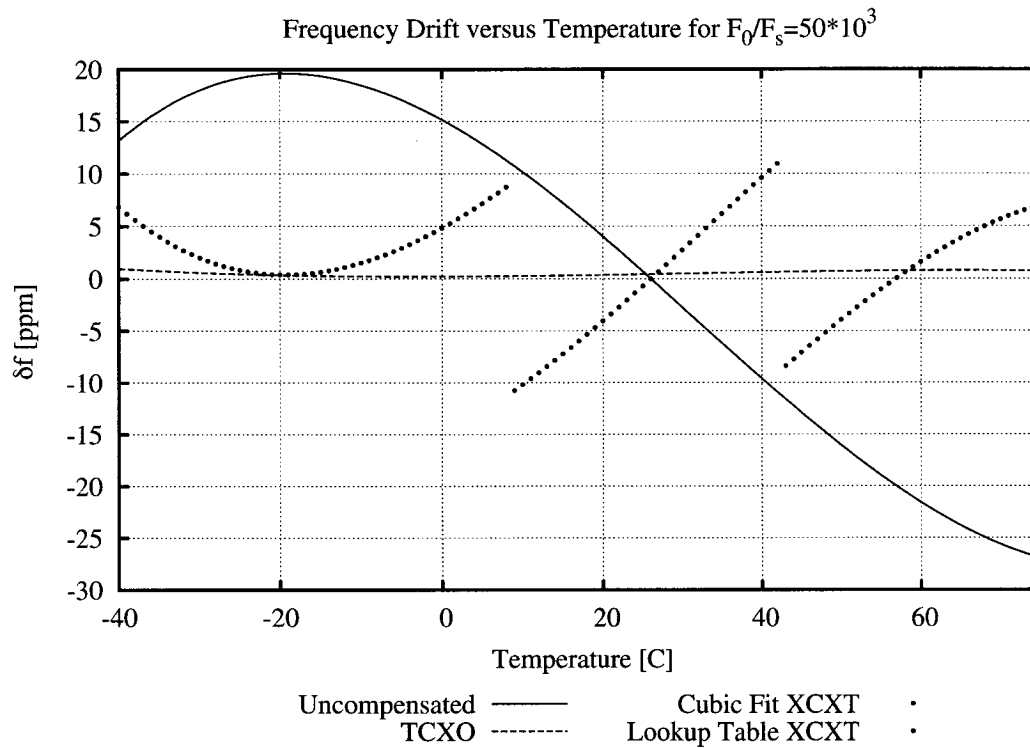


Figure 2.7: Frequency Drift versus Temperature characteristics for Clock Source 2 showing detrimental effects of quantization when $F_0/F_s = 50 \times 10^3$

F_0/F_s	Cubic Fit Gain (dB)		Lookup Table Gain (dB)	
	μ	σ	μ	σ
$5K$	0	0	0	0
$50K$	6.1	5.4	6.1	5.4
$500K$	14.3	6.3	14.5	6.1
$5M$	18.5	6.1	25.8	6.6
$500M$	18.5	6.8	45.2	6.0

Table 2.1: Quantization Effects on Compensation Gain

increased, it would lead to better stability. This is verified through simulation. Figure 2.6 illustrates the performance when F_0/F_s is set to 50×10^6 and Figure 2.7 depicts the case when F_0/F_s is 50×10^3 . Note that the dynamics of temperature variation pose a lower limit on the value F_s as it must be ensured that F_s is high enough that temperature variations are not missed. Consequently, the choice of $F_s = 2 \text{ Hz}$ in our evaluation (described in Section 2.4) is conservative from a performance perspective.

2.3.7 Other Effects

Apart from the effect of choices made within the compensation technique described above, numerous other areas have been identified that could lead to reduced performance if not handled properly. With the exception of certain latency issues covered in Section 2.4.3, the detailed effects of these issues will be considered in subsequent research. Firstly, an error is introduced in the measurement of δC_{12} when $f_\gamma \neq F_s$. This affects to which part of the $\delta f_{12} - \delta f_1$ curve δC_{12} is being mapped. Secondly, when the span of δf_{12} is small, as in $\theta = 6', 8'$ curve in Figure 2.3, the effects of quantization error are more pronounced.

The third set of errors accrue from the fact that executing the compensation algorithm

in software results in latencies and some indeterministic jitter due to interrupt handling. The following latencies all affect the accuracy of the compensated clock:

1. Computational latency in executing the algorithm, which sets an upper bound on the value of F_0/F_s
2. Latency in capturing counters C_1 , C_2 and also latency in adjusting the value of γ in the timer
3. Latency in resetting counters for the next sample
4. Time difference in capturing and resetting counters (i.e., time difference between first four steps in Procedure 2.3) since they cannot be done concurrently in software (effect is especially large if the processor clock is lower than F_0)

Finally, the algorithm assumes floating point computation, and only considers temporal quantization errors. However, if a floating point unit is unavailable and fixed point computation is required due to resource constraints, additional quantization error will be introduced.

2.4 Algorithm Evaluation

A testbed simulates the oscillators behavior under different environmental impacts, in order to evaluate the algorithm on real hardware. The goal of the testbed is to emulate different oscillator characteristics without the need of rewiring different types of oscillators for every test, and to remove the need of a temperature chamber, that would be needed in order to change the oscillators drifts. Furthermore, the testbed allows us to simulate other impacts on clock drift, such as change in voltage source, change of oscillator speed itself, etc.

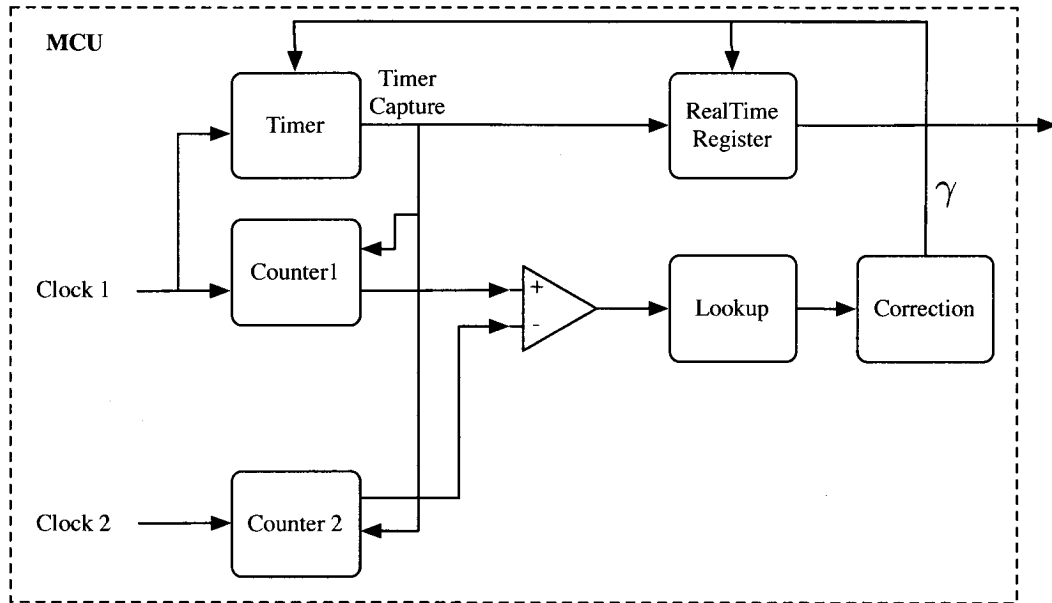


Figure 2.8: Block diagram of the implementation of the XCXT.

2.4.1 Environment Emulator

Two Agilent 33220A arbitrary waveform generators simulate the oscillators to emulate any environment temperature by remotely programming them with a predefined frequency. Figure 1.4 depicts the frequency drift for different AT-cut crystals. Using these graphs and choosing two different cuts, one can set the two waveform generators to two individual frequencies, and thus successfully emulate the impact of a specific environment temperature on the crystals themselves.

2.4.2 Testbed

Figure 2.8 illustrates a simplified block diagram of the testbed hardware implementation. The two Agilent waveform generators are connected to the two clock inputs, and emulate the oscillators themselves. The waveform generators are factory calibrated, and have

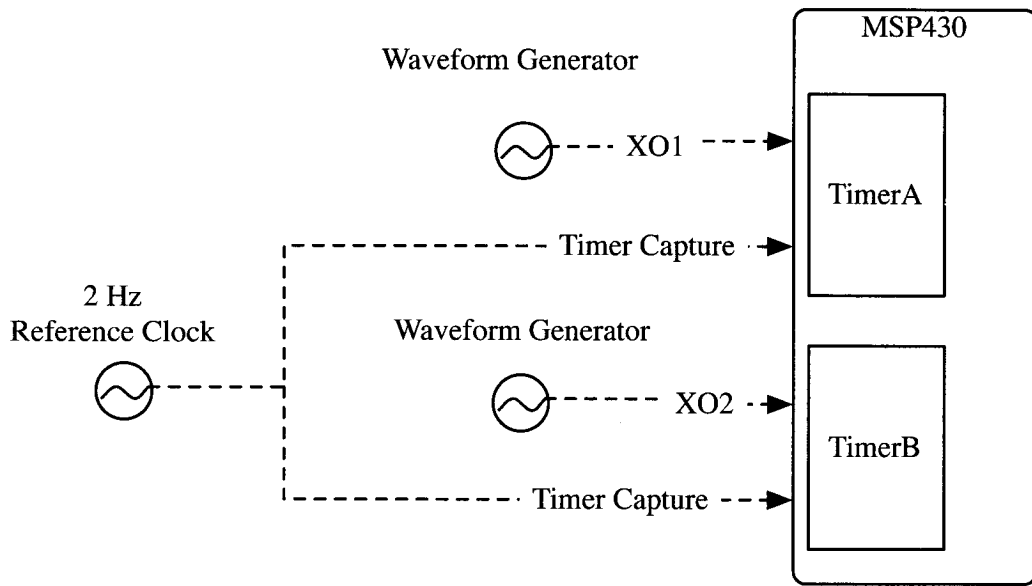


Figure 2.9: Experimental setup with the two Agilent waveform generators, the 2Hz reference clock, and the MSP430 microcontroller with its two timer units.

a frequency accuracy of 10ppm. They can produce a square wave with a RMS jitter of $1ns + 100ppm$ of the period. This is no concern for us since we are running the frequency generators at approximately $1MHz$. Therefore, the RMS jitter in the period is in the order of $1.1ns$. For the timer hardware, we use the MSP430 microcontroller that gives us easy to use 16-bit timers, and has enough processing power to do the compensation phase on the fly. Figure 2.9 illustrates the setup. The waveform generator sends its output to the oscillator input of the MSP430. The external reference clock connects to the two timer capture pins of the MSP430's TimerA and TimerB input captures. These will be used to calibrate the oscillators. TimerA's output connects to a second input capture of TimerB. This will be used for the compensation phase.

2.4.3 Interrupt Latency Considerations

Interrupts on microcontrollers have the tendency to be serviced with non constant latency. For that reason, we have to make sure that interrupt latency and interrupt jitter don't affect our measurements in our hardware implementation. Therefore, we make extensive use of the MSP430's Timer features. Two key features are the timer capture and the timer output. The timer capture allows us to trigger the reading of the timer value by an external signal on an input pin. When the input pin transitions from low to high, the value of the timer is stored in a temporary register and an interrupt is sent to the microcontroller. The application can then read that register and gets an accurate reading of when the signal transitioned, regardless of how much latency the microcontroller takes to service the interrupt.

Similar to the timer capture, the timer output allows an application to toggle an output pin at a specific time. The application writes the time it wants the pin to be toggled into a predefined register. When the timer reaches that value, it will automatically toggle the output pin and send an interrupt back to the application. Both techniques, timer capture, and timer output, are necessary to achieve a high accuracy in the calibration, as well as compensation phase.

2.4.4 Limitations

Our testbed has one particular limitation. The MSP430's Timers are 16-bit units, and thus one has to take care of timer overflows. Additionally, the input frequency to the timers, F_0 , should be chosen such that overflows don't happen too often or the microcontroller will be overloaded with interrupts. At the same time, as discussed in Section 2.3.6, we want F_0 to be as high as possible to achieve a higher precision. We opted to set F_0 to 1MHz. This will generate a timer overflow interrupt every $65.5ms$,

enough time for the microcontroller to treat the interrupt and do some calculations on the data. Additionally, we chose $F_s = 2\text{Hz}$ which is slow enough to measure significant difference between the two clocks, and at the same time not too slow to make data point collection for the experiments unfeasible.

2.4.5 Calibration

We let the two frequency generators run through the whole temperature range from -40° to 75° Celsius and generate the according frequencies. At the same time, as previously described in Section 2.3.1, the reference clock triggers the input capture of the two microcontrollers at a rate of 2 Hz. At each event, the two microcontrollers store the values of their timers to generate the calibration graph for δC_{12} vs. δC_1 . Figure 2.10 shows the result of the calibration run. As expected, the measurements show a quantization effect, and follow a cubic curve. This curve will subsequently be used in the compensation to achieve a high accuracy clock source.

2.4.6 Compensation

The setup for the compensation phase slightly changes from the calibration setup. Now, as described in Section 2.3.2, TimerA initiates the sampling of the clocks after γ clock ticks of the local clock source. Figure 2.11 illustrates this. The node initiates the sampling by programming its TimerA unit to toggle the timer output when the timer reaches γ . At that time, Timer B captures the value of its counter. The node can then calculate δC_{12} and from this the new γ .

While correcting for the new γ , the node also keeps track of the global time estimate by incrementing a third counter by F_0/F_s every time it estimates a new γ .

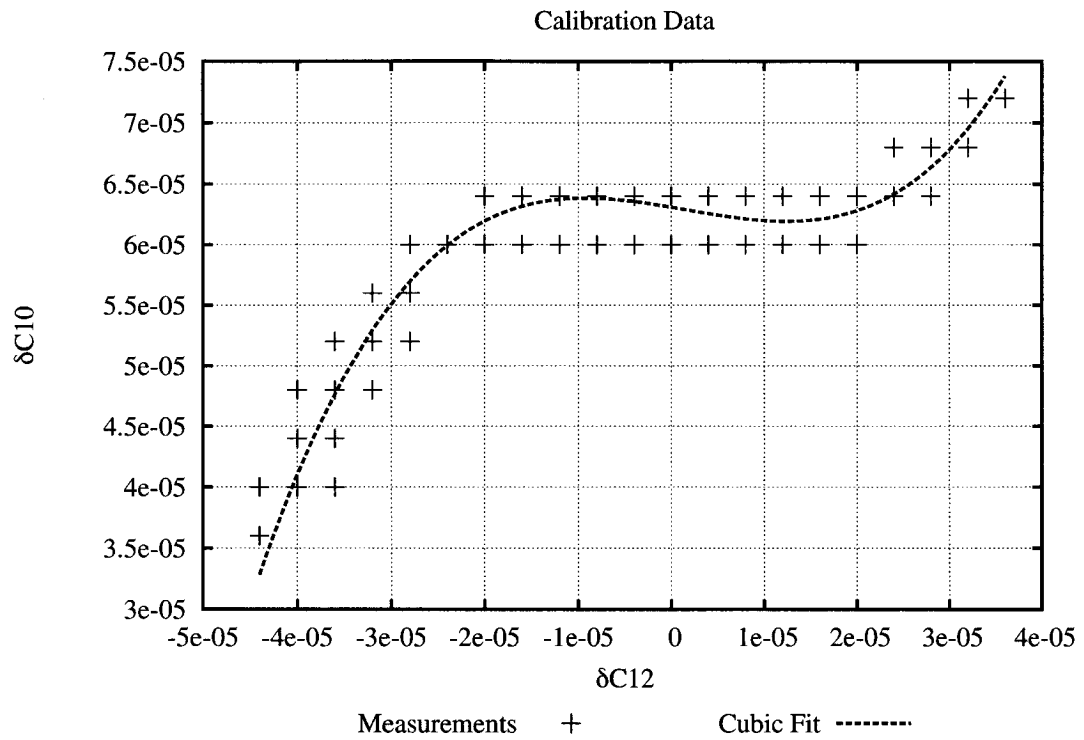


Figure 2.10: Measured calibration and its cubic curve fit. Note the quantization effect which comes from measuring frequency difference with digital counters. The effect on error of this is discussed in Section 2.3.6.

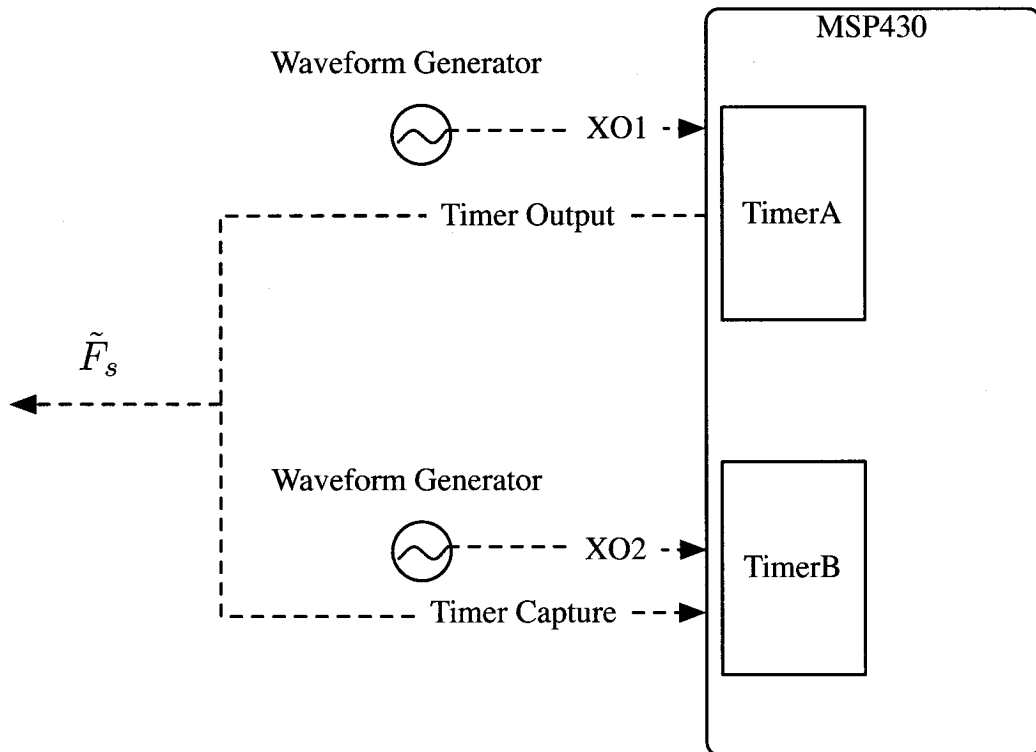


Figure 2.11: Block diagram of our compensation experiment.

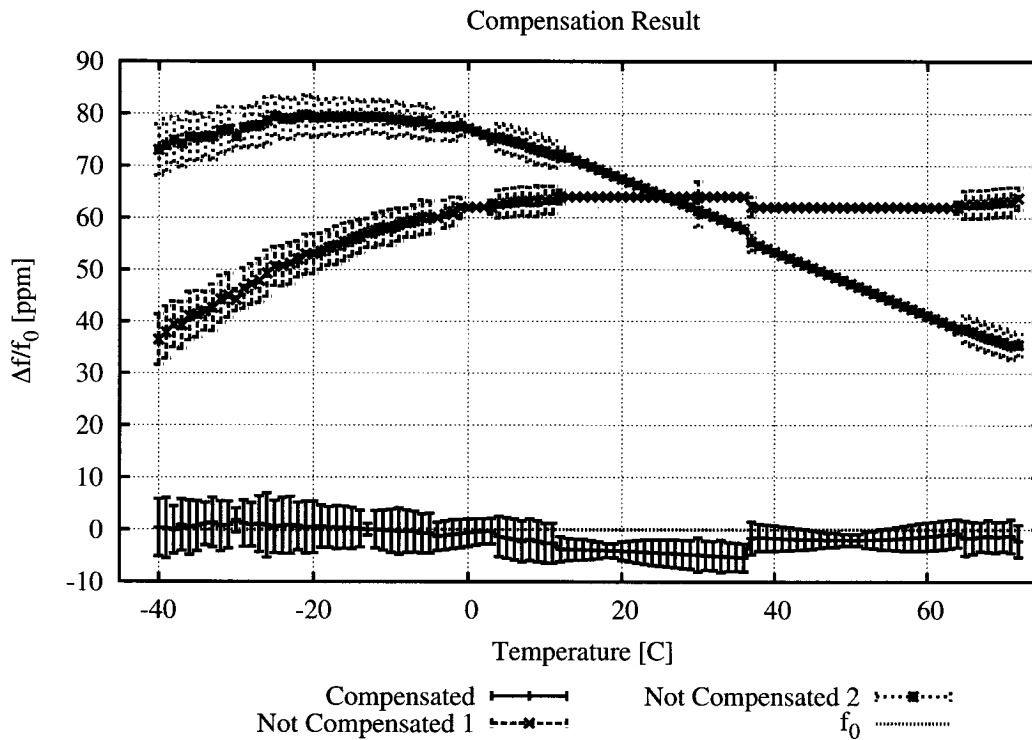


Figure 2.12: Compensated clock drift over the full temperature range of -40° to 75° Celsius. We also show the drift of the two uncompensated crystals used in the compensation algorithm.

2.4.7 Experimental Results

We tested our implementation of the compensation phase by subjecting the two nodes to the frequency drifts happening over the full scale of temperature from -40° to 75° Celsius. We measured how much the estimated γ diverges from F_0/F_s . Figure 2.12 illustrates this further. We can see that our implementation of the compensated clock lies within ± 7 ppm of the real frequency F_0 , 95% of the time. This is expected since the Agilent frequency generator sources we use for the clocks themselves have an inaccuracy of about 10ppm.

Given our algorithm, we expected the XCXT to have a poor short term (order of seconds) stability. This happens because γ is quantized and thus will jump between two values back and forth. If γ is slightly below the value it should be after the fitted compensation curve, δ_{12} will be smaller in the next iteration, and thus the next γ will be a little bit bigger again. Nevertheless, this should even itself out for medium term (order of tens of seconds) stability. A good measure for clock stability is the Allan Variance [AAH97] defined as

$$\sigma_y^2(\tau) = \frac{1}{2} \mathbb{E} [(y_{n+1} - y_n)^2] \quad (2.13)$$

where y_n is the normalized average frequency over the sample interval, and τ is the length of each sample interval. Figure 2.13 depicts the Allan Variance for two different runs of our experiment. From this graph we can see that the compensated clock source indeed does have a worse short term than mid term stability. Over longer terms (order of minutes to hours) the clock stability gets worse again which indicates that there is still some drift over these time ranges. This was to be expected since we went through a huge temperature range. Future long term tests under realistic temperature ranges obtained from real measurements will show how the compensation techniques works under these conditions. At the same time, Figure 2.13 shows that the XCXT trades short term versus long term stability compared to the uncompensated clock source.

A more interesting measurement for application developers is the drift of the local clock in relation to global time. Figure 2.14 shows this difference for two runs of our emulator. Again, we changed the temperature from -40° to 75° Celsius and queried the node in regular intervals for its estimate of the global time. The graph shows the difference between the node's estimate and the real global time. After 2 hours, our compensated clock was 11.5 milliseconds too slow. This represents a mean effective clock stability of 1.6 ppm.

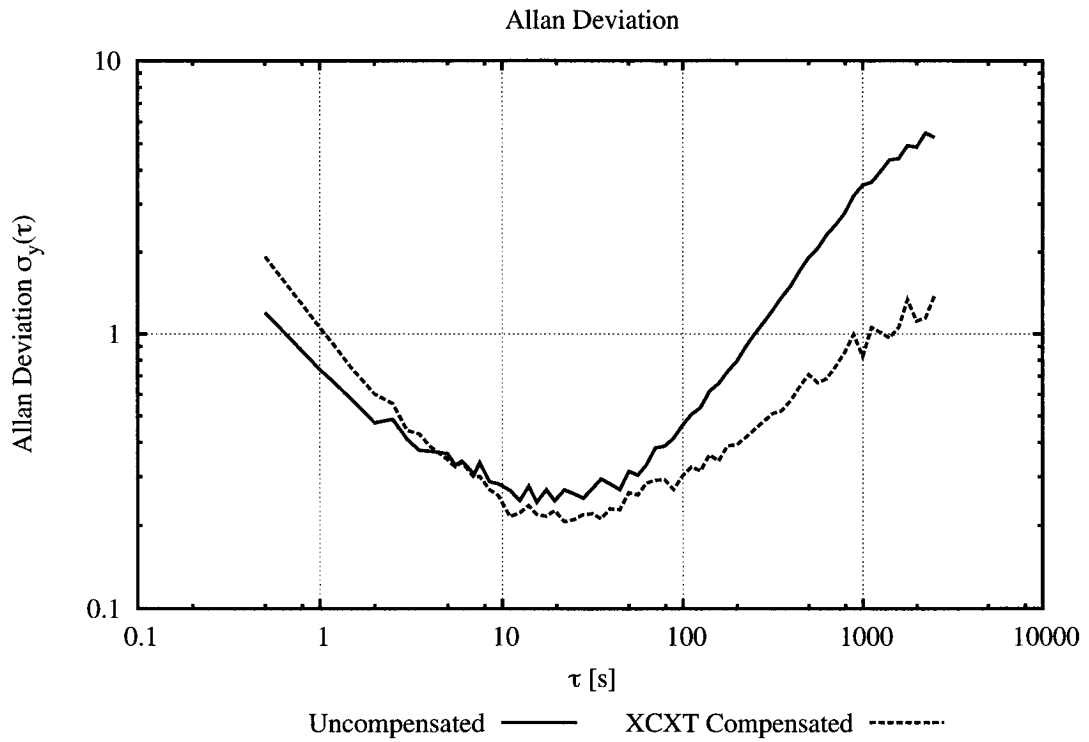


Figure 2.13: Frequency stability expressed by the Allan Variance for the compensated and uncompensated clock. We can see that we trade short term vs. long term stability by employing our compensation algorithm.

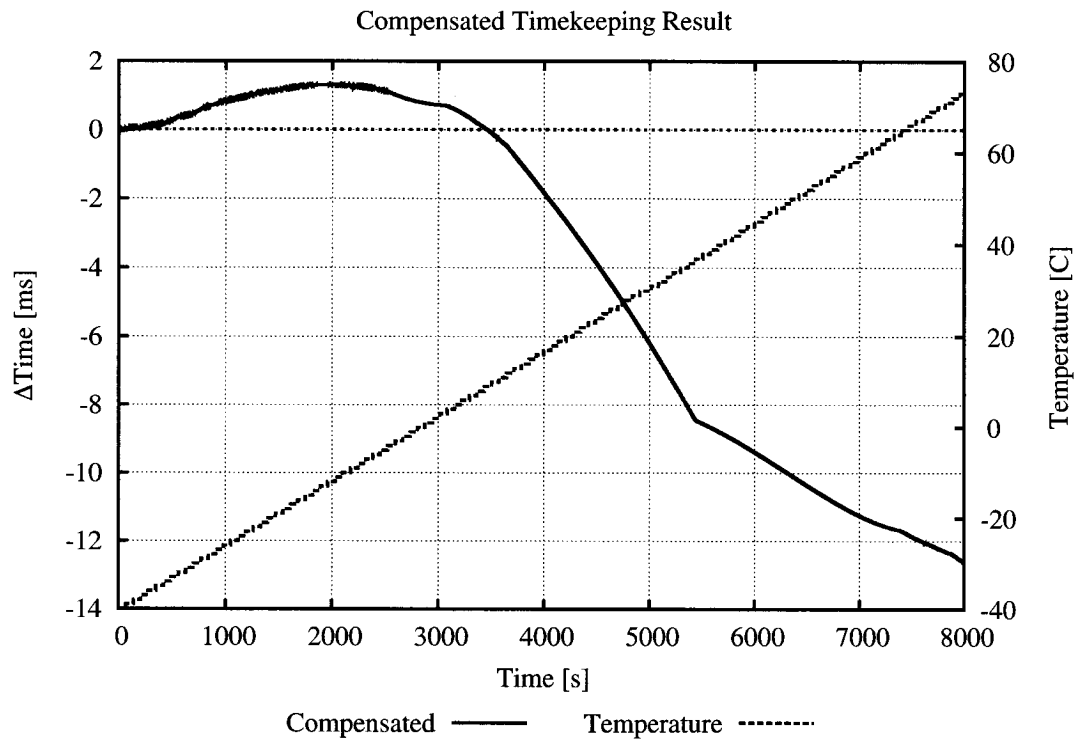


Figure 2.14: Difference between global time and the estimated time at node 1 over 2.5 hour and a change in temperature from -40° to 75° Celsius. The different phases come from the inaccuracy introduced by the frequency generators, which not always set the output frequency to the desired frequency (see Figure 2.12).

2.5 XCXT Hardware Implementation

After successfully simulating and emulating the differential drift algorithm we continued and implemented the whole algorithm on the TMote Sky[Mot], a Texas Instrument MSP430F1611 based sensor network platform. We chose this platform because the MSP430 has two crystal inputs and two timer units, two of the prerequisites to implement our algorithm. However, this necessitates the removal of the 32kHz crystal, in order to make room for a high frequency one.

The implementation is written in C and occupies 3292 bytes. We expect this number to shrink further as we optimize the code. Additionally, the implementation makes heavy use of the two timer units and its input capture and timer output capabilities. This is necessary in order to minimize unpredictable software interrupt latencies that could introduce large errors in timer measurements.

We subjected the hardware implementation to the same environmental conditions as used for the calibration phase, and measured its capability to self compensate. Figure 2.15 shows the compensation results and compares the uncompensated clock to the compensated clock output. The achieved mean stability is 0.47ppm and the standard sample deviation is 0.31ppm over the temperature range of -10°C to 60°C , resulting in an effective frequency stability at a 95% confidence interval of $\pm 1.2\text{ppm}$. This is very close to the simulated theoretic accuracy of 0.14ppm standard deviation that we measured by extracting the drift model of the two 8MHz crystals and using them as an input to our simulation framework.

The hardware implementation showed us that the concept of Differential Drift not only works in simulation or emulation, but that we actually can build such a system in an XCXT and achieve clock stabilities close to what the simulations predicted. It is now of interest to investigate the power consumption of the XCXT since this is not

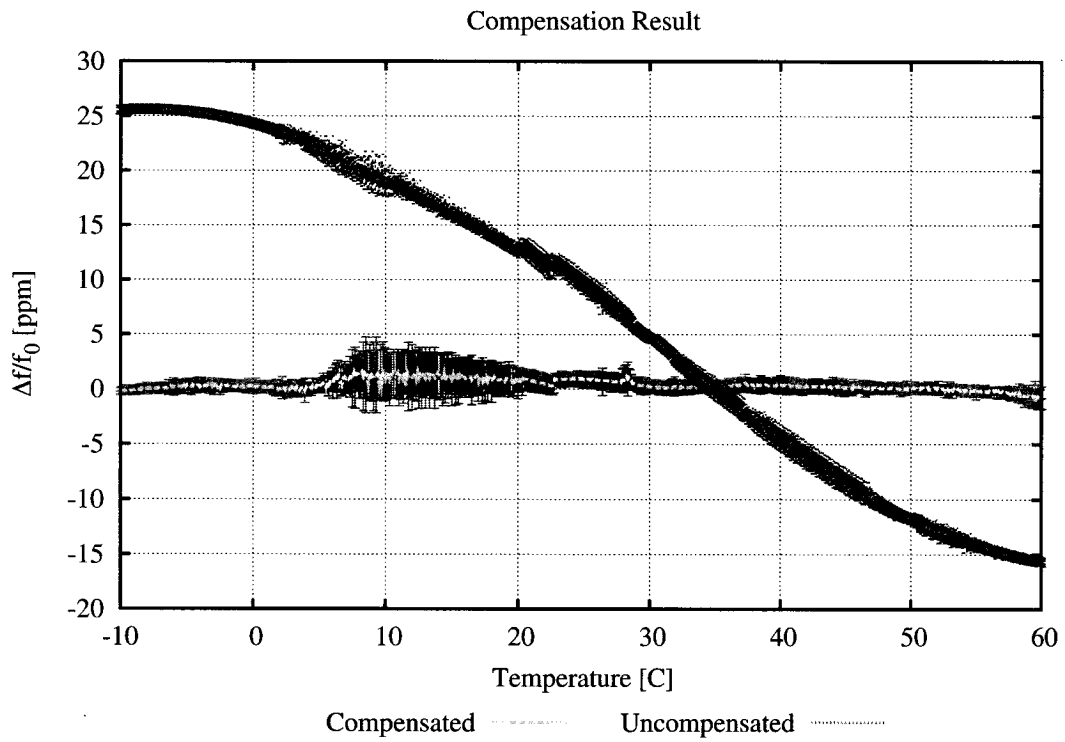


Figure 2.15: Using the LUT from the calibration data, we can successfully compensate one of the crystals for its temperature drift. Over the full temperature range of -10°C to 60°C we measured a standard deviation of 0.31ppm .

something our simulation framework can predict.

2.5.1 XCXT Power Measurements

When we first measured the average power consumption of the XCXT implementation at 1.27mW, we were already impressed since we didn't do any special optimizations to decrease the power consumption. Additionally it is to note that a commercially available TCXO that achieves about the same precision already consumes 5mW[Max08]¹ without the drive and oscillatory circuitry, which are included in our measurement. This led us to investigate which parts of our hardware implementation spend how much energy, in order to optimize our hardware even further.

2.5.2 Microprocessor Power Consumption

The MSP430 has several low power states. In the second lowest one, LPM3, all the peripherals, except one timer unit and one clock signal, are turned off. In that state, the microprocessor consumes as low as $7\mu A$ if it uses a standard 32kHz crystal as its timer input. Unfortunately, we cannot use this low power mode in our XCXT implementation since we need two active timers and two input clock signals in order to measure the drift between the two of them. Thus, the lowest possible mode is LPM1, where the CPU and the internal digitally controlled oscillator (DCO) are turned off, but both timer units are kept on. The microprocessor wakes up from sleep only if one of the timers overflows, or the timer unit interrupts the microprocessor to measure the difference between the two frequencies. At that point, the power consumption sharply rises as depicted in Figure 2.16. However, the computations are small and thus the length of active time is very short. As we will show in the next subsection, the average power

¹The Maxim DS4026 TCXO datasheet also gives numbers for the drive and oscillatory circuit. They are 3mA and 2mA respectively (typical). This gives a grand total of 21.4mW at 3.3V.

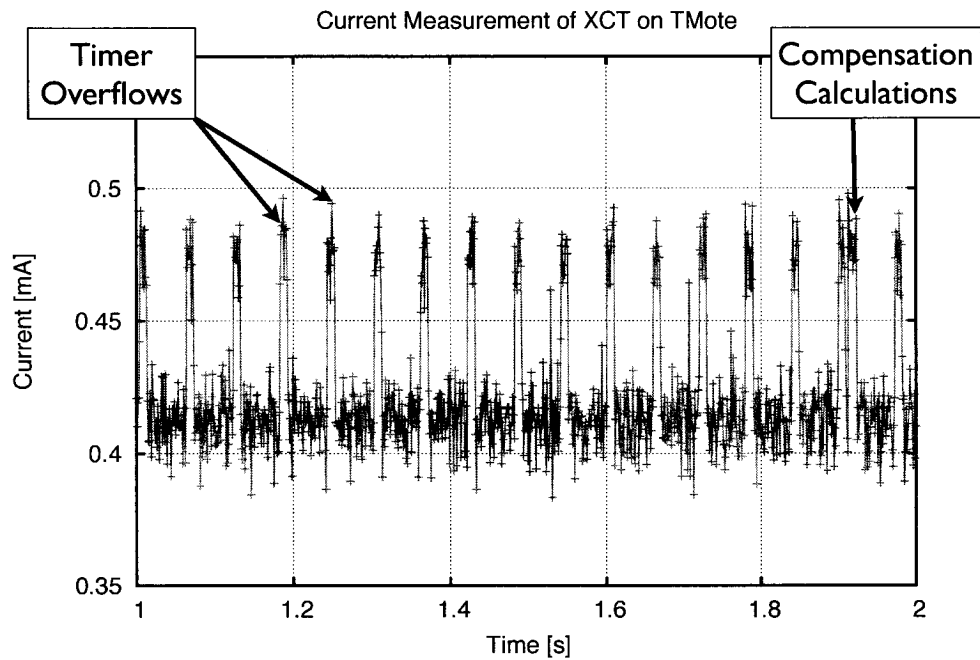


Figure 2.16: This is a typical current consumption plot of the XCXT. The measurements were taken at 3V. The spikes represent instances when the microprocessor woke up because of a timer overflow which has to be treated.

consumption is dominated by the power consumption of the oscillatory circuitry and timer units that need to keep track of the counters. In Section 2.6.3 we will go deeper into techniques and algorithm improvements that exploit this fact. But first, let's look at the power consumption for one oscillator and its timer unit.

2.5.3 Oscillator and Timer Power Consumption

Provided with the power measurements from the last section, we can see that most of the energy of the XCXT is spent while the CPU is off. Thus, it is very important to be able to attribute that energy to the specific components in order to optimize the XCXT power efficiency. In a series of experiments we tested the different sub-components of the microprocessor and the TMote platform. We found that the 8 MHz oscillators and the according clocking and timer subsystem consumes most of the remaining power budget at sleep. In order to get a more detailed view, we conducted several different experiments with different combinations of number of clocks, timing units, clock dividers, and even timer dividers.

To better understand the different settings we present a quick overview of the basic clock system of the used MSP430 platform. The MSP430 has three different clock signals called ACLK, SCLK, and MCLK. Each one of them can be configured to be clocked from different sources, like the crystal 1 (XT1), crystal 2(XT2), or the internal DCO. Additionally, each clock signal can use a divider that divides its clock source by a factor of 1, 2, 4, or 8 respectively. The MCLK is the master clock and drives the CPU. Thus, this signal cannot be used to feed the two timer units. But either ACLK or SCLK can be used as a source for each of the timer units. Additionally, each timer has again a divider that can be set to divide the input signal by 1, 2, 4, or 8. In our configuration, both XT1 and XT2 are a 8MHz crystal. XT1 is the source for the ACLK clock signal, and XT2 for SCLK. Additionally, Timer A is fed by ACLK and Timer B by SCLK. Thus, using the different dividers, different clock speeds can be achieved at the input to the timers, as well as within the timer.

Using the different divider combinations, and by turning on and off the timer units, we were able to measure the power consumption of one oscillator using different clock dividers, as well as the power consumption of the oscillator including a timer unit.

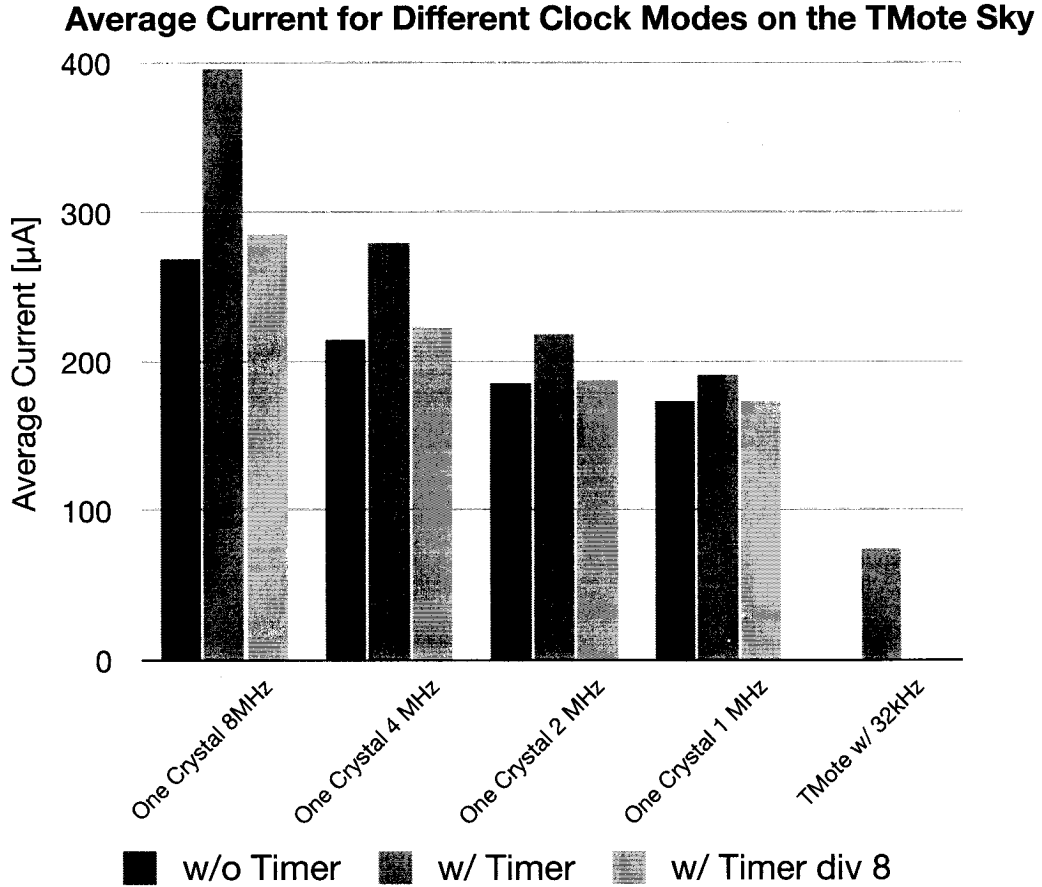


Figure 2.17: These are the typical current consumptions of the clock and timer subsystem of the MSP430. The clock source was an 8MHz crystal, and the different clock speeds were achieved by dividing that signal using an internal clock divider. As comparison, we also measured the MSP430 power consumption when it uses a 32kHz crystal, one timer active, CPU turned off.

Figure 2.17 shows a summary of the result. For each possible clock divider that results in a 8 MHz, 4 MHz, 2 MHz, and 1 MHz clock signal, we made measurements with the timer unit disabled, enabled, and enabled with an additional division by 8. Additionally, we measured the power consumption of the TMote platform when the CPU is at sleep, and one timer unit is clocked from a 32kHz crystal. This will help us to identify the impact to the power budget when using a high frequency crystal, instead of a low frequency one.

From the obtained measurements, we can establish some general and intuitive rules:

1. The higher a clock divider, the less power the clocking subsystem consumes.
2. Enabling a timer unit adds a considerable power overhead at high speeds.
3. Dividing at the timer unit is less efficient than dividing the clock signal ahead of the timer.
4. A high frequency clock system has a considerable impact on the power budget and should thus be avoided if high granularity time is not needed.

Using these simple rules, we can come up with new algorithms and techniques to improve the power efficiency of the XCXT and make it into a Smart Timer Unit. In the next section, we will analyze some of these possibilities and show how they can be implemented on our current prototype. But before that, let's look again at the power consumption we measured for our XCXT prototype. We measured the consumption while running the compensation phase, where the two clock sources run at 1 MHz each, at 1.27mW. From our other measurements we know that one crystal with a divider by 8 and a timer unit consumes 191 μW . Thus, the two crystals with their timers alone consume a total of 1.15mW at 3V. This confirms our suspicion that the crystals, and not our compensation algorithm or the microprocessor, account for the majority of the power consumption of the XCXT.

2.6 Discussion

2.6.1 Crystal Compensated Crystal Oscillator (XCXO)

One major difference between the XCXT and a regular TCXO is that it doesn't provide a corrected crystal, but a corrected counter. A future idea is to use a similar technique as applied in the XCXT and tune the crystals to the correct frequency. This would have the advantage that the compensated crystal could be used directly in digital circuits without counters. Figure 2.18 illustrates a block diagram of such a crystal. It still exploits two crystals and their difference, but instead of adapting the value of a counter, it directly tunes the crystal through a tuning circuit. However, the formulas and exact working of such a crystal compensated crystal oscillator (XCXO) have to be worked out and we leave it for future study.

2.6.2 Cost – Performance Comparison

In this section, we will discuss the expected bill of material (BOM) for an implementation of a XCXT/XCXO, and compare it to similar devices, mainly TCXO's. First, we need to estimate the BOM of a TCXO. [NH68a] mentions that 40-50 % of the cost of a TCXO go into calibration of the device. Thus, the pure BOM of a TCXO, given the data from Figure 1.2 lies between \$2.50-\$14 for an accuracy ranging between 8-1ppm. For an XCXT, we need two crystal oscillators (\$0.40/unit) and an additional unit that can do the compensation. For a prototype, we estimate that the whole algorithm could fit in hardware onto less than 100,000 gates. An appropriate FPGA would be the Xilinx Spartan-3E XC3S100E FPGA that cost \$2/unit². Thus, a prototype costs around \$3 which is well in the lower range of comparable TCXO's cost. Nevertheless, one could

²Price for 500,000 units November 2007

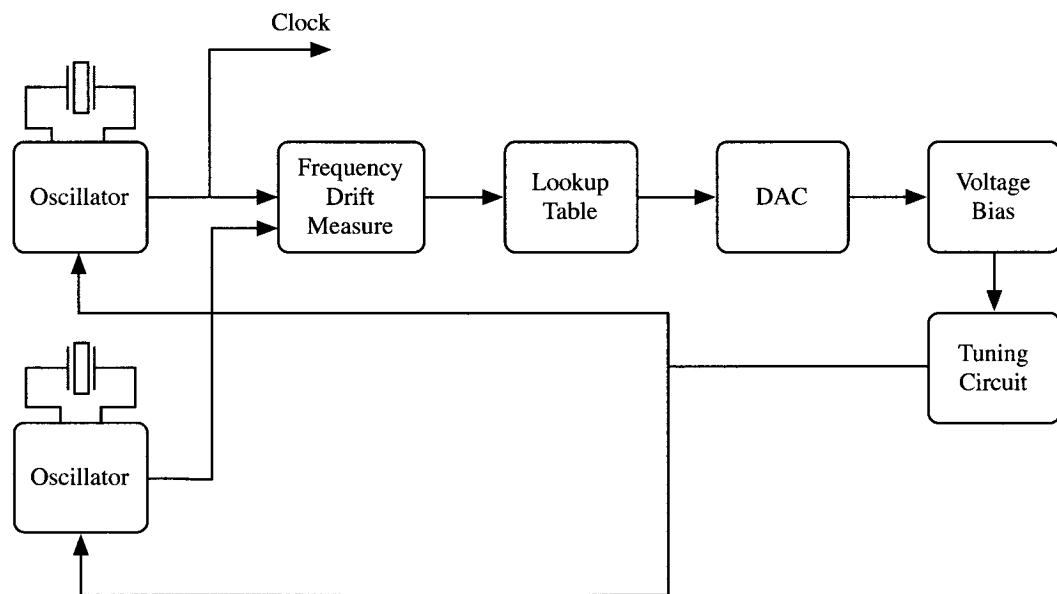


Figure 2.18: Block diagram of an ideal hardware base Crystal Compensated Crystal Oscillator (XCXO).

imagine that the whole circuit would be an additional peripheral on a microcontroller or system on chips (SoC), where the increase in number of gates, and thus cost and complexity, would be negligible. In such a scenario, a XCXT with a precision of <5ppm comes at the cost of <\$1, since it is just one additional crystal.

2.6.3 The Smart Timer Unit

As we showed in the Section 2.5.1, the oscillator circuitry, high speed crystals, and the timer unit consume most of the power of the XCXT. Thus, duty cycling one of the two crystals of the XCXT seems to be an efficient method in order to cut down on the energy consumption. Indeed, it is not necessary to have both timers on all the time. For example, if the XCXT experiences a constant environment temperature, the measured drift is always the same and doesn't add any new information for the compensation phase. This is similar in the event where the environment temperature changes only slowly. By this, we mean where the environment doesn't change more than $\pm 1^{\circ}\text{C}$ per minute. In such an environment, the second crystal could be turned off for 58 seconds. Then, the crystal is turned on for 2 seconds. The first second is there to let the crystal stabilize itself. In general, this should not take more than a couple of tens of milliseconds. In the second second we perform the drift measurement and thus compensate for any change in temperature. At the same time, we saved approximately 48% of the total power, since one of the crystals is turned off most of the time.

A second improvement can be done by using one slow crystal, like a standard low frequency 32kHz crystal as it is found on many real time clocks and embedded systems, and one high frequency one, like the ones we use in the XCXT. The advantage in a duty-cycled system is evident, since the 32 kHz crystal can be used to bridge the time while the node is asleep, and doesn't use a high granularity clock. Once the microprocessor wakes up, the fast clock is turned on and provides a high granularity

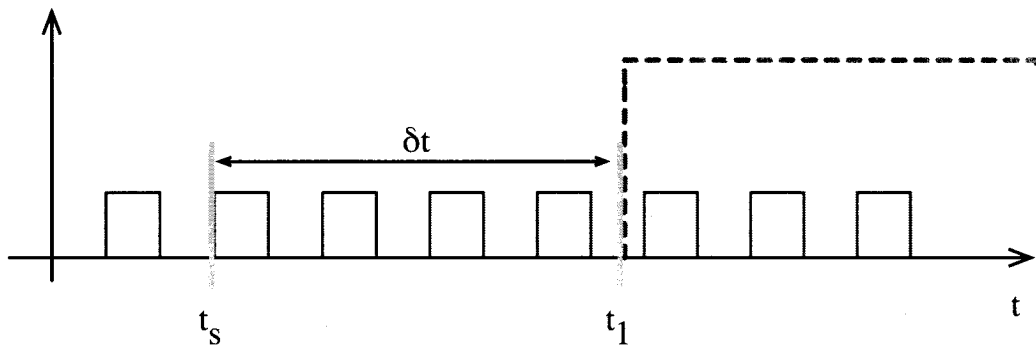


Figure 2.19: This is the timeline of the smart timer unit. We can find the phase of the slower timer (red, dashed) since the time instance t_s by counting the number of ticks of the fast clock (blue, solid)

clock source to measure precise time. At the same time, the two crystals compensate each others drift using the differential drift algorithms. The problem is that the current XCXT algorithm does not work in such a scenario since it is based on the fact that both crystals have the same speed and thus time resolution (minus their drift, which we calculate in the algorithm). Nevertheless, we developed a technique that extends the XCXT algorithm and allows the usage of a slow and a fast crystal. We will now explain the basics behind the extension, although we cannot provide a prototype yet, since there are still some technical problems left we need to solve. We will mention them at the end of this section.

The main idea behind the extension is to interpolate the slow frequency and measure its phase by using the high frequency clock signal. Figure 2.19 illustrates this concept. The blue line (solid) represents the fast clock signal, and the red line (dashed) is the slower clock. Let's assume that the sampling interval is f_s , timer A uses the slow clock signal, timer B the fast, and the two timers are started at the exact same time. At the sampling time t_s we read both timer values as $C_A(t_s) = C_{A0}$ and $C_B(t_s) = C_{B0}$. Now, in

order to find the phase of the timer A we wait until timer A's clock signal rises again. Let's call this time t_1 . We know that $C_A(t_1) = C_{A0} + 1$ and $C_B(t_1) = C_{B0} + \lfloor \delta t_1 \cdot f_B \rfloor$ where the floor operator comes from the digital nature of the timers. Thus, we can calculate the phase of timer A as

$$\phi_A(C_B) = \frac{f_B/f_A - (C_B(t_1) - C_B(t_s))}{f_B/f_A} = 1 - \frac{\lfloor \delta t \cdot f_B \rfloor}{f_B/f_A}$$

This phase information can now be used to calculate the drift between the two crystals that source the timer A and B, and the same differential drift algorithm as for the XCXT can be applied.

There are still some technical problems left that need to be solved, in order to implement the smart timer unit in hardware:

- Commercially available 32kHz crystals are all tuning-fork based. A tuning-fork crystal has a quadratic temperature vs. frequency drift curve. Combining this with a cubic curve from a high frequency AT-cut crystal does not yield a calibration function and can thus not be used for calibration, i.e., there is no unique mapping from $C_{12} \rightarrow C_{10}$.
- The time between the high frequency timer interrupts, and the time the slow timer signal rises (δt in Figure 2.19) can be very short. We will see if the timer hardware is fast enough to rearm the input capture of the fast timer such that an earlier result doesn't get overwritten. A possible solution is to use multiple timer capture registers.

Chapter 5 explores the STU further and provides an actual implementation of an STU in an FPGA.

2.7 Summary

This chapter described the XCXT, a new way of compensating a pair of crystals and thus achieved a $\pm 1.2ppm$ precision over a temperature range of -10 to 60°C . The XCXT can achieve this precision while using only 1.27mW . Benchmarking the XCXT to pinpoint the components that consume most of the energy reveals that the two 8MHz crystals and their timer units consume more than 90% of the XCXT's power budget. Therefore, two different ways on improving the algorithm to minimize the crystals power consumption are one, to simply duty cycle one of the crystals and gaining 48% power efficiency, and two to use two crystals, one fast, and one slow. The fast crystal can be used if high granularity time is needed, and the slow while the system is in sleep. Still, both crystals are used to compensate each other's drift and thus provide a highly stable timer unit. We will explore the later possibility further in Chapter 5 of this dissertation, with a slightly different motivation.

Even though the XCXT improves clock stability, power efficiency, and reduces cost, it still relies on a factory calibration (up to 40% of the end cost), and does not exploit any of the sensor networks inherent communication capabilities. In the next chapter, we will first explore the effects on temperature on networked time synchronization, before we refine the XCXT principle to use communication.

CHAPTER 3

The Effect of Temperature on Time Synchronization

3.1 Introduction

Networks of embedded sensing devices have achieved a level of maturity that allows them to delve into areas that were dominated by wired solutions. Early research platforms demonstrated the feasibility of constructing low power wireless embedded sensors using readily available hardware. Recently emerging industrial applications of embedded sensing devices have begun to demand increased system performance and greater system reliability. These requirements are leading new systems away from original hardware decisions and towards more customized solutions.

Sensor network research has formally examined these emerging pressures and helped formalize guidelines for application specific hardware designs [DTJ08]. For example modern distributed sensor devices now select from a diverse range of radios each optimized for a specific domain. Foliage dense environments may necessitate a platform using the 433MHz CC1000 radio [HPC08], while other deployments may require the higher bandwidth provided by a CC2420 [PSC05] or even WiFi radio. Similarly, a diverse selection of processors are used in different application domains ranging from the low-power MSP430 to powerful ARM class processors [PSC05, DTJ08].

Just as specific applications have required a refinement of radio and processor selection, upcoming sensor network applications involved with control, measurement,

and automation will require a range of timing solutions that are not currently supported. In these domains the robustness of local clock systems and time synchronization protocols is critical.

Current state of the art time synchronization protocols in sensor networks [MKS04, SKL06] achieve an accuracy between 1 to 10 *microseconds*. This level of accuracy suffices for many environmental monitoring applications and acoustic localization systems (e.g. a sniper localization system [SML04]). However, many systems from the wired network community already need significantly more accuracy for control and measurement! The Precision Time Protocol, IEEE 1588 [Eid06] was developed for accuracies below 100 *nanoseconds* and was standardized in 2002¹ because NTP [Mil91], the de facto standard to synchronize computers over the Internet, did not provide enough accuracy. There is currently no implementation available for IEEE 1588 over wireless channels and only preliminary studies have been conducted on the feasibility of such a protocol [CEP07]. Problems occur because uncertainties are introduced by the wireless channel. Deep fading, interference, unpredicted latencies due to the broadcast nature of a wireless channel, and higher bit error rates than in wired networks all complicate synchronization efforts.

To achieve the required accuracy, any IEEE 1588 implementation will require resynchronization almost every second, due to clock uncertainties, to remain within specification and even more frequently for accuracies beyond the baseline. An implementation of IEEE 1588 in sensor networks would face an additional issue due to these frequent resynchronization periods – power consumption. In sensor networks, where energy is scarce and communication is the largest component of the energy budget, these frequent communication exchanges are clearly infeasible. As a matter of fact, the power consumption of high accuracy time synchronization protocols in sensor

¹The latest revision is from 2008, which addressed several performance enhancements.

networks has largely been ignored. While minimizing for the number of transmitted messages does optimize the power consumption, the local clock power consumption of the underlying clock systems is as important, and becomes even more dominant the lower the duty cycle of the system.

3.1.1 Contribution

An understanding of the intrinsic interactions between the scattered local clocks and their cumulative effect on network-wide performance is incumbent upon anyone using wireless embedded sensors in time sensitive applications. This chapter carries the readers through a detailed look at the interactions between clocks, power, and synchronization on low power embedded devices. The main contribution of this work is the formal introduction of temperature into the time synchronization error, and how they are linked to each other. With this formalization, we show that the error can be divided into two regions, and that the intersection of these two regions is the optimal operation point for a time synchronization protocol.

3.2 Understanding Power Consumption in Duty-Cycled Devices

To motivate the subsequent exploration into the interactions between clocks and power consumption, we present a brief case study on duty cycled systems. Duty cycling has become a critical technique to minimize the power consumption in wireless embedded sensing devices. The intuition behind this is clear. Keep hardware in a low power sleep state except during the infrequent instances when the hardware is needed. In many environments this allows even the processor to be put into a low power state for extended periods of time while only an external clock tracks time to trigger a later wakeup.

Recent work has identified clock stability as a limiting factor for duty cycling in networks of devices using scheduled communication. When duty cycling in scheduled communication systems it is important that the communicating nodes wake up at the correct time so that they can communicate. Less stable clocks require nodes to more frequently resynchronize to account for clock frequency error. For example, Dutta showed in [DCS07] that the lower bound of a clock with a stability of $\pm 50\text{ppm}$ is a duty cycle of 0.01% for a scheduled communication MAC protocol. Thus, more stable clocks or better synchronization techniques are necessary in order to improve the duty cycling capabilities of embedded systems.

It thus appears, at first glance, that high stability clocks will reduce power consumption by improving duty cycling of these devices. With our interest in high resolution timers and work developing a smart timer unit, we looked to apply our work to increase duty cycling on sensor devices and thus decrease power. Our formalization of this problem revealed an unexpected surprise.

Currently available high stability clocks do *not* reduce power consumption of duty cycled embedded sensing systems due to the increased power consumption of the clock. This section presents this formalization for a scheduled, and a polled MAC protocol, alongside a precise numerical analysis for a concrete case comparing a standard 32kHz crystal oscillator clock to a potential high frequency stability 8MHz clock.

3.2.1 Clock Stability and Duty Cycling

There are two major MAC communication modes, scheduled, and unscheduled. In scheduled networks, such as TDMA MAC protocols, each node gets assigned a specific time when it is allowed to transmit, or receive. Thus, precise timing is indispensable or the messages would start to collide, or nodes would miss messages intended to them. The later case, unscheduled MAC protocols like B-MAC, don't establish any schedule

and assure through long preambles that the receiving party will be able to receive a message. This makes the protocol much simpler, by putting the burden of assuring connectivity to the transmitter, and was shown to be very effective in low load networks. An improvement on this simple system is WiseMAC, where the nodes learn the wakeup schedules of their neighbors, and thus only start transmitting when the neighbor is about to wake up. The following subsections will describe a formalized analysis of the impact of clock error on the duty-cycle and average power consumption of these two MAC protocol classes.

3.2.1.1 Scheduled Communication

The MAC protocol is one of the main components that enables duty-cycling in networked systems, i.e., networked nodes need to wake up at the same time in order to communicate with each other. SCP-MAC [YSH06] is considered to be the current state of the art for sensor networks and achieves a duty cycle as low as 0.1%. In [YSH06], Ye et al. give a theoretical analysis of the SCP-MAC protocol performance based on a standard 30ppm crystal oscillator. Using the same methods and code generously provided by the authors of [YS08], we studied what the improvements are given a better clock system. Figure 3.1 illustrates the improvement if we go from a 30ppm clock source to 1ppm. We can see that we have a 30% to 50% improvement in duty-cycling, which results in a 30% to 50% improvement in battery lifetime. There is only one drawback in this analysis. Since the duty-cycled node already consumes only $67\mu W^2$ in the worst case, an improvement of 50% means a gain of about $30\mu W$. Therefore, the improved clock system cannot use more than $30\mu W$ or else the gain in energy would be nullified.

²Power consumption of a TI MSP430 and a ChipCon CC2420 radio chip as found on the TMote Sky platform[Mot]

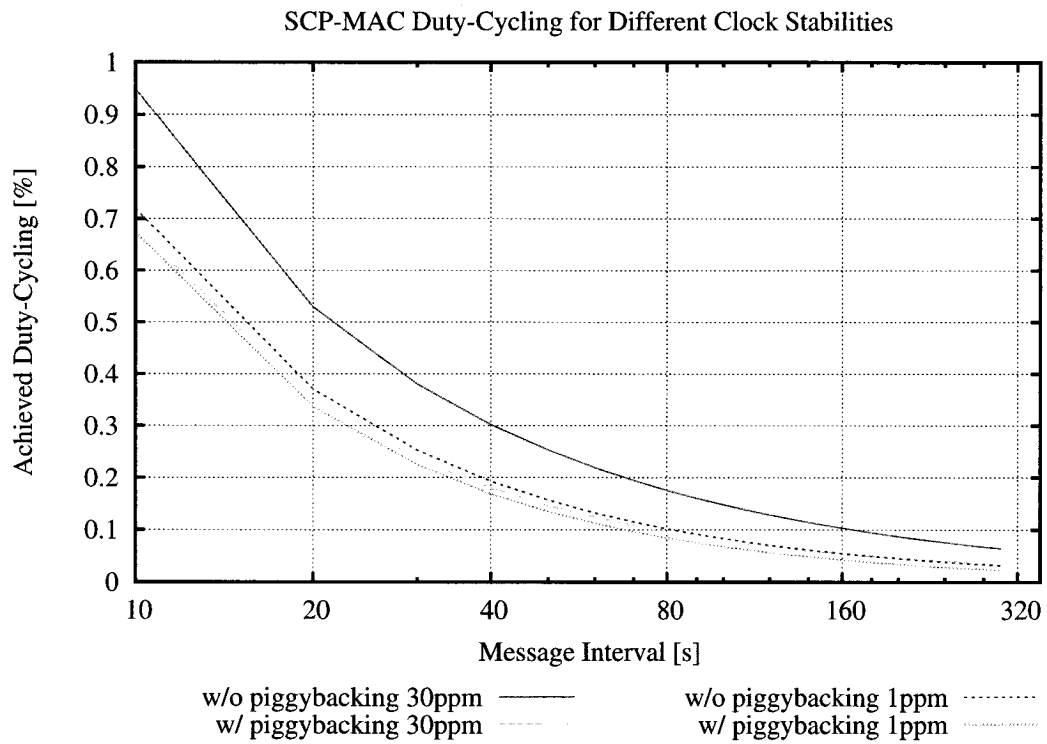


Figure 3.1: Duty-Cycling performance of SCP-MAC using different clock source stabilities. Note that with a 1ppm clock source, the difference between piggybacking the sync messages on regular data messages or not virtually disappears.

This short analysis of SCP-MAC shows that the clock system plays an important role in duty cycled systems. Great care has to be given to its design, or else its possibilities of saving power are easily offset by an increased standby power consumption of the more accurate clock system. Additionally, as we showed in Section 2.5.1, if high accuracy and high granularity clocks are needed for a specific application, like localization systems or time synchronization between nodes, the standby power consumption of such clocks significantly increases and cannot be ignored if the nodes need to be duty cycled.

3.2.1.2 Polled Communication

In the following formalization, we assume a MAC protocol similar to WiseMAC [ED04], where a node learns the wakeup schedule of its neighboring nodes. However, it is still not establishing a time schedule, as in a TDMA protocol, and thus the node has to incorporate guard bands to assure connectivity.

We are interested in the average power consumption of two systems, M and N , using different clocks. We assume both systems have the same sleep power consumption P_s and active power consumption P_a . Each node has a local clock source with frequency stability s_M and s_N (in Hz/Hz) that consume power P_{cM} and P_{cN} , respectively. Additionally, we assume that both platforms have the same duty cycle ratio DC between sleep (T_s) and active time (T_a). However, in order to communicate with their peers, both systems include a guard time that is proportional to their local clock source's frequency stability $T_{gX} = 2 \cdot T_s \cdot s_X$, where s_X is the system's local clock frequency stability. This guard time allows the nodes to compensate for the drift in their clock frequency while asleep by starting the communication process early enough to ensure that both nodes are awake when the active period starts. The guard time is set by the communication period, which, in this case, is the sleep period since we assume nodes will resynchronize when they are awake. Note that communication in this case comes

for free, since it implicitly happens whenever a message gets transmitted from M to N , or vice-versa.

The average power consumption of this system is simply the sum of the power consumed in sleep mode and the power consumed in awake mode, normalized by the total time the system is on:

$$P_X = \frac{T_s \cdot (P_s + P_{cX}) + (T_{gX} + T_a) \cdot (P_a + P_{cX})}{T_s + T_a + T_{gX}}. \quad (3.1)$$

where the subscripted X is either M or N . Let us also assume that, without loss of generality, N is a more stable clock such that $s_M > s_N$ and consequently $P_{cM} < P_{cN}$. In order for system N to be more efficient than system M we have to show that

$$P_M > P_N, \quad (3.2)$$

and thus

$$\frac{T_s \cdot (P_s + P_{cM}) + (T_{gM} + T_a) \cdot (P_a + P_{cM})}{T_s + T_a + T_{gM}} > \frac{T_s \cdot (P_s + P_{cN}) + (T_{gN} + T_a) \cdot (P_a + P_{cN})}{T_s + T_a + T_{gN}}. \quad (3.3)$$

Quartz crystals are commonly used in embedded systems, are inexpensive, and are a representative clocking mechanism. For comparison, we now assume that the clock of system M is a quartz crystal. These crystals consume very little power, and thus $P_{cM} \sim 0$. This assumption is justified by our precise numerical analysis presented in Section 3.2.1.3. Using the assumption that $P_{cM} \sim 0$, Equation 3.3 can be simplified to

$$P_{cN} < \frac{2 \cdot T_s^2 \cdot (P_a - P_s) \cdot (s_M - s_N)}{(T_s \cdot (1 + 2s_M) + T_a)(T_s \cdot (1 + 2s_N) + T_a)}, \quad (3.4)$$

where T_{gM} and T_{gN} are replaced by their respective definitions. By definition, the duty cycle of the system is $DC = T_a / (T_a + T_s)$. Since clock stability is fairly low (usually measured in *ppm*) with $s_M, s_N \ll 1$, the relation can be further simplified to

$$P_{cN} < 2 \cdot [1 - DC]^2 \cdot (P_a - P_s) \cdot (s_M - s_N). \quad (3.5)$$

This relation shows that for system N to be more energy efficient than system M , its clock power consumption cannot exceed a threshold dependent on the duty cycle, the difference in active and sleep power consumption, and the difference in their clock stability. If we further assume that $DC \ll 1$, $P_s \rightarrow 0$, and $s_M \gg s_N$ a final simple relation is reached:

$$P_{cN} < 2 \cdot P_a \cdot s_M. \quad (3.6)$$

This reveals that for system N to be more energy efficient than system M , the clock system used in system N must use less power than the active power consumption multiplied by twice the precision of system M 's clock stability. For example, if we assume that $P_a = 1W$, $s_M = 50ppm$, and $s_N = 1ppm$ (in order to satisfy $s_M \gg s_N$) then $P_{cN} < 100\mu W$. We are currently unaware of any commercially available technology that can achieve a clock stability of $1ppm$ with a power budget of less than $100\mu W$. The closest available clock is the MAXIM DS32BC35 that is an RTC with integrated 32kHz TCXO, consuming about $600\mu W$ for a stability of $\pm 3.5ppm$. In research, low-power, and high stability clocks have been presented [AOV97], however none of them are readily available despite the more than 10 years of their existence.

3.2.1.3 Adding Time Precision

The formalization provided above disregarded clock resolution that prevents arbitrary sleep periods due to the discretization of time by the clock ticks. Clock resolution can be easily added into our formalization. The only term that changes are the guard bands. Now the device must wake up $1/f_X$ seconds earlier in order to guarantee that the node is awake at the correct time. Thus, the guard bands become $T_{gX} = 2 \cdot T_s \cdot s_X + 1/f_X$.

Our formalization also made a number of assumptions to simplify the resulting inequality presented in Equation 3.5 that must be verified. We add time precision to Equation 3.3 and numerically solve the equation to verify our formalization. We

use the same parameters as in the last example, representing a typical wireless sensor network platform: $P_a = 1W$, $s_M = 50ppm$, and $s_N = 1ppm$. We further add $DC = 0.01$, $P_s = 5\mu W$, $P_{cM} = 10\mu W$, $f_M = 32.768kHz$, and $f_N = 8MHz$. Numerically solving Equation 3.3 extended with clock guard bands taking into account clock resolution reveals the inequality $P_{cN} \leq 106.9\mu W$. This is nearly identical to the result from our formalization. We conclude that for guard band reduction, there is no incentive to increase the clock resolution beyond the very common 32kHz tuning fork crystal clocks.

3.3 Clock-Characteristic Impacts on Resynchronization Rate

An isolated clock cannot improve or correct its accuracy by itself. However, a network of clocks with communication capabilities, can use other clocks to verify their accuracies, and possibly increase their stability. Many time synchronization protocols, like [MKS04, GGS05, EGE02], showed that high time synchrony can be achieved by a group of nodes. The general approach is to estimate the current frequency error between the nodes using message time stamping, and different estimation techniques. One common assumption of these protocols is that the temperature, and thus the frequency error, is constant over the time intervals these synchronization protocols operate on.

The remainder of this chapter investigates the impact of temperature on the frequency error, and shows that depending on the system clock characteristics, frequency error can become a problem, especially if one wants to increase the sleep periods between resynchronization points to minimize energy consumption. The following analysis sheds light onto the decision of how often a time synchronization protocol has to resynchronize in order to stay within a specific time synchronization error.

3.3.1 Temperature Effects on Frequency Error

The biggest contributor to frequency change is the change in environmental temperature. The temperature dependent frequency can be expressed using the nominal frequency f_0 as

$$f(t) = (1 + \delta(t)) \cdot f_0, \quad (3.7)$$

where $\delta(t)$ is the frequency error over time and can be expressed as $\delta(t) = g(\kappa(t))$. Here, $g()$ is a function defined by the resonating element³, and $\kappa(t)$ is the temperature over time. Given this new definition, the counter on a system reads at any time t since the counter was reset

$$c(t) = \left\lfloor \int_0^t f(\tau) d\tau \right\rfloor = \left\lfloor f_0 \cdot \left(t + \int_0^t g(\kappa(\tau)) d\tau \right) \right\rfloor \quad (3.8)$$

3.3.2 Time Synchronization Error

We assume a simple two way time synchronization protocol, like IEEE 1588, for our error analysis. More precisely, we consider a two node network, where one node is the master, and the second node is the slave. We further assume that the master clock has perfect time, and that the slave node tries to synchronize to the master node's time reference. The counter on the master node will read $c_0(t) = \lfloor f_0 \cdot t + \varphi \rfloor$, where φ is a random phase offset between $[0, 1)$. To make the analysis simpler, we assume that there are no timestamping errors. We will relax this assumption later on.

The master and slave node regularly exchange time synchronization messages that contain precise timestamping information. Using these timestamps, the slave node can accurately estimate the current offset between its local clock and the master clock. The slave node estimates its current frequency error using the two last offset measurements exchanged with the master node. Assuming that these two messages were T seconds

³For an AT-Cut crystal, $g()$ is a cubic curve, or for a tuning fork crystal, $g()$ is a quadratic curve.

apart, the slave node can estimate the local frequency error with

$$\delta_Q = \frac{c(-T) - c(0)}{c_0(-T) - c_0(0)} - 1 = \frac{\lfloor f_0 \cdot T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau \rfloor}{\lfloor f_0 \cdot T + \varphi \rfloor} - 1. \quad (3.9)$$

Using this frequency error estimate, the synchronization error between $t = 0$ and the next time synchronization interval at $t = T$ is

$$\begin{aligned} \varepsilon_Q(t) &= (c(t) - \delta_Q \cdot c(t)) - c_0(t) \\ &= c(t) - \left(\frac{\lfloor f_0 \cdot T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau \rfloor}{c_0(T)} - 1 \right) \cdot c(t) - c_0(t) \\ &= 2 \cdot c(t) - \frac{\lfloor f_0 \cdot T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau \rfloor}{c_0(T)} \cdot c(t) - c_0(t). \end{aligned} \quad (3.10)$$

3.3.2.1 Bounding the Time Synchronization Error

There is no closed form solution for Equation 3.10 due to the non-linearity of the floor operator and the unpredictability of the temperature. However, we can find bounds on the maximum of the time synchronization error.

The fractional part function is defined as

$$\{x\} = x - \lfloor x \rfloor \quad (3.11)$$

for all x , $0 \leq \{x\} < 1$. Thus, we can write the floor operator as $\lfloor x \rfloor = x - \{x\}$. Substituting the fractional parts with the variable v_x and expanding the counter values $c(t)$ to their definition, the resynchronization error from Equation 3.10 becomes

$$\begin{aligned} \varepsilon_Q(t) &= \\ &= 2 \cdot (f_0 t + f_0 \int_0^t g(\kappa(\tau)) d\tau - v_3) \\ &\quad - \frac{f_0 T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau - v_1}{f_0 T + \varphi + v_2} \cdot (f_0 t + f_0 \int_0^t g(\kappa(\tau)) d\tau - v_3) \\ &\quad - (f_0 T + \varphi - v_2). \end{aligned} \quad (3.12)$$

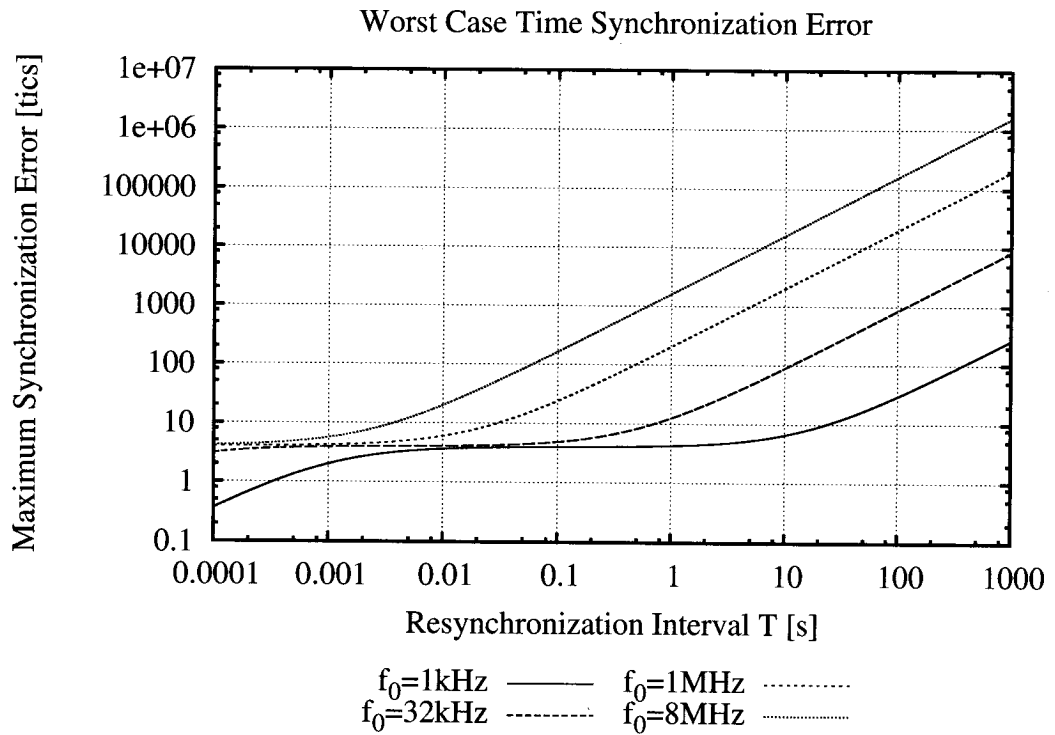


Figure 3.2: Worst case time synchronization error for four different clock speeds. The two low frequency clocks are tuning fork crystals with a maximum frequency error of 120 ppm, and the high frequency clocks are AT-cut crystals with a maximum frequency error of ± 50 ppm.

We can now bound the time synchronization error

$$\begin{aligned}
\varepsilon_Q(t) &\leq \max(\varepsilon_Q(t)) \\
&\leq \max\left(2 \cdot (f_0 t + f_0 \int_0^t g(\kappa(\tau)) d\tau - \nu_3)\right) \\
&\quad - \min\left(\frac{f_i T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau - \nu_1}{f_0 T + \varphi + \nu_2} \cdot (f_0 t + f_0 \int_0^t g(\kappa(\tau)) d\tau - \nu_3)\right) \\
&\quad - \min(f_0 T + \varphi - \nu_2)
\end{aligned} \tag{3.13}$$

By definition, we know that $\min(\nu_x) = 0$, and $\max(\nu_x) = 1$. Additionally, we can bound the integral over the temperature dependent frequency error by

$$\min\left(\int_0^t g(\kappa(\tau)) d\tau\right) = t \cdot \delta_{\min} \tag{3.14}$$

$$\max\left(\int_0^t g(\kappa(\tau)) d\tau\right) = t \cdot \delta_{\max} \tag{3.15}$$

$$\min\left(\int_{-T}^0 g(\kappa(\tau)) d\tau\right) = T \cdot \delta_{\min} \tag{3.16}$$

$$\max\left(\int_{-T}^0 g(\kappa(\tau)) d\tau\right) = T \cdot \delta_{\max} \tag{3.17}$$

where δ_{\min} and δ_{\max} are the minimum and maximum temperature dependent frequency error of the employed clock circuit. Putting it all together, we find that the maximum time synchronization error is bounded by

$$\varepsilon_Q(t = T) \leq 2f_0 T \cdot (1 + \delta_{\max}) - \frac{(f_0 T(1 + \delta_{\min}) - 1)^2}{f_0 T + 1} - (f_0 T - 1) \tag{3.18}$$

Figure 3.2 illustrates the worst case synchronization accuracy for a given time interval between synchronization messages. We can see that there are two different error regions:

1. The synchronization error is dominated by quantization error
2. The synchronization error is dominated by temperature induced frequency drift

In region one, we assume that the frequency error $\delta = 0$, and thus the synchronization error simplifies to

$$\varepsilon_Q(T) \leq 2f_0T - \frac{(f_0T - 1)^2}{f_0T + 1} - (f_0T - 1). \quad (3.19)$$

We can now calculate the limit and find

$$\lim_{T \rightarrow \infty} \varepsilon_Q(T) = 4. \quad (3.20)$$

In the second region, where the frequency drift overshadows the quantization errors, the synchronization error reduces to

$$\begin{aligned} \varepsilon_Q(T) &\leq 2f_0T \cdot (1 + \delta_{\max}) - \frac{(f_0T(1 + \delta_{\min}))^2}{f_0T} - f_0T \\ &= 2f_0T \cdot (1 + \delta_{\max}) - f_0T \cdot ((1 + \delta_{\min}))^2 - f_0T \\ &= f_0T \cdot (2 + 2\delta_{\max} - 1 - (1 + \delta_{\min})^2) \\ &= f_0T \cdot (2\delta_{\max} - 2\delta_{\min} - \delta_{\min}^2) \end{aligned} \quad (3.21)$$

To achieve an optimum between small time synchronization error, and low message exchange rate, a time synchronization protocol has to operate exactly at the border of these two regions. We can find this point by equating Equation 3.20 with equation 3.21 and solving for T . Thus, the optimum resynchronization rate assuming worst case synchronization errors is

$$T^* = \frac{4}{f_0 \cdot (2\delta_{\max} - 2\delta_{\min} - \delta_{\min}^2)} \quad (3.22)$$

3.3.2.2 Introducing Time Stamping Errors

Timestamping error affects the precision with which a node estimates its current frequency error $\delta_Q(t)$ and we can model it as

$$\delta_Q(0) = \frac{c(-T) - c(0) + \eta}{c_0(-T) - c(0) + \eta} - 1 \quad (3.23)$$

where η is the timestamping error. Propagating this change, we find that the maximum synchronization error can be expressed as

$$\varepsilon_Q(t = T) \leq 2f_0T \cdot (1 + \delta_{\max}) - \frac{(f_0T(1 + \delta_{\min}) - 1 - \eta)}{f_0T + 1 + \eta} \cdot (f_0T(1 + \delta_{\min}) - 1) - (f_0T - 1) \quad (3.24)$$

The effect of this change on the graph is a raised error in region (1). However, over longer synchronization intervals, these timestamping errors become negligible, and thus the approximation for the second region does not change.

3.3.3 A More Realistic Bound on the Time Synchronization Error

Our approximations made in Equations 3.15 to 3.17 are very conservative due to the worst case modeling of temperature induced frequency error. It is highly unlikely that in a real system, a clock would experience these extreme conditions consecutively within two time synchronization intervals. For most embedded systems that are not used in extreme environments, like rockets, the temperature changes gradually and thus we can find a tighter bound for the time synchronization error by analyzing actual temperature traces.

Assume that we are given a temperature trace of the environment where our embedded system is located. We can scan this temperature trace and search for the biggest change in frequency error in a window of $2T$. Marking the minimum of this frequency error with δ_{\min} and the maximum with δ_{\max} we can reevaluate Equation 3.18 and get a more realistic bound on the maximum time synchronization error.

We collected three different temperature traces for three different environments. The three traces are each 7 days long, and have a time resolution of 5 seconds. We collected the data in a A/C controlled not sun exposed, a sun exposed not AC controlled, and a shaded not A/C controlled indoors area. Figure 3.3 shows the result for each of these

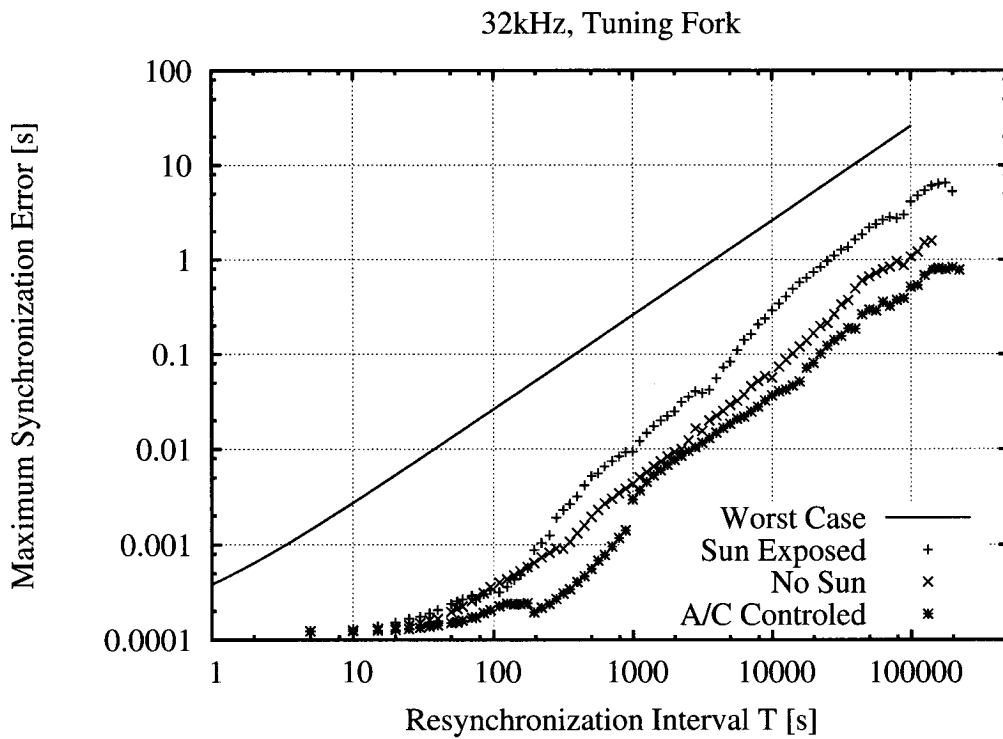


Figure 3.3: This graph shows the effect of using the maximum drift change calculated from a temperature traces on the maximum synchronization error. In general, the smaller the temperature changes, the better the synchronization accuracy for longer resynchronization intervals.

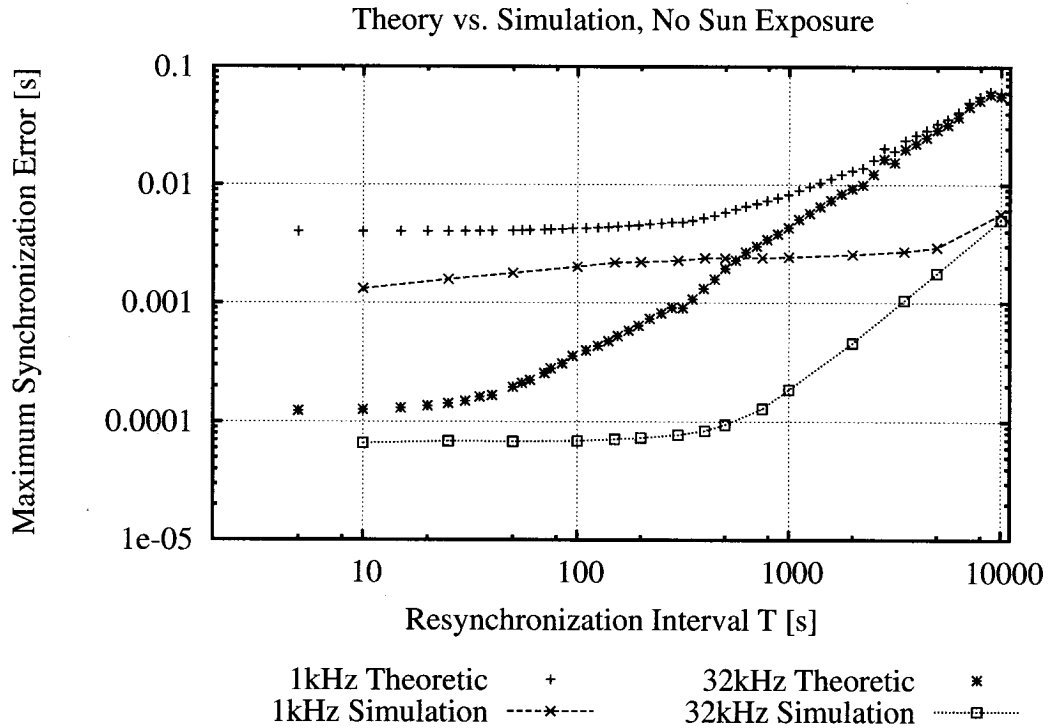


Figure 3.4: This graph compares the theoretical upper bound to the 95% confidence interval found through simulating a time synchronization protocol.

temperature traces and the estimated maximum time synchronization error bound. We can conclude that the more variation there is in the change of temperature, the worse the expected time synchronization error bound becomes. However, we can still see a clear cut between the two regions where (1) the quantization error dominates, and (2) where the temperature induced frequency drift dominates.

3.3.4 Verification Through Simulation

There are many different sensor network simulators, each providing a different set of features and tools. However, none of the existing sensor network simulators captures

the change in frequency error that occurs due to changes in local temperature. At best, some simulators model different static clock frequency offsets [Bou07, VXS07], but these offsets are static over the time of a simulation, and are thus not useful for our experiments.

We decided to enhance Castalia [Bou07] with a revised clock model, that takes as input a temperature trace file and adapts the clock’s frequency error at runtime. In addition, we added a realistic timestamping mechanism that models the timestamping errors found on the CC2420 radio chip used in many sensor network platforms. We implemented FTSP in Castalia to simulate a simple time synchronization protocol.

To verify our theoretical analysis, we run several simulations with changing resynchronization intervals. Figure 3.4 shows the 95% confidence intervals for the measured maximum time synchronization error, and compares the result from theory and simulation for two different clock speeds. The simulations verify that our bounds on the temperature induced frequency drifts are indeed upper bounds, and that in simulation the time synchronization errors are much smaller than the bounds themselves.

3.4 Bringing it All Together: Synchronization, Duty-Cycling, and Power

We discussed the notion and need of duty cycling in section 3.2.1.2, and we have now the tools to add to that analysis the impact of time synchronization on duty cycled systems. More precisely, we introduced the need for guard bands and defined them as

$$T_{gX} = 2 \cdot T_s \cdot s_X, \quad (3.25)$$

where T_s is the sleep time, and s_X the node’s clock stability. Using the worst case time synchronization error from the last section, we can redefine the guard band for a node

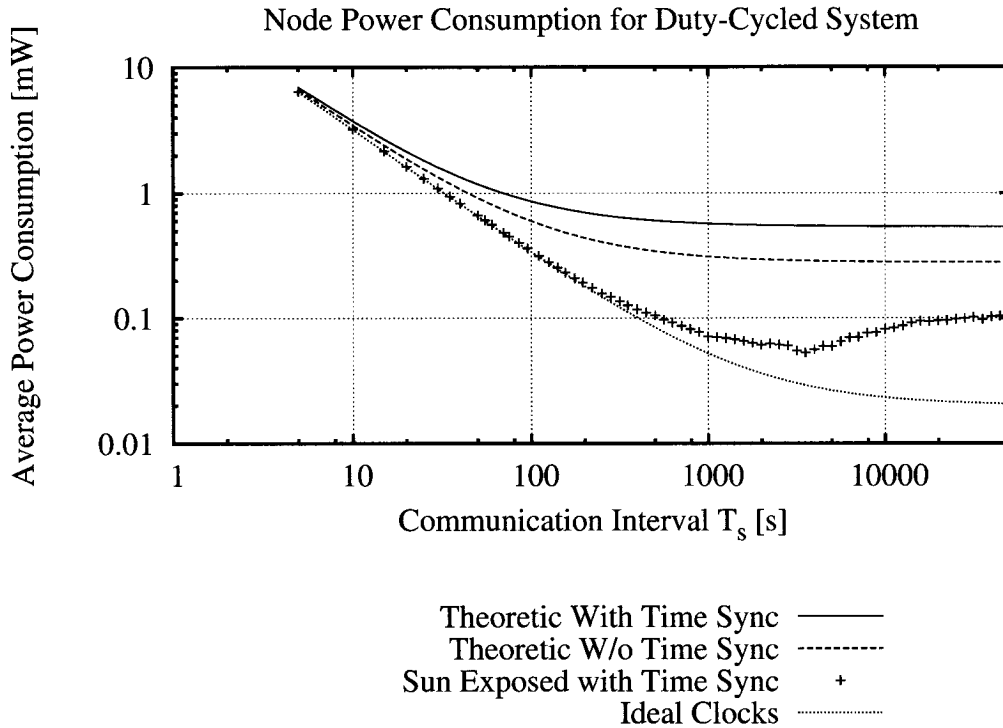


Figure 3.5: This graphs shows the average power consumption for different synchronization techniques in a duty cycled system. With ideal clocks, no overhead is required, and thus a minimum power consumption is achieved. We only show the sun exposed data set because it is a worst case for the three temperature traces we collected.

with time synchronization as

$$T_g = 2 \cdot \varepsilon_Q. \quad (3.26)$$

This bound guarantees that a node that wakes up within T_g can communicate with its neighbors, even if it experiences the worst case synchronization error.

Given the guard band of a duty cycled system without time synchronization, and one with time synchronization, we can calculate their power consumption using Equation 3.1. Figure 3.5 depicts this comparison using the same system parameters as was used

in Section 3.2.1.3. In addition, Figure 3.5 shows the ideal case power consumption where the guard band $T_g = 0$.

It is not surprising that in the theoretical worst case synchronization error calculations, a system without time synchronization fares better than one with time synchronization. The cause of this is that in the worst case, the frequency error estimation for δ_Q can introduce twice as much error as without time synchronization. However, in reality the occurrence of such an error is highly unlikely. Thus, using the more realistic time synchronization error bounds developed in Section 3.3.3 already shows that using time synchronization can greatly improve the power efficiency of a duty cycled system.

This analysis ignores the power overhead that a time synchronization protocol introduces. Unfortunately, this overhead is poorly studied and we can only speculate on how much this overhead would be in a real system. In general, if a system needs less than 10ms accuracy, then we showed that a time synchronization interval of up to 1000s is sufficient to guarantee such accuracies. In most systems, this interval is much larger than the communication interval necessary to transmit sensor data from the nodes to a fusion center, and we can assume that the overhead of time synchronization in these scenarios is minimal. However, if high accuracy is needed, the synchronization intervals have to be on the order of tens of seconds. In these cases, time synchronization could be piggy-backed on regular data messages, only introducing a small message overhead on the order of a timestamp.

3.5 Summary

The theoretical analysis and validation through simulation presented in this chapter shows that the maximum synchronization error has a two-region behavior. In the first region, clock quantization error dominates any other errors, and in the second region,

frequency drift due to changes in temperature overshadow other errors. This shows that there is no gain in operating a time synchronization protocol with a resynchronization rate in the first region. It will only consume more power due to higher message rates, and not gain anything in terms of time accuracy. Operating a time synchronization protocol inside the second region decreases the time synchronization accuracy due to changes in frequency, however we gain in terms of a lower overall power consumption, since fewer radio messages are necessary.

The analysis shows that the power aspect of clocking subsystems are non-trivial, and that sometimes a simple assumption, like more stable clocks will save power in smaller guard band, have to be investigated carefully before taken for true. This chapter concentrated on the better understanding of the link between clocks, power, and synchronization. Though this is not the end of the story. What if an embedded systems needs very high absolute accuracy (demands high-frequency clocks)? In that case, the second region of resynchronization intervals still demands a high resynchronization rate. $2\mu s$ at a resynchronization rate of 30 seconds seems to be the lower end of what is possible for time synchronization in a wireless sensor network with current technology. Our analysis is a first attempt to explain that going beyond that accuracy with reasonable average power consumptions for wireless sensor networks is impossible if we don't incorporate and locally compensate for the change in temperature induced frequency error. The next chapter will investigate this further, and develops a temperature compensating mechanism using regular time synchronization, and the local knowledge of temperature changes given by an on-board temperature sensor.

CHAPTER 4

Temperature Compensated Time Synchronization

4.1 Introduction

Synchronizing two embedded systems over a wireless channel is a challenging problem. One has to consider several error sources including the accuracy of time stamping of a common event, message loss due to wireless channel fading and shadowing, and changes in the clock frequency due to environmental changes (temperature, acceleration, etc.). In addition to this, sensor networks add yet another challenge into the equation: energy efficiency. Several synchronization protocols have been proposed in recent years, addressing many of these problems.

In Reference Broadcast Synchronization [EGE02], every node keeps the relative drift between its local clock and every other clock in the network. By comparing the timestamps of periodic broadcast messages, the nodes calculate the clock offsets between the receiving nodes, thus successfully eliminating any transmit latencies. Only processing delay at the receiver and the difference in propagation delay between the nodes are potential error sources.

A more complex model is solved by the Timing-sync Protocol for Sensor Networks (TPSN, [GKS03]). In TPSN, every node tries to synchronize to one reference node through the establishment of a synchronization tree. In addition, TPSN uses a handshake synchronization exchange which eliminates receive, transmit, and propagation delays. This handshake is similar to what is used in the Precision Clock Synchronization

Protocol (IEEE 1588, [Eid06]). However, IEEE 1588 is targeted for wired Ethernet networks, and only early results on the feasibility of IEEE 1588 over IEEE 802.11 have been published [CEP07].

The Flooding Time Synchronization Protocol (FTSP, [MKS04]) improves upon TPSN through an elaborate and accurate method of timestamping messages. In addition, FTSP contains a reference node election mechanism and implicitly establishes a synchronization tree. This makes the synchronization algorithm extremely simple, and thus has become one of the most popular synchronization protocols used in sensor network research. One problem that plagues FTSP, and any other synchronization protocol relying on a synchronization tree, is that two nodes that are in two different synchronization branches can still be radio neighbors. It has been shown that the synchronization accuracy of these two radio neighbors can be severely impacted, due to the fact that synchronization errors propagate differently down each synchronization branch. A solution to this problem is provided by the Gradient Time Synchronization Protocol (GTSP, [SW09]). In GTSP, the nodes listen to the beacon messages of all their neighbors and update their local time using these timestamps. Thus, there is no reference node nor synchronization tree within the network. Even though this solves the problem of high synchronization errors between neighbors, it introduces the problem of how to anchor the network to some global reference, such as Coordinated Universal Time (UTC).

One problem none of these synchronization protocols addresses is the change in frequency over time due to temperature variation. Every protocol assumes that the temperature changes slowly enough within one synchronization period, and thus the resulting frequency error is constant. We showed in Chapter 3 that there are clear limits to this assumption. One attempt at solving this problem was introduced in the Rate Adaptive Time Synchronization Protocol (RATS, [GGS05]). RATS uses a model of

long term clock drift in order to predict the synchronization interval. It can thus achieve with high probability an application specific synchronization accuracy, while optimizing the resynchronization interval. However, RATS does not use temperature measurements directly, and thus ignores an information source responsible for clock drift itself.

It is a well known fact that the quartz crystal's resonant frequency changes with change in temperature. This led early on to the development of Temperature Compensated Crystal Oscillators (TCXO, [ZZX05]). In a TCXO, the crystal is co-located with a temperature sensor. After an initial factory calibration, where the typical frequency vs. temperature curve of the particular crystal is measured, a small compensation mechanism measures the temperature during runtime, and accordingly tunes the crystal to the right frequency. There are two major problems with using a TCXO in sensor network platforms: (1) high cost and (2) high power consumption. The increase in cost comes from the necessary initial calibration process, and the higher power consumption is due to the addition of electronic parts in the TCXO. At the same time, new low-power communication schemes, like Koala [MLT08] turn the radio off for days at a time. After such long sleep intervals, their clocks accumulate a significant time error due to changes in temperature, and thus have to increase their guard bands. The increased guard bands waste energy, and thus temperature compensated clocks could help to minimize this effect.

The question we address is whether a time synchronization protocol can be used to temperature calibrate the local clock? The contribution of this work is the development of such a protocol, the Temperature Compensated Time Synchronization Protocol (TCTS). TCTS is a technique that autonomously learns the calibration parameters of the local crystal to essentially provide a stable TCXO. TCTS enhances standard synchronization protocols by allowing them to increase the time between resynchronization beacons, without impacting the synchronization accuracy. This increase in resynchro-

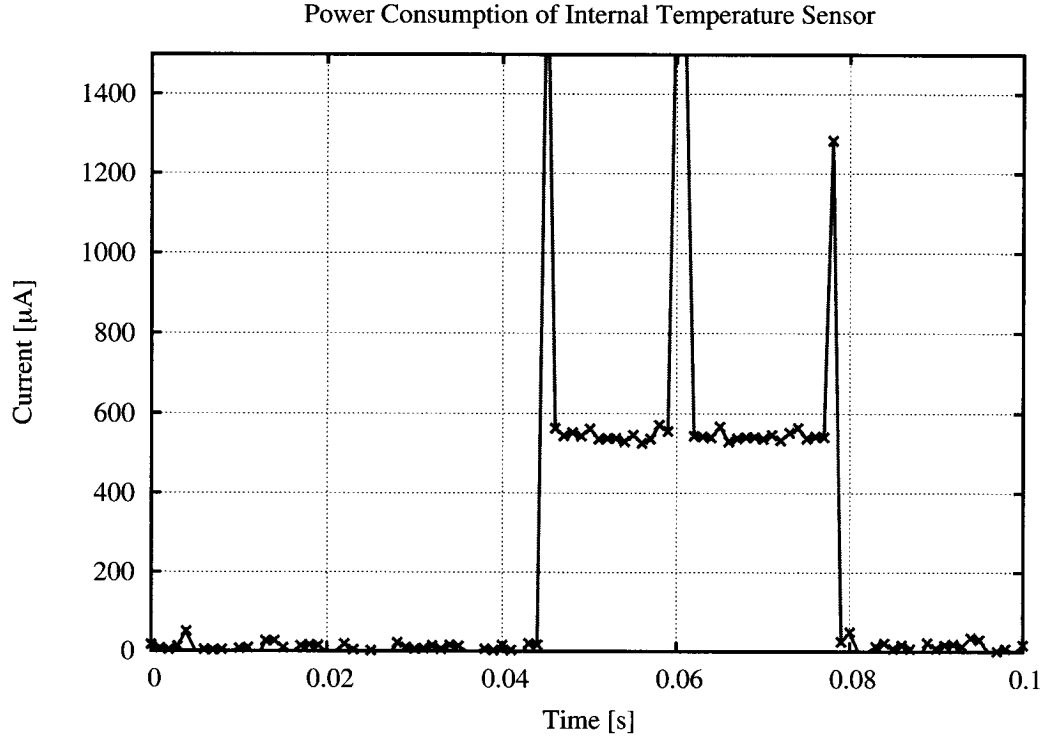


Figure 4.1: Current profile of a TMote Sky while sampling the internal temperature sensor. It takes 35ms to wakeup the CPU, stabilize the internal voltage reference, take the sample, and shut down the CPU again. During that time, the node consumes a total of $66.5\mu\text{J}$, or about 10% of what a radio message consumes ($\sim 600\mu\text{J}$).

nization period reduces the impact of time synchronization on power consumption, as well as communication overhead. An additional benefit is a more robust time base in case of communication loss or extremely long intercommunication periods.

4.2 The Cost of Measuring Temperature

One important factor in TCTS is that every node has to locally measure the temperature. But this measuring doesn't come for free and costs energy itself. We measured the

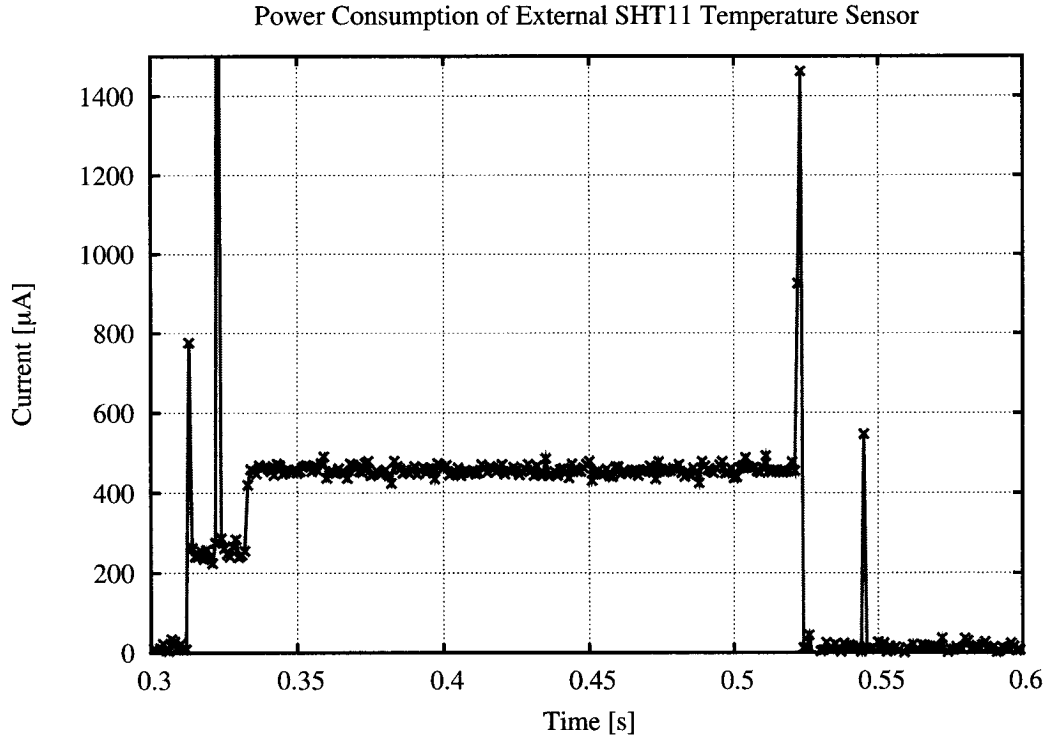


Figure 4.2: This is the current consumption of a TMote Sky while sampling the external SHT11 temperature sensor. It takes a total of 220ms or about $287\mu\text{J}$.

energy consumed during temperature measurement of a TMote Sky platform running TinyOS 2.x, in order to get a sense of how much energy is used to just measure a temperature sensor, compared to sending a message over the radio. The TMote Sky has two different temperature sensors, the internal on-chip sensor, and an external Sensirion SHT11.

Figure 4.1 shows the current profile when a node wakes up, samples the internal temperature sensor, and goes back to sleep. Figure 4.2 shows a similar graph for the case of using the external SHT11. Integrating over the current curve, and multiplying by the power supply voltage of 3V gets us the energy consumed for just measuring the temperature. In the case of the internal temperature sensor, this corresponds to $66.5\mu\text{J}$,

and for the external SHT11 to $287\mu\text{J}$. This is almost one order smaller for the internal sensor compared to just sending or receiving a radio message which costs about $600\mu\text{J}$ [HC08]. It is to mention that for the SHT11 case, the microcontroller does not shut off while the SHT11 prepares the temperature sample. Such a modification could greatly improve the power profile in that case, and drop the current consumption much further down.

This shows that a node can sample a temperature sensor at least 10 times, before it consumes as much energy as sending one radio message. However, the radio message will most likely incur cost on several nodes (one node broadcasting one message will get received by everyone else, thus greatly multiply the overall network power consumption). Thus, a big gain can be made if a node can reduce the amount of messages sent over the radio by measuring the local temperature instead.

4.3 Frequency Error Estimation

The nominal frequency of a local clock f_0 is the fundamental unit of a time synchronization protocol. It is impossible for a system to achieve a better accuracy than $1/f_0$ due to quantization effects in the local clock. Thus, the local frequency has a major impact in how accurately a time synchronization protocol can estimate its current frequency error. More precisely,

$$\delta_E(T) = \frac{1}{f_0 \cdot T} \quad (4.1)$$

where δ_E is the error in the frequency error estimation, and T the resynchronization time between timestamp beacons.

Equation 4.1 suggests that the longer we wait between synchronization exchanges, the better the accuracy of our frequency error estimation. Unfortunately, this is only true

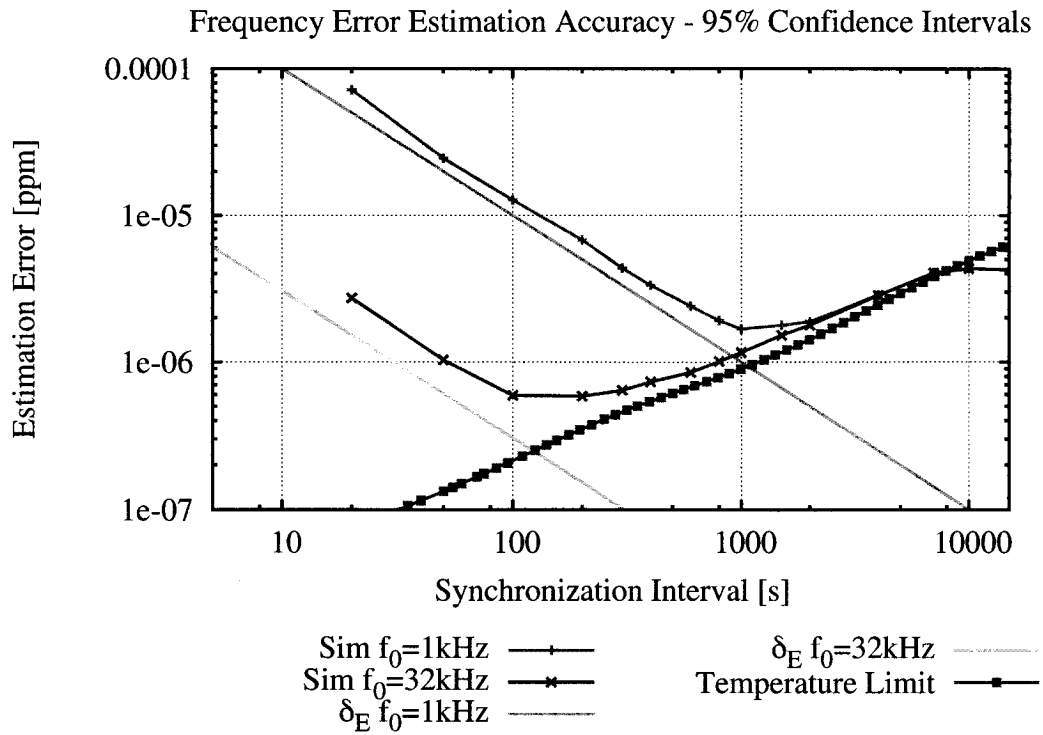


Figure 4.3: The error in the frequency error estimation of a sensor node is hampered by two different phenomena: (1) quantization due to the digital nature of a clock, and (2) temperature induced frequency drift. Simulation shows that there is an optimal resynchronization period at which estimation error is minimized.

in an environment where the temperature doesn't change. In reality, the frequency error of a clock changes with temperature. Similar to the calculations performed in Section 3.3, and repeated here for clarity, we define the temperature dependent frequency as

$$f(t) = (1 + \delta(t)) \cdot f_0$$

where $\delta(t)$ is the frequency error expressed as $\delta(t) = g(\kappa(t))$. Here, $g()$ is a function defined by the resonating element, e.g. a quadratic curve for a tuning fork crystal, and $\kappa(t)$ the temperature at time t .

Given two beacons from a remote node A , each containing the precise time the beacons were sent, a node B can estimate its current frequency error relative to node A 's clock using

$$\delta_Q = \frac{(t_{A2} - t_{B2}) - (t_{A1} - t_{B1})}{t_{A2} - t_{A1}}.$$

Without loss of generality, let's assume that $T = t_{A2} - t_{A1}$ and that the clock of node A is perfect. Thus, the measured frequency error becomes

$$\delta_Q = \frac{T - (t_{B2} - t_{B1})}{T} = \frac{\int_{t_{A1}}^{t_{A2}} g(\kappa(\tau)) d\tau}{T},$$

where $\int_{t_{A1}}^{t_{A2}} g(\kappa(\tau)) d\tau$ is essentially the accumulated error over the interval from t_{A1} to t_{A2} . In other words, it computes the average frequency error over the time between the two messages. Thus, assuming that the frequency error changes over time, the error the time synchronization protocol will make in terms of frequency error estimation at the time instance t_{A2} is

$$\delta_E(t_{A2}) = \delta_Q - g(\kappa(t_{A2})), \quad (4.2)$$

where δ_Q is the current drift estimate, and $g(\kappa(t_{A2}))$ the actual drift of the crystal.

This gives us the theoretical background to investigate the errors made in frequency error estimation of a time synchronization protocol. In order to verify these results, we need to simulate an actual time synchronization protocol. We reuse our modified Castalia simulator from Section 3.3.4.

Figure 4.3 summarizes the simulated result for two different nominal frequencies, $f_0 = 1kHz$ and $f_0 = 32kHz$. It shows that the simulated error of the frequency error estimates follows the two theoretic limits, once limited by the quantization accuracy given by Equation 4.1, and then hindered by the change of the frequency error over time due to changes in temperature, given by Equation 4.2. This shows that there is an optimal resynchronization time that minimizes the error in frequency error estimation. This optimum highly depends on the nominal clock frequency f_0 , and the temperature environment the node is located at.

4.4 TCTS Algorithm

The main idea behind TCTS is to measure the temperature during frequency error estimation. Once a node learned the relationship between frequency error and temperature (i.e. it found $g(\kappa)$), the node can back off and increase the time between synchronization intervals while adapting its frequency error estimation from temperature measurements alone. Thus, the protocol contains two phases: (1) calibration and (2) compensation.

TCTS itself does not make any assumptions on the network architecture, i.e., it is a technique to elongate the time between resynchronization intervals, and thus is compatible with network synchronization protocols like FTSP, GTSP, or a synchronization tree similar to TPSN or IEEE 1588. However, it is assumed that a node running TCTS, let's call it slave node, receives accurate time measurements from a remote node, called master, and that the slave node can communicate with its master. This communication is necessary, since the slave node has to let the master node know the resynchronization interval. More details on this communication will follow in the next two subsections.

4.4.1 Calibration

At initialization, the slave node does not know its current frequency error. Thus, it needs to rely on the cooperation of a master node that has knowledge of the accurate time. In order to increase the precision with which the slave estimates its current frequency error, it requests from the master node timestamped beacons with the optimal interval. The slave node can determine this interval by the knowledge of the nominal frequency of the local clock, and some knowledge about the temperature environment it expects, as we explained earlier in Section 4.3.

For each timestamp beacon, the slave node notes its current local time *time* and temperature *T*. Using this information, and a simple regression on the *time*, and time offset between the timestamp and local time, the slave node can estimate its current frequency error δ_Q and offset average *offset*. It can now store this frequency error in a calibration table containing temperature vs. frequency error estimates. Assuming that the temperature doesn't change, the slave node can now inform the master that it is calibrated, and thus the master increases the time between beacons. Subsequently, the slave node will switch into the compensation state.

4.4.2 Compensation

During compensation, the slave node regularly measures the environmental temperature. After every measurement, the slave checks the temperature vs. frequency error curve to see if it knows the frequency error for the corresponding temperature. If the table entry is missing, then the node switches back into the calibration state, informing the master to send timestamp beacons at the calibration rate. However, if the frequency error is known then the current frequency error estimate is updated using the calibration information.

After the update in frequency error, the slave node has to compensate its local offset estimation. More precisely, every time the slave node changes its frequency error estimate after a temperature measurement, it has to update its current local offset estimate using

$$\text{offset} = \text{offset} + \frac{\delta_Q + \delta'_Q}{2} \cdot (t - t') \quad (4.3)$$

where δ'_Q is the old frequency error, δ_Q the new frequency error, t' the last local compensation time, and t the current local time.

Even if the slave node is completely calibrated, i.e., if we found the frequency error estimate for every temperature possible, the master still needs to send the occasional synchronization beacon due to inaccuracies in the calibration. These occasional synchronization beacons assure that the absolute time error between the master and the slave never increases above a certain threshold. After each such beacon, the slave node calculates the absolute time error. If this error is below the required synchronization accuracy, then the slave informs the master to increase the beacon interval, in order to save energy. There are several such increase strategies possible. In our current TCTS implementation, we use an additive increase, multiplicative decrease, similar to TCP's congestion avoidance algorithm.

4.5 Simulation Results

The difficulty in comparing TCTS to a legacy time synchronization protocol is the lack of a static resynchronization period. The length of the resynchronization period during the calibration phase of TCTS is given by the nominal clock frequency f_0 and the expected temperature environment. However, TCTS has its limit. It cannot infinitely wait between resynchronization periods and still periodically resynchronizes, as explained in the last section. In addition, TCTS' average resynchronization period

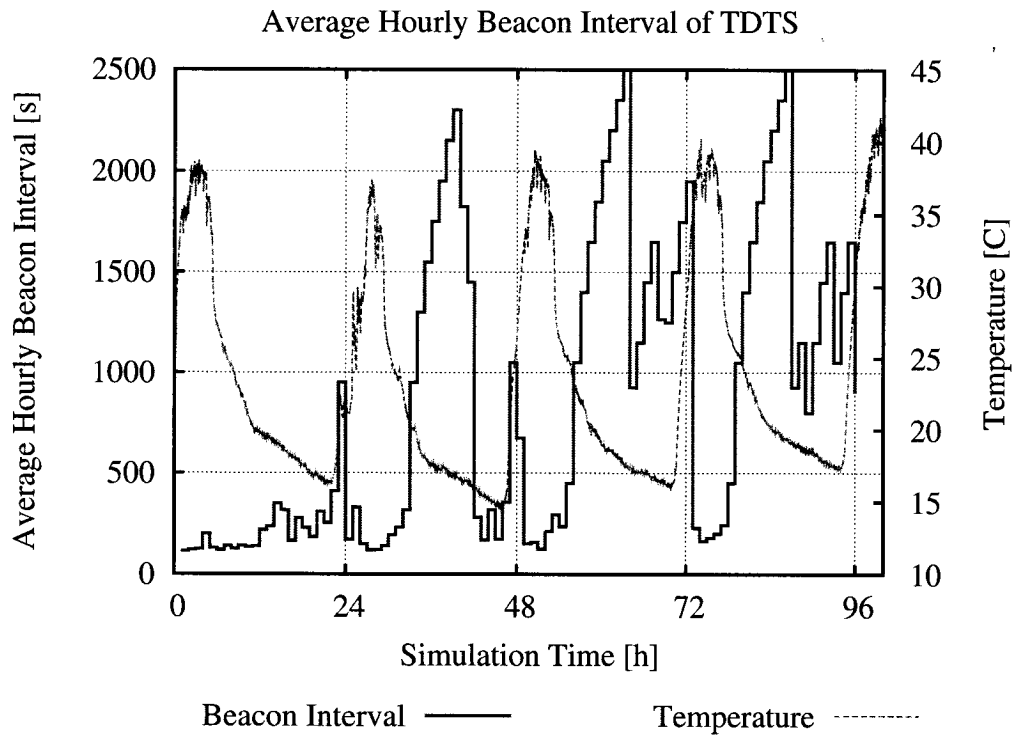


Figure 4.4: This graph shows the hourly average resynchronization time. In the first 24h, TCTS calibrates the local clock, and thus uses a very short synchronization period. Once calibrated, the resynchronization time rapidly increase and thus saves power and communication overhead.

will increase the less it has to calibrate. Figure 4.4 shows the average beacon interval per hour. We can clearly see that during the first 24 hours of the simulation, TCTS is primarily in calibration mode, since it has never encountered the environmental temperatures. However, as soon as TCTS can rely on the calibration information, the average resynchronization period rapidly increases.

Compared to FTSP, the overall accuracy of TCTS is very comparable. On average, through the whole 96h simulation, TCTS has an average beacon interval of 329 seconds and 95% of the errors lie within ± 4 tics of a 32kHz clock. FTSP with a static resynchronization rate of $T = 400$ seconds achieves 95% of the errors within ± 3.2 tics. However, by examining TCTS after its initial calibration period, TCTS' average beacon interval increases rapidly, without impacting its overall accuracy. Disregarding the first 24h of the simulation, the average resynchronization period of TCTS increases to $T = 439$ seconds, after 48h it becomes $T = 590$ seconds, and after 60 hours even $T = 730$ seconds. At $T = 700s$ FTSP's accuracy drops to 95% of the errors within ± 11 tics, whereas TCTS maintains the ± 4 tics accuracy. Figure 4.5 illustrates this behavior further.

4.5.1 Robustness Towards Communication Loss

One further advantage of TCTS over a legacy time synchronization protocol is in scenarios where communication either gets lost or severely impacted. Examples of such scenarios are heavy environmental conditions (snow on antenna, heavy rain, etc.), mobility (moving outside radio range of peers), or covert military operations where radio silence is crucial. In these scenarios, TCTS switches automatically into a TCXO mode, where the frequency of the local clock gets regularly updated. In contrast, legacy synchronization protocols will have to rely on the last frequency error measured. Figure 4.6 compares the synchronization accuracy of TCTS with FTSP if the synchronization

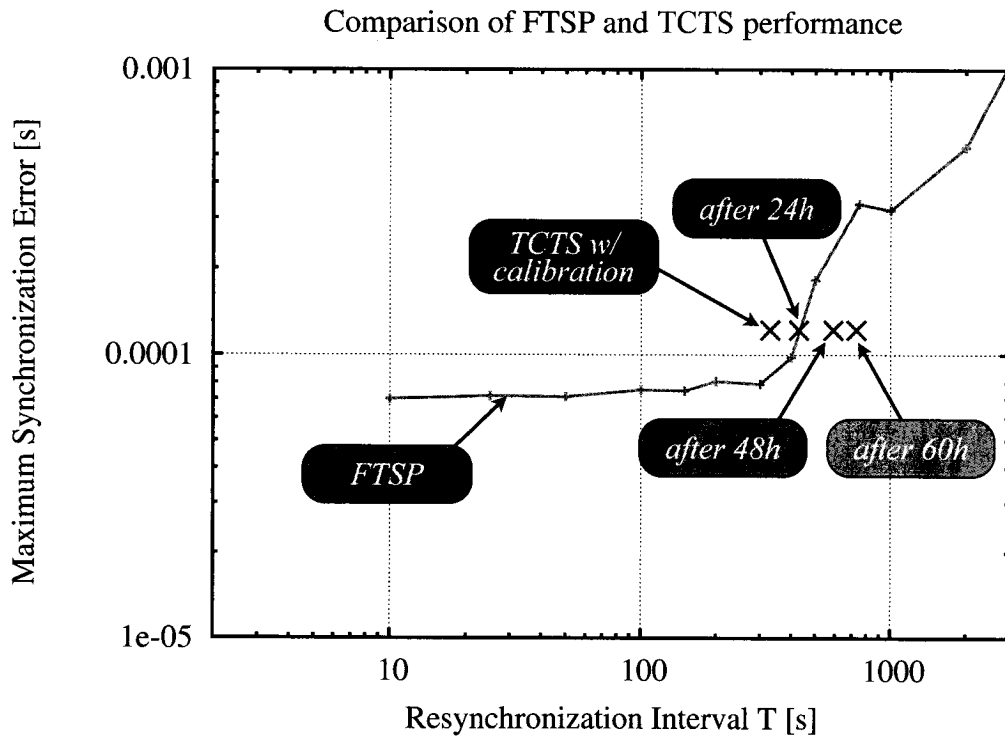


Figure 4.5: If one ignores the one-time calibration overhead, TCTS quickly outperforms FTSP in terms of accuracy with long resynchronization intervals. This is for a 32kHz tuning fork crystal.

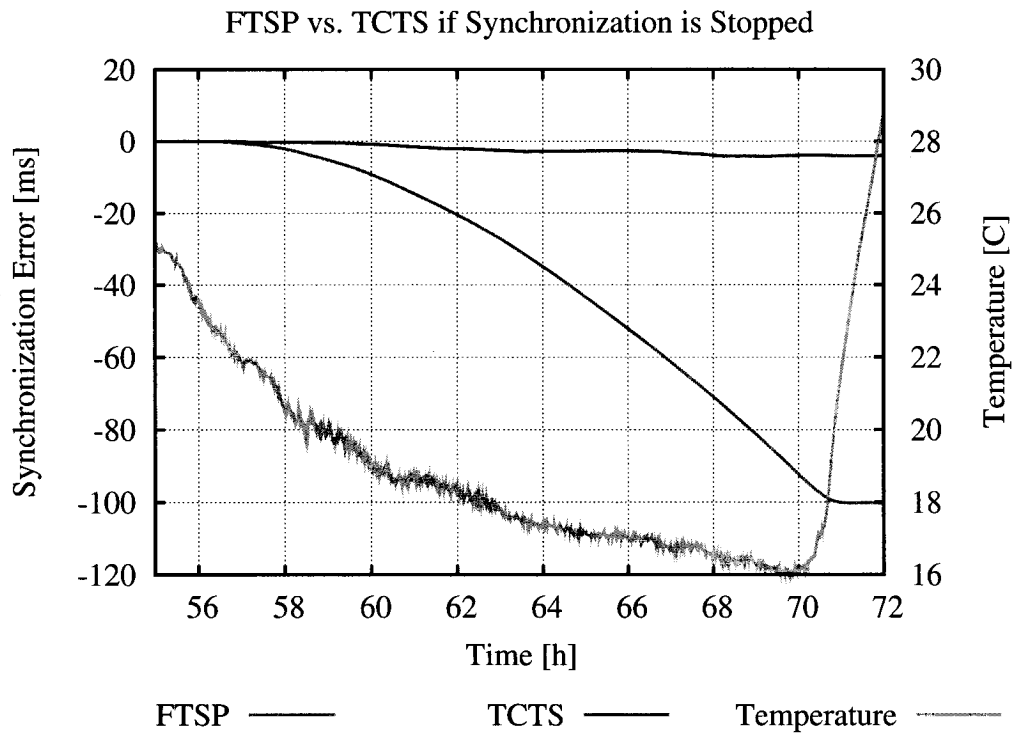


Figure 4.6: Illustration of what happens if time synchronization is stopped. We can clearly see how FTSP's performance worsens once the temperature starts to change, whereas TCTS keeps the synchronization accuracy at a much higher level.

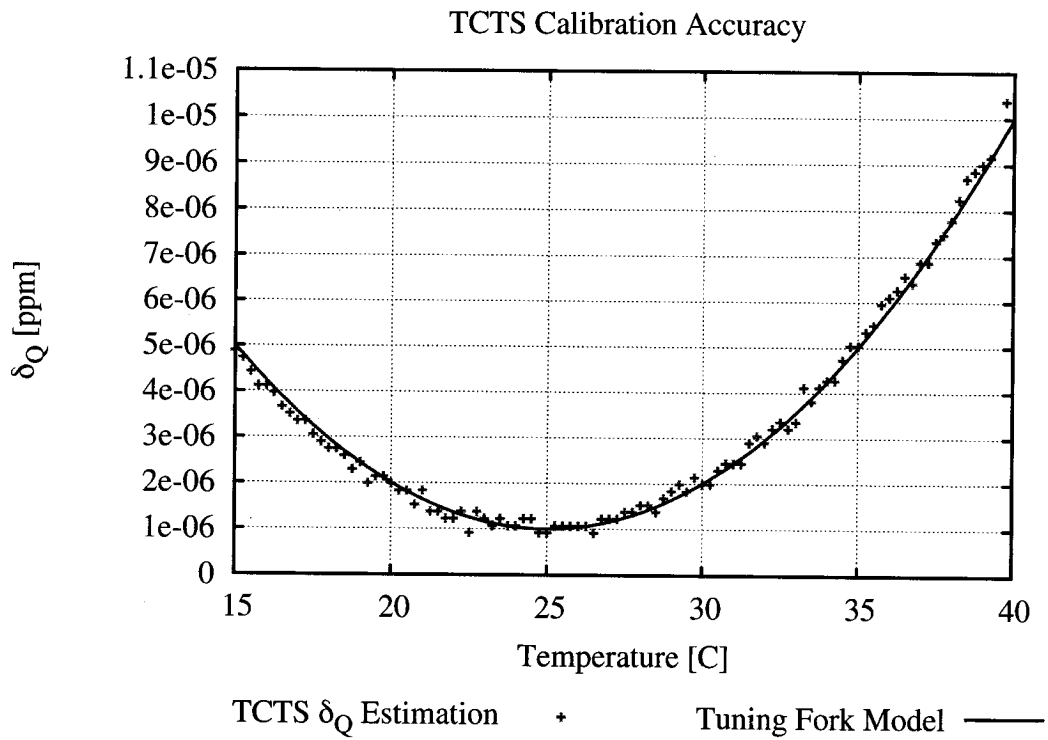


Figure 4.7: TCTS estimates the frequency drift function $g(\kappa)$ for the current temperature, and stores it in a calibration table. In our simulation, $g(\kappa)$ is a quadratic curve, as found in the tuning fork crystals. Even with only a 0.25C temperature resolution, the performance of TCTS is extremely high.

process gets stopped after 56 hours. As soon as the temperature starts to change significantly from the last temperature observed during the last synchronization interval, FTSP starts to lose accuracy, while TCTS can keep its accuracy at a very precise level. Over the 16 hours where no resynchronization occurs, TCTS accumulates only 4ms of time synchronization error. This corresponds to a clock stability of $< 0.07\text{ppm}$! Figure 4.7 shows the calibration information TCTS collected during this experiment, and compares it to the actual clock model fed into the simulator itself. This plot shows that a time synchronization protocol can successfully temperature calibrate a local clock oscillator, and thus removes the factory calibration step necessary for a TCXO.

4.6 Summary

Temperature Compensated Time Synchronization is a first step towards an automatic temperature calibration of a local clock. This chapter showed that TCTS can outperform a standard time synchronization protocol by exploiting local temperature information to increase the synchronization intervals. This increase results in overall network power savings, since fewer synchronization messages have to be transmitted. In addition, TCTS improves the robustness of an embedded device in case of communication difficulties.

This chapter introduced a first implementation of TCTS. For this implementation, FTSP was chosen as the host synchronization protocol, since it is one of the most widely used synchronization protocols in the sensor network community, and its implementation is readily available. Overall, this chapter gave insight into time synchronization by solving a problem introduced by the underlying hardware a synchronization protocol relies on. The next chapter will go one step further and improve said hardware to minimize power consumption, while improving time resolution.

CHAPTER 5

Architectural Support for Increased Time Accuracy

5.1 Introduction

Wireless sensor networks, due to their mission-critical applications and wire-free nature, present one of the most extreme power budget design challenges in the field of electronics. Improvements in timing can reduce the energy required to operate a sensor network in three ways: (1) an improvement in timing synchrony among network members prevents unnecessary radio on-time, (2) it allows beam-forming and other collaborative techniques to displace larger more energy intensive sensory systems bringing new, previously unachievable capabilities to the network nodes, and (3) improved timing precision may eliminate the need for continuous sampling, through a signal processing technique known as Compressed Sensing, saving power in the sensory and acquisition subsystems.

The average power consumption in sensor network research platforms has been steadily declining for many years, mainly due to refinements in wireless communication protocols. Wireless communication is the most power hungry subsystem of a wireless sensor network node, largely due to the high cost of idle listening, e.g. the necessity of guard bands to offset clock inaccuracies [DCS07].

Accurate time on an embedded system is not only needed for more precise communication timing, new applications of sensor networks in industrial settings push the limits of the currently available technology. For example, IEEE 1588 Precision Timing

Protocol was developed because NTP, the de facto Internet time synchronization standard, was not precise enough for industrial measurement and control systems. However, IEEE 1588 requires sub-microsecond accuracies, a level of precision not yet attained by any wireless synchronization protocol [MKS04, SW09]. To achieve sub-microsecond accuracy, high resolution becomes as important as high stability. This requires high speed clock signals in the MHz range, translating into high power consumption due to high frequency.

Recent advances in sampling, called Compressive Sampling [CW07], suggest huge power savings. The theory shows that one can reconstruct a sparse signal with high probability while just randomly sampling it¹. This removes the necessity of constantly acquiring samples at a regular constant time interval, potentially saving a significant amount of energy. However, it introduces the problem that high precision is needed to know the exact time between the random samples, or else reconstruction becomes error prone.

5.1.1 Architectural Requirements

One key observation is that providing accurate time to an embedded system does not necessitate a stable clock frequency. A globally synchronized high-resolution alarm or real time clock is enough. There are three main requirements that need to be addressed, in order to provide such a service to a system:

1. Low-Power, High-Resolution Local Clocks

In general embedded systems, local clocks get sourced from a quartz crystal with frequencies ranging from a couple of kilo Hertz, to several tens to hundreds of Mega Hertz. One important thing to remember is that the higher the frequency

¹The author is aware that this is a gross simplification of the Compressive Sampling theory. For more information on CS, please see [CDS98, CRT06]

of a crystal, and thus its associated hardware, the higher the power consumption. More precisely, $P \sim P_0 + (C \cdot V^2 \cdot f)$, where P_0 is the power lost due to leakage, C is the effective load capacitance, V the driving voltage, and f_0 the frequency at which the circuit runs. For that reason, low-power real time clocks are sourced from a low-frequency crystal, usually a 32 kHz Tuning Fork crystal. Additionally, during system sleep, high frequency clock signals and crystals get turned off in order to save energy. Therefore, the system will lose high resolution time.

2. **Efficient and Robust Time Synchronization Protocols**

Many different time synchronization protocols have been developed for wireless sensor networks [MKS04, SW09]. An extension to make them more robust against temperature changes was presented in the last chapter. Even simple primitives, such as the 'Estimated Time on Arrival' [KDL06] have been investigated. While all these protocols work extremely well, they all rely on a high frequency clock in order to get to their claimed $< 10\mu s$ time accuracies. Thus, their hardware cannot achieve a low-power sleep state, or else the systems would lose time, and thus synchronization. The problem does not lie within the time synchronization protocol itself, but with the hardware that they are running on.

3. **Hardware Timer Units that Support Long Sleep Intervals at Minimal Power**

Many of the popular sensor network platforms contain timer units with counters of different bit lengths. However, often the counters are not long enough and thus have to be extended in software by tracking their overflow bits. For example, the popular Texas Instrument MSP430 has several 16-bit timer units. If a 32kHz frequency clock is connected to one of them, then the microcontroller will have to wake up every 2 seconds to track its overflow. This time could be extended by using prescalers existing in the timer unit, though the system would lose significant time resolution, since dynamic switching between different clock

frequencies implies losing track of time.

In this chapter, we will address all three requirements and give a solution to provide global synchronized real time clocks with high time resolution at very low power consumption. We will first introduce a formalization of the power consumed by the Flooding Time Synchronization Protocol (FTSP) as a representative example. Using this knowledge, we investigate how a dual-crystal system, containing a high and a low frequency crystal, can optimize the power consumption of an embedded system during sleep times, while still providing high resolution time during the active state, and thus greatly improves the power consumption of a legacy single crystal system. We conclude this chapter by presenting a first prototype of the High-Low Frequency Timer Unit (HLTimer) implementation on an Actel Igloo FPGA. We will show its power performance, and how it could be interfaced in future sensor network platforms to provide hardware architectural support for existing microcontrollers.

5.2 The Cost of Time Synchronization

The distributed nature of time synchronization protocols makes it difficult to track its power consumption. An action on one node can have implications on another. For example, if a node sends out a timestamped message, every node in radio range will receive it, no matter if it is useful to that node or not. Recent advances in the sensor network community [FDL08] make it for the first time possible to track such activity across nodes. The Quanto system is a network-wide time and energy profiler for embedded networked devices. By tracking the internal power state of a device and its associated node activity, and adding this activity information to radio messages, Quanto can successfully track activity migration from node to node.

While measuring activity duration on a single node is possible through toggling I/O

pins on the hardware, and then externally measuring the time between I/O toggles, this becomes quickly intractable considering the distributed nature of time synchronization protocols. In addition, the more subsystems one wants to observe, the more I/O pins are necessary. Quanto solves this problem in software by adding software instructions into the code. No additional hardware is necessary, and thus observation of large networks of nodes become feasible. While in our later example we use Quanto on 3 nodes only, it is technically possible to extend these measurements over a large network with several hops, to provide accurate accountability of how much resources a certain distributed algorithm consumes. In our case, this is not necessary since we can get all the measurements for our analysis over a 1-hop network.

Quanto leaves the problem open on how to cross correlate activities across nodes since the logs are collected using local time information at each node, i.e., while activities migrate from node to node, the timestamps of these activities are in terms of each nodes local time. Thus, time synchronization becomes an integral part of Quanto to show the activities on a common time line. We extended our Quanto testbed with the Flooding Time Synchronization Protocol (FTSP), and at the same time use Quanto to track the state of FTSP across nodes. The objective is to find out how much time each node spends to treat messages sent out by FTSP in order to synchronize the network.

5.2.1 The Flooding Time Synchronization Protocol

The strength of FTSP lies in its simplicity. By default, every node broadcasts its estimate of the global time every T seconds. Upon reception of such a message, a neighboring node timestamps the message using its local timebase, and decides if it records this global time, local time pair in its time log or not. The deciding factor is a global sequence number that gets transmitted in every message. If the sequence number in a received message is lower or the same as the one the node received in the last message,

then it throws out the time pair. If it is higher, then it is a new time estimate, and the node keeps the message in its log. After successful reception of a new time pair, the node runs a linear regression on the last N time pairs to extract its local clock frequency error and time offset between local and global time. These estimates can later be used to translate any local time into global time, and vice-versa.

Another important part of FTSP is its root election mechanism. The root node is the only node in the network that increases the global sequence number in the time synchronization messages. Thus, the whole network will eventually synchronize to this nodes time. By default, a node calling itself root will give up that role if it hears from an other node with a lower node id. It turns out that this very simple rule works very efficiently, and within a short time a network elects the node with the lowest node id as the synchronization root.

Given this simple algorithm, there are two important events of interest. The first event is when a node receives a time synchronization message, and it rejects it because of an old sequence number. The second event happens when the message gets treated and the algorithm runs the linear regression in order to update its estimates.

Figure 5.1 and 5.2 show the activity log retrieved from 3 nodes using Quanto representing the two different events of interest. For each node, we tracked the activity on the CPU, as well as the node radio (Texas Instrument's CC2420). The different colors represent the different activities that Quanto tracks. The most important one is activity "X:Ftsp", where "X" represents the node id. This activity represents the FTSP algorithm running on node "X". Figure 5.1 shows how the activity "1:Ftsp" originates at node 1, and then, after reception of the packet at the other two nodes, migrates to their CPUs. In this case, the packet originated from the synchronization root, and thus both receivers keep the message and calculated the linear regression on their time pair logs. In contrast, Figure 5.2 depicts the case when the message gets quickly discarded

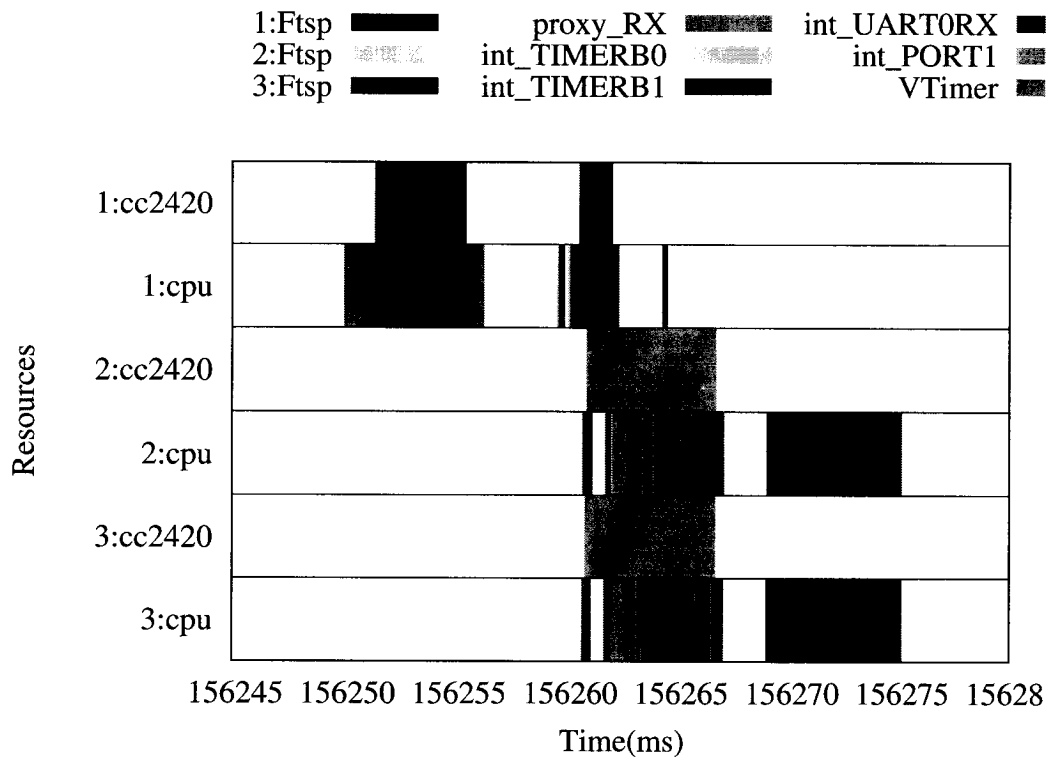


Figure 5.1: Time synchronized Quanto Activity log of three nodes participating in time synchronization. Node 1 sent out a timestamped broadcast message. Node 2 and 3 receive the message and process it, since Node 1 is their synchronization root node.

after reception, since its sequence number is too old.

Using these two figures, we can extract four time intervals, the time to transmit a message T_{TX} , the time to receive a message T_{RX} , the time to calculate the linear regression T_{Calc} , and the time to throw out a message T_{NCalc} . Table 5.1 summarizes the result. These four time durations build the basis for a simplified time synchronization power model, that we introduce in the next section.

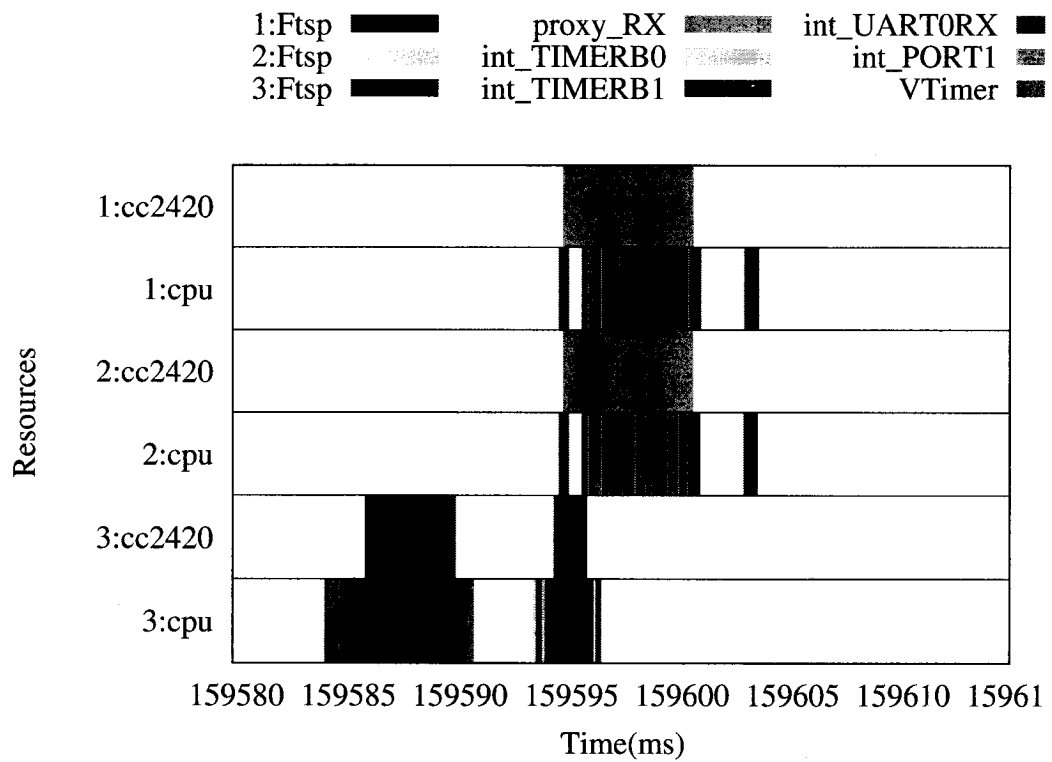


Figure 5.2: Time synchronized Quanto Activity log of three nodes participating in time synchronization. Node 3 sent out a timestamped broadcast message. However, since 3 is not the synchronization parent of node 1 or 2, they both just receive and then discard it.

5.2.2 Time Synchronization Power Model

One can differentiate two modes of time synchronization in order to calculate its power consumption; Single Hop, and Multi-Hop cases. The single hop case is easier to understand, and therefore we will start with this case, before we generalize it to the multi-hop network. The goal of our model is to find out the power vs. frequency stability curve that a clocking subsystem has to have, in order to improve over a regular 32kHz quartz crystal.

T_{TX}	T_{RX}	T_{Calc}	T_{NCalc}
11ms	7.5ms	7.5ms	3ms

Table 5.1: Different Time Constants

Case 1: Single Hop Network In the single hop case, where every node can hear each other, the average power consumption of a node can be calculated as follows.

Root Node The root node does not process other node's broadcast messages. However, it still has to receive them because potentially, there could be a node with a lower node ID and then the root node would give up its role to that node. Thus

$$P_{avg} = \frac{P_{TX} \cdot T_{TX} + (N - 1) \cdot (P_{RX} \cdot T_{RX} + P_{Proc} \cdot T_{NCalc})}{T} + P_{Clk}, \quad (5.1)$$

where P_{TX} is the power consumed by the system during transmit, T_{TX} the time it takes to send the message, N the total number of nodes in the network, P_{Proc} the power during packet processing, T_{NCalc} the time to process a message and reject it, T the time interval between resynchronization attempts, and P_{Clk} is the average power consumption of the clocking system.

Other Nodes All the other nodes in the network will have to process messages coming from the root node, and reject the ones from the other nodes. Thus,

$$P_{avg} = \frac{P_{TX} \cdot T_{TX} + (N - 1) \cdot P_{RX} \cdot T_{RX} + P_{Proc} \cdot (T_{Calc} + (N - 2) \cdot T_{NCalc})}{T} + P_{Clk}, \quad (5.2)$$

where T_{Calc} is the time it takes to process a message and run the time synchronization algorithm using the message.

Case 2: Multi-Hop Case The main difference between the multi-hop and single-hop network is that N becomes the number of neighbors for each individual node. Thus, the node with the lowest node id in the network will have an average power consumption described by Equation 5.1. Every other node in the network will have an average power consumption described by Equation 5.3. However, for every node, N is the number of direct neighbors in radio range, and since this number is different for every node, every node will have a different average power consumption.

5.2.3 Power – Frequency Stability Equilibrium

One system variable in Equations 5.1 and 5.3 is the time between resynchronization requests T . This parameter can be freely chosen by the system developer. However, in Chapter 3 we showed that this parameter is intrinsically linked to the accuracy a synchronization protocol can achieve. Thus, given a specific maximum synchronization error ϵ , we can estimate the resynchronization interval T to achieve this level of accuracy.

For a general clocking system, $\delta \cdot T \leq \epsilon$ must hold, where δ is the clock stability over the resynchronization interval T . We now can compare a new hypothetical clocking system to a regular 32kHz system by finding P'_{avg} such that

$$P_{avg} \geq P'_{avg}, \quad (5.3)$$

where P_{avg} is the average power consumption of the legacy 32kHz system, and P'_{avg} the average power consumption of the new hypothetical clocking system.

First, let's define the energy consumed purely by the resynchronization process:

$$E = P_{TX} \cdot T_{TX} + (N - 1) \cdot P_{RX} \cdot T_{RX} + P_{Proc} \cdot T_{Calc} + (N - 2) \cdot P_{Proc} \cdot T_{NCalc} \quad (5.4)$$

We can rewrite Equation 5.3 and solve for the power the new system clock can

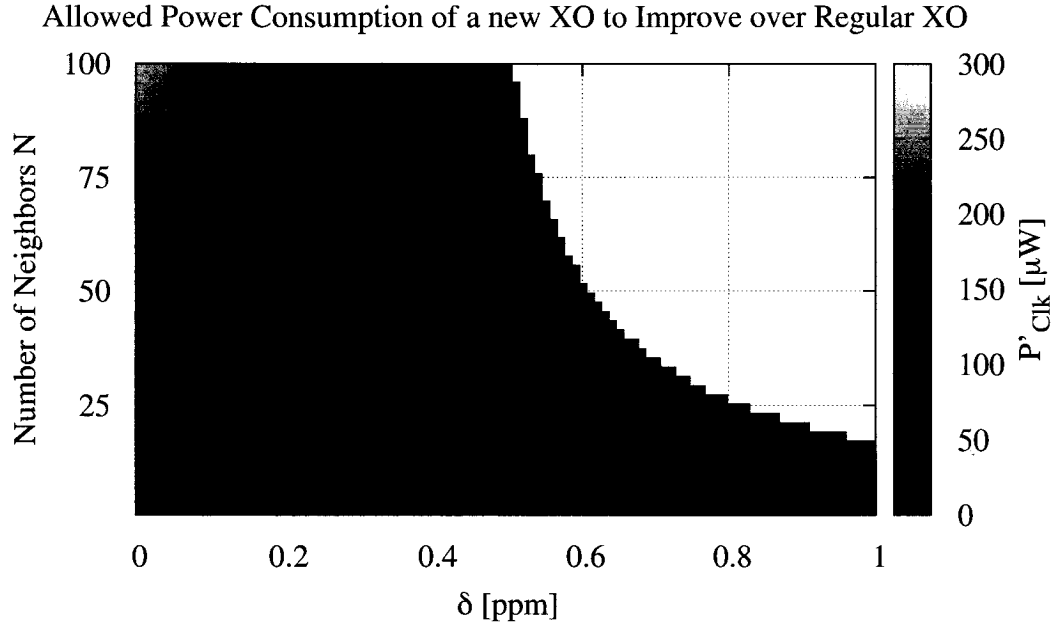


Figure 5.3: Result of Equation 5.6 using the Epic platform as an example. We can observe that for dense networks ($N > 75$ nodes), a TCXO with a power consumption of around $200\mu W$ starts to be a viable solution to improve the power consumption.

consume as P'_{Clk}

$$\begin{aligned}
 P'_{Clk} &\leq P_{avg} - \frac{E}{\epsilon/\delta} \\
 &\leq \frac{E \cdot (\epsilon - T \cdot \delta)}{T \cdot \epsilon} + P_{Clk}.
 \end{aligned} \tag{5.5}$$

5.2.4 Concrete Example using the Epic Sensor Network Platform

In Chapter 3 we developed a model relating the time synchronization accuracy to the resynchronization interval, give a certain temperature environment. From these results, we can show that for an accuracy of $\epsilon \leq 0.1ms$, a standard 32kHz crystal using the

P_{Cik}	P_{TX}	P_{RX}	P_{Proc}
$56\mu W$	59.1 mW	66.7 mW	1.65 mW

Table 5.2: Different Average Power Consumptions

Flooding Time Synchronization Protocol has to resynchronize every $T \leq 250s$. With this number, well published power measurements of the Epic platform (see [DTJ08], summarized in Table 5.2) and the timing numbers obtain in the last section using Figures 5.1 and 5.2, summarized in Table 5.1, we can solve Equation 5.6 and find the region in which a more stable clock has to operate, in order to improve the power consumption of a regular quartz crystal.

Figure 5.3 illustrates the different power regions. An interesting observation is that the more neighbors N a node has, the more energy that node can spend on having a more stable clock. However, this is only valid if the stability of the clocks between resynchronization attempts is smaller than 0.4 ppm. Above that, the more neighbors there are, the worse it is for the node since it has to transmit too many resynchronization messages. However, if the stability increases, the more neighbors there are, the more energy is spent on resynchronization given a fixed T of 250s. Thus, there is more energy available for the clocking system.

5.3 Exploiting Dual-Clock Hardware Designs for Increased Time Resolution

One major problem in high-accuracy time synchronization in embedded systems is the need for high-frequency clock signals in order to achieve high-resolution time. A simple, though novel, approach to this problem is to use two clocks, one high frequency, one low frequency, to track time. During system sleep, only the low-frequency clock

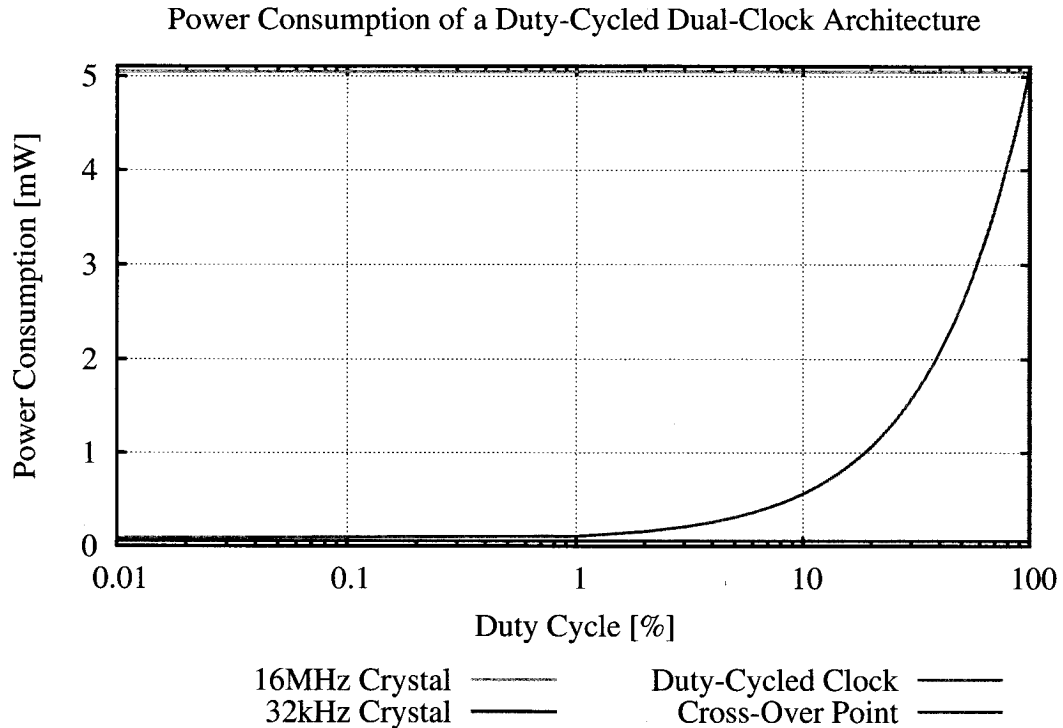


Figure 5.4: Power consumption of a dual-clock hardware design given a specific duty-cycle of the radio, assuming the slow clock consumes $56\mu W$ and the fast clock $660\mu W$.

is on and tracks time. However, during active time, the high frequency clock is used to interpolate between the low frequency's clock ticks, and thus keep high resolution time. The benefits of such a system is that during system sleep, no high frequency clock signal is using precious energy.

This advantage of a high and low frequency clock becomes even more evident when considering the low duty-cycles observed in sensor network applications. Figure 5.4 shows the average power consumption of a dual-clock system. For this figure, we assumed that the low frequency clock consumes around $56\mu W$, while the high frequency clock consumes $5mW$. We can see that even at a very high duty-cycle of 10%, the clock

system will consume only 1/10th of the high frequency's active clock consumption, while still keeping high time resolution.

The key technology behind this system consists of correctly wiring two timer units, each sourced by either the low, or the high frequency clock. More precisely, the low frequency clock has to be connected to one of the capture units of the high frequency timer unit. This will allow the microcontroller to keep track of the phase of the low frequency clock. By tracking the phase of the low frequency clock, during an external capture event, both the high and the low frequency timer capture their content. Using some multiplication and modulo operations, we can now find the precise time in the high frequency clock's resolution.

5.3.1 Sub- μ Second Time Synchronization Using the Dual-Clock Approach

To show the effectiveness of the dual-clock approach, we implemented it in TinyOS using the Epic platform. The hardware modifications added an external 8MHz clock to the second clock input, thus allowing accurate timing at a high frequency. Both timers of the MSP430F1611 were used, TimerB connected to the already existing 32kHz crystal, and TimerA to the new 8MHz clock signal. We modified FTSP and the CC2420 driver to allow tracking of the phase of the 32kHz signal. Thus, FTSP can easily go from the 32kHz clock signals to a higher rate accuracy by multiplying the 32kHz counters by 244 and adding the phase offset.

We deployed a network of 5 of these modified nodes, each running FTSP in low power listening mode, i.e., the nodes are duty-cycling. During their sleep states, the nodes solely rely on the 32kHz crystal, while the 8MHz clock is turned off. When the node wakes up, the 8MHz clock is turned on too to provide high-resolution time. The FTSP resynchronization rate was set to 30 seconds. In addition, a 6th node, not running FTSP, sent a regular beacon. This beacon was received and timestamped by the 5 nodes

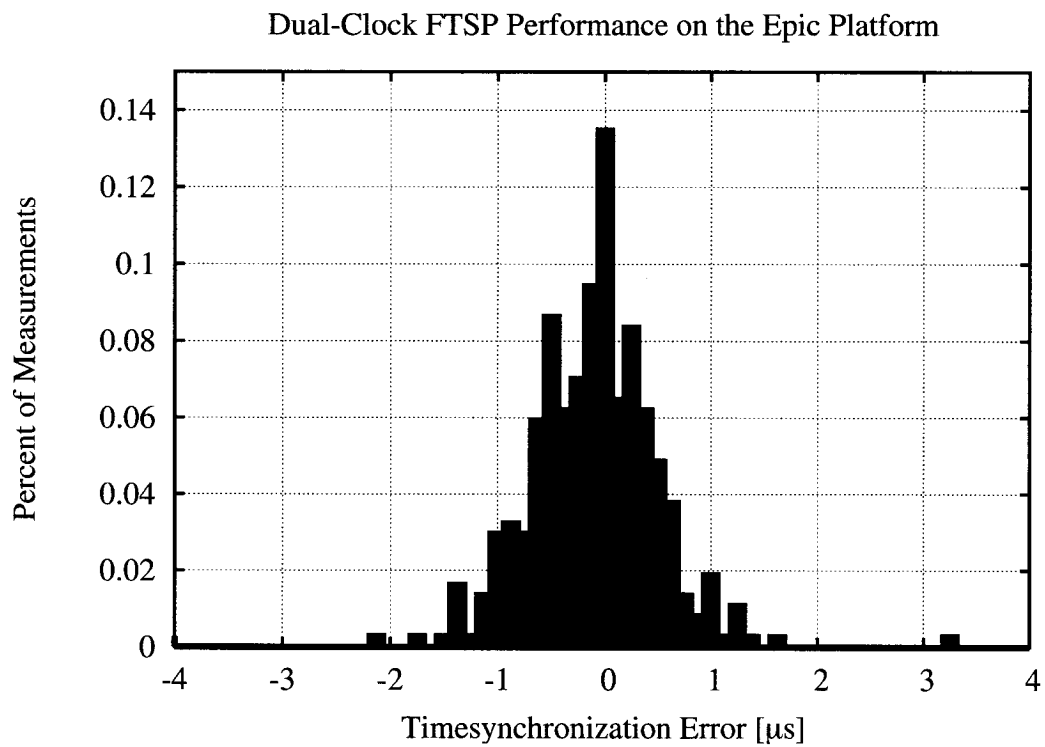


Figure 5.5: Performance of a dual-clock enhanced FTSP implementation. The hardware contains one 8MHz and one 32kHz crystal. The radio chip is a TI CC2420, using the start of frame delimiter (SFD) of the 802.15.4 messages for time stamping. On average, the accuracy is well below 1 μs .

simultaneously. The timestamps were then transmitted to a basestation computer which calculated the differences in timestamp accuracies.

Figure 5.5 shows the effective accuracy of the dual-clock FTSP implementation. It shows that the accuracy is well below $1\mu\text{s}$, while still using the TinyOS's Low Power Listening capabilities. During system sleep, the high frequency clock is turned off, and thus doesn't consume any energy, while during active time, the high frequency clock is turned on, and high accuracy time synchronization can be achieved.

This is fundamentally different from what FTSP was achieving before. While in the first publications [MKS04] FTSP did achieve an accuracy of a couple of μs , this was only possible by keeping the high frequency clock active at all times. Thus, the platform lost the ability of a low-power sleep mode and batteries drained very fast. In our implementation, the system will still sleep at its minimum power possible, and thus long lasting battery life is to be expected.

5.4 A Power-Aware Timer Module, the HLTimer

One negative point of our dual-crystal, sub- μSecond synchronization implementation is that it uses both timer units available on the Texas Instrument MSP430F1611. This leaves no timer available for any triggered ADC conversion or other tasks relying on its own timer. In addition, the use of two full timer units is quite a waste of hardware, since only a small sub-part of the timer unit is used. Thus, we implemented a new High-Low Timer (HLTTimer) unit on an Actel Igloo FPGA that leverages the advantages of a dual-clock system.

Figure 5.6 shows a simplified view of the major blocks contributing to the HLTimer's VHDL implementation. The core contains two counting registers, and one 16-bit counter. Two clock signals, the low frequency (LCLK) and the high frequency (HCLK)

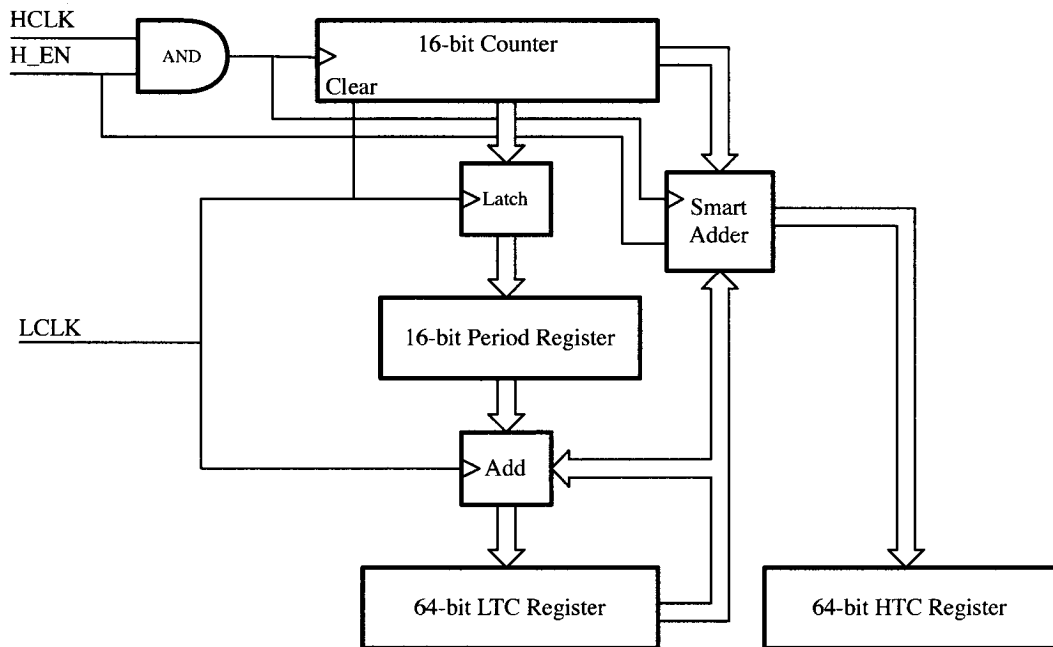


Figure 5.6: This is a simplified block diagram of the main HLTimer components. The main counter is a 16-bit counter that is fed by the high frequency clock signal HCLK. The low frequency clock LCLK drives the rest of the logic and increments the different counting registers LTC and HTC based on the HLTimer's algorithm.

drive the respective parts of the implementation. An enable signal (H_EN) indicates to the timer unit if the HCLK is active or not, and switches the time resolution of the counting register HTC from high to low.

The key to the HLTimer implementation is a period counter register. This register stores the value of the 16-bit counter at every LCLK clock tick. In addition, at that instance the 16-bit counter is reset to 0, and thus starts counting again. This number indicates how many HCLK ticks there are for each LCLK, and thus indicates the increment by which the counting register should be incremented at each LCLK clock tick while the HCLK clock is turned off, i.e.,

$$\text{increment} = \left\lfloor \frac{f_H}{f_L} \right\rfloor.$$

In summary, at every LCLK clock tick, an adder (Add) adds the period register to the LTC counting register. This register keeps track of time in a low-resolution mode. At the same time, a Smart Adder, depending on the H_EN signal, either adds the 16-bit counter value to the LTC counting register, and stores it in the HTC counting register, or if the HCLK is turned off just stores the LTC counting register directly in the HTC counting register. Thus, the HTC register eventually switches between a low and high resolution time, without ever losing time itself.

An important component of a timer unit is the capture and a compare unit. But neither of these units is different from a regular timer unit already found in microcontrollers. The only difference is that they don't operate on the counter that gets directly incremented by the clock signal, but on the derived HTC counting register instead. The capture unit will upon reception of a trigger signal copy the HTC register into a timer capture register TCCR, and thus store the value for a microcontroller, that can read it at convenience. The only difference that a microcontroller might want to know is if the value corresponds to a high, or low resolution time. This could easily be indicated by one of the bits in the field, or by a special additional configuration register, since the

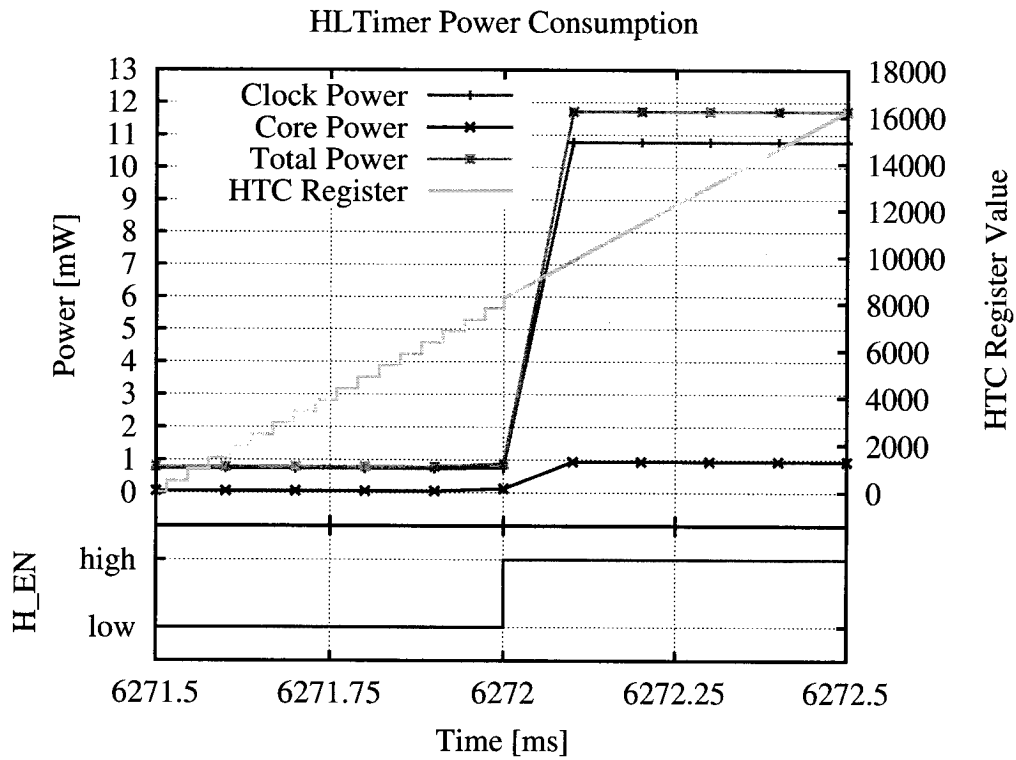


Figure 5.7: The power consumption of the HLTimer abruptly increases if the resolution is set to high. The effect is that the counting in the HTC register becomes much finer, and thus high resolution time becomes available.

HLTimer always knows if it is in low or high resolution mode.

The current implementation of the HLTimer consists of the counting mechanism illustrated in Figure 5.6 and a single capture unit that allows to capture the HTC register if the signal on an input line goes high. We tested the implementation on an Actel IGLOO AGL600 FPGA. This particular FPGA has 600'000 system gates and 13824 core cells (D-Flip Flops). Our implementation consumes only 868 core cells, or a mere 6%. This shows that the HLTimer could easily fit onto a smaller version of the IGLOO, lowering the static power consumption even further.

Even with the larger FPGA, the system performs exceptionally well. The FPGA core consumes only $42.8\mu\text{A}$ while sleeping and a 32kHz clock on, while consuming $767\mu\text{A}$ during active time when a 16MHz clock drives the circuit. Note that the quiescent current of the core itself is $36\mu\text{A}$ during flash freeze mode, i.e., when nothing happens on the FPGA itself.

Figure 5.7 depicts the measured power consumption of our prototype implementation and illustrates the time accuracy during a change of time resolution. The *H_EN* signal indicates to the core if the high frequency clock should be on, or off. Figure 5.7 also shows the power breakdown between FPGA core, and clocks. One can observe that the clocks consume the vast majority of the power, especially when the high frequency clock is active (11mW compared to 1mW for the core).

5.5 Summary

This chapter explored the link between time synchronization accuracy and power consumption of an embedded system. We developed a model to accurately determine the power consumption of a time synchronization protocol, using time synchronized logs of Quanto, an activity tracking system for sensor networks. We developed boundary conditions for clocking systems and its requirements on power consumption and clock stability. Using a popular sensor network platform as example, the Epic module, we showed that the more neighboring nodes there are to a node, the more gain we can have by providing a more stable clock.

Exploiting several small hardware modifications, and a smart way of exploiting a dual-clock system, we showed how sub- μsecond time synchronization is possible without spending the tremendous energy a high-frequency clock needs. The main idea behind the dual-clock system is to use a low-frequency clock signal to keep absolute

time during system sleep, and a high-frequency clock signal during system active time to interpolate between the low frequency ticks, to provide high-resolution time. Using such a system, we demonstrated an actual dual-clock implementation using a modified version of TinyOS 2.x and the Flooding Time Synchronization Protocol (FTSP). Our implementation exceeded any preceding sensor network synchronization protocol accuracy by a factor of 5, while still keeping the average node power consumption to a minimum.

However, this system is hard to implement on regular microcontrollers and consumes too many system resources. Thus, we developed the High-Low Timer (HLTimer) unit on an FPGA. With this implementation, we showed how the HLTimer could be integrated into a microcontroller, and what the major power gains could be, if a microcontroller manufacturer would decide to add such a unit as a peripheral.

CHAPTER 6

Discussion & Conclusion

Despite the demand for better time synchronization accuracy and the quest for ever lower power consumption, the actual source of time in embedded systems has seen less innovation than other parts of the systems. While clock source design is still a very active field, and the clocking network within chips provides many challenges to the chip designer, innovation in timer units to provide accurate time to software running on a microcontroller has been missing. This dissertation advocates the need for a new timer design and developed theory to better understand the link between environmental temperature changes, the biggest factor of change in clock frequency, and time synchronization accuracy. It culminates in the design of a temperature compensation technique for time synchronization protocols, and the development of a next-generation timing unit that allows a duty-cycled embedded system to achieve high resolution time during the active state, while still consuming minimal power during sleep. We now conclude by discussing how the latter two contributions, temperature compensation and high-low timers, would interact, and examine the benefits of this new approach.

6.1 Low-Power, High-Accuracy Time in Embedded Systems

Time in embedded systems is one of the most important services. It becomes even more important in networked embedded systems which have to communicate with other devices. Here, time is often used to either tell a device when to talk to whom, or

to timestamp sensor measurements in order to compensate for transmission latencies. While improvements in time services have been made over the years, most of them stayed within their isolated domain:

- While compensating for temperature induced frequency drift in crystal oscillators has been done for years in TCXOs, their dollar and energy cost is often too high to make them viable solutions for the lowest tiers in sensor networks. Unfortunately, TCXO designers can only optimize their designs within their domain, not crossing and taking advantage of other embedded system services, like communication or time synchronization. While some TCXOs and other clocking devices can accept a 1pps signal for recalibration and compensation for aging, this signal assumes that there is an other clocking system available that is even more precise than the TCXO, thus, adding even more cost to the platform itself.
- Time synchronization made leaps and bounds over the last couple of years, ever improving its accuracy. While carefully taking into account the errors introduced by the communication layer, timestamping accuracies, lost messages, interrupt latencies, the research largely ignored the cost and errors introduced by the local clock, often abstracting it to a clock signal with constant frequency error over the time intervals of interest.
- The timer unit in a microcontroller is responsible for providing time to the software running on the core. It takes a clock signal as an input, and simply counts the number of ticks in it. In addition to that, it usually provides options to capture the counter for a certain event, or produce an output signal when the counter reaches certain time values. In more advanced timer units, one sometimes gets the possibility to chain several units together, and thus doubling the width of the counting register. While this can greatly decrease the interrupt burden on a microcontroller, it is as far as innovation goes in timer units.

This dissertation showed the importance of cross-layer optimization in the case of time accuracy and its power consumption as follows.

- The Dual Crystal Compensated Crystal Timer was our first attempt at improving clock stability for embedded systems, not by trying to generate a stable clock signal, but by using smart software algorithms compensating for the frequency error. This development gave us insight into the critical interaction of the change in environmental temperature, frequency error, and timer units. However, we quickly realized that we can improve upon this by incorporating communication capabilities.
- Temperature Compensated Time Synchronization is the answer to our in-depth analysis on the impact of change in environmental temperature on time synchronization accuracy. By cutting through the layers and observing the main cause of frequency error, change in environmental temperature, a temperature aware algorithm can greatly improve the local clock stability and thus decrease the overall power consumption of a time synchronization protocol.
- The last part addressed in this dissertation is the often needed high time resolution in duty-cycled embedded systems. The High-Low Timer solves this problem by intelligently linking a slow frequency clock to a high frequency clock. By implementing this strategy on an FPGA, we reduced the burden of keeping track of time from the microcontroller, which now can be better tasked for communication and sampling. At the same time, the introduction of the slow clock removes the necessity of keeping the high clock active during sleep times, thus greatly reducing the sleep power consumption.

Chapter 3 showed that in order to estimate the current frequency error within a reasonable time frame, high frequency clocks are necessary. Additionally, the shorter

the time necessary to measure that error, the smaller the error due to temperature changes during the measurement interval. For that reason, combining the HLTimer with Temperature Compensated Time Synchronization is the perfect fit, taking the best of both worlds. The high frequency measurements can be used to reduce the time in which TCTS has to stay in calibration mode. For example, if TCTS has to rely on 32kHz crystals, resynchronization rates during calibration have to be 100 seconds long, or else the error estimation would not be optimal. However, with the HLTimer's 8MHz crystal, this interval can be made as small as 10 seconds, achieving even better frequency error estimation due to higher time resolution than with a 32kHz crystal.

This dissertation removes the usual boundary that is drawn between clock source and timing unit, by incorporating frequency error compensation into the time counting element. This was achieved by carefully analyzing the effects on temperature on these clock sources, and exploiting the inherent communication capabilities of sensor network platforms. Thus, we believe that this dissertation makes a strong case for a reexamination of that particular microcontroller peripheral, the timing unit.

6.2 Looking Ahead

When all is said, this dissertation is about the interaction of the lowest timing element, the clock generator, the timer unit keeping time, and how software uses this time for synchronization and timing accuracy. It revisited common assumptions embedded system developers make on each of them, and shows how bringing information from the lowest tier up into the highest timing elements, the algorithms running in the microprocessor, can greatly improve accuracy and power consumption.

REFERENCES

- [AAH97] D.W. Allan, N. Ashby, and C.C. Hodge. *The Science of Timekeeping*. Hewlett-Packard, 1997.
- [AOV97] D. Aebischer, H.J. Oguey, and V.R. Von Kaenel. “A 2. 1-MHz crystal oscillator time base with a current consumption under 500 nA.” *IEEE Journal of Solid-State Circuits*, **32**(7):999–1005, 1997.
- [BMH89] M. Bloch, M. Meirs, and J. Ho. “The microcomputer compensated crystal oscillator (MCXO).” *Frequency Control, 1989., Proceedings of the 43rd Annual Symposium on*, pp. 16–19, 31 May-2 Jun 1989.
- [Bou07] Athanassios Boulis. “Castalia: revealing pitfalls in designing distributed algorithms in WSN.” In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pp. 407–408, New York, NY, USA, 2007. ACM.
- [CCD89] V. Candelier, G. Caret, and A. Debaisieux. “Low profile high stability digital TCXO: ultra low power consumption TCXO.” *Frequency Control, 1989., Proceedings of the 43rd Annual Symposium on*, pp. 51–54, 31 May-2 Jun 1989.
- [CDS98] S.S. Chen, D.L. Donoho, and M.A. Saunders. “Atomic Decomposition by Basis Pursuit.” *SIAM Journal on Scientific Computing*, 1998.
- [CEP07] T. Cooklev, JC Eidson, and A. Pakdaman. “An implementation of IEEE 1588 over IEEE 802.11 b for synchronization of wireless local area network nodes.” *IEEE Transactions on Instrumentation and Measurement*, **56**(5):1632–1639, 2007.
- [CRT06] E.J. Candes, J. Romberg, and T. Tao. “Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information.” *IEEE Transactions on Information Theory*, 2006.
- [CW07] E.J. Candes and M.B. Wakin. “People hearing without listening: An introduction to compressive sampling.” *IEEE Signal Processing Magazine*, 2007.
- [DCS07] P. Dutta, D. Culler, and S. Shenker. “Procrastination Might Lead to a Longer and More Useful Life.” *HotNets-VI*, 2007.
- [DTJ08] Prabal Dutta, Jay Taneja, Jaemin Jeong, Xiaofan Jiang, and David Culler. “A Building Block Approach to Sensornet Systems.” In *SenSys*, 2008.

- [ED04] A. El-Hoiydi and J.D. Decotignie. “WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks.” *Lecture Notes in Computer Science*, **3121**:18–31, 2004.
- [EGE02] J. Elson, L. Girod, and D. Estrin. “Fine-grained network time synchronization using reference broadcasts.” *ACM SIGOPS Operating Systems Review*, **36**:147–163, 2002.
- [Eid06] J.C. Eidson. *Measurement, Control, and Communication Using IEEE 1588*. Springer, 2006.
- [FDL08] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. “Quanto: Tracking Energy in Networked Embedded Systems.” In *OSDI*, 2008.
- [GGS05] Saurabh Ganeriwal, Deepak Ganesan, Hohyun Shim, Vlasios Tsiatsis, and Mani B. Srivastava. “Estimating clock uncertainty for efficient duty-cycling in sensor networks.” In *SenSys*, pp. 130–141, New York, NY, USA, 2005. ACM Press.
- [GKS03] S. Ganeriwal, R. Kumar, and M.B. Srivastava. “Timing-sync protocol for sensor networks.” *SenSys*, 2003.
- [HC04] Moshiul Haque and Ernest Cox. *Use of the CMOS Unbuffered Inverter in Oscillator Circuits*. Texas Instruments, January 2004. Application Report.
- [HC08] Jonathan W. Hui and David E. Culler. “IP is dead, long live IP for wireless sensor networks.” In *SenSys*, pp. 15–28, New York, NY, USA, 2008. ACM.
- [HPC08] John Hicks, jeongyeup Paek, Sharon Coe, Ramesh Govindan, and Deborah Estrin. “An Easily Deployable Wireless Imaging System.” In *ImageSense*, 2008.
- [KDL06] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler. “Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services.” *International Journal of Ad Hoc and Ubiquitous Computing*, **1**(4):239–251, 2006.
- [KS96] K. Kubo and S. Shibuya. “Analog TCXO using one chip LSI for mobile communication.” *Frequency Control Symposium, 1996. 50th., Proceedings of the 1996 IEEE International.*, pp. 728–734, 5-7 Jun 1996.
- [KSS05] S. Knappe, P. Schwindt, V. Shah, L. Hollberg, J. Kitching, L. Liew, and J. Moreland. “A chip-scale atomic clock based on 87Rb with improved frequency stability.” *Opt. Express*, **13**(4):1249–1253, 2005.

- [KV90] J.A. Kusters and J.R. Vig. “Thermal hysteresis in quartz resonators-A review.” *Frequency Control, 1990., Proceedings of the 44th Annual Symposium on*, pp. 165–175, May 1990.
- [LHH00] Se-Joong Lee, Jin-Ho Han, Seung-Ho Hank, Joe-Ho Lee, Jung-Su Kim, Min-Kyu Je, and Hoi-Jun Yoo. “One chip-low power digital-TCXO with sub-ppm accuracy.” *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, 3:17–20 vol.3, 2000.
- [LHT05] MinQiang Li, XianHe Huang, Feng Tan, YanHong Fan, and Xun Liang. “A novel microcomputer temperature-compensating method for an overtone crystal oscillator.” *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 52(11):1919–1922, Nov. 2005.
- [Lip99] Kristen Lippincott. *The Story of Time*. Merrell Holberton, London, 1999.
- [Lle92] Claire Llewellyn. *My First Book of Time*. Dorling Kindersley, Inc., 1992.
- [Max08] Maxim. “DS4026-10MHz to 51.84MHz TCXO.” http://www.maxim-ic.com/quick_view2.cfm/qv_pk/5408, Oct. 2008.
- [Mil91] D. L. Mills. “Internet time synchronization: the network time protocol.” *IEEE Trans. on Communications*, 39(10):1482–1493, 1991.
- [MKS04] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. “The flooding time synchronization protocol.” *SenSys*, pp. 39–49, 2004.
- [MLT08] R. Musaloiu-E, C.-J.M. Liang, and A. Terzis. “Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks.” In *International Conference on Information Processing in Sensor Networks (IPSN '08)*, pp. 421–432, April 2008.
- [Mot] Moteiv. “Tmote Sky Datasheet.”
- [NB63] D.E. Newell and R.H. Bangert. “Temperature Compensation of Quartz Crystal Oscillators.” *17th Annual Symposium on Frequency Control. 1963*, pp. 491–507, 1963.
- [NH68a] D. E. Newell and H. Hinnah. “Automatic Compensation Equipment for TCXO’s.” *22nd Annual Symposium on Frequency Control*, pp. 298–310, 1968.
- [NH68b] D.E. Newell and H. Hinnah. “Automatic Compensation Equipment for TCXO’s.” *22nd Annual Symposium on Frequency Control. 1968*, pp. 298–310, 1968.

- [NNW67] HC Nathanson, WE Newell, RA Wickstrom, and JR Davis Jr. “The resonant gate transistor.” *IEEE Transactions on Electron Devices*, **14**(3):117–133, 1967.
- [PSC05] J. Polastre, R. Szewczyk, and D. Culler. “Telos: enabling ultra-low power wireless research.” In *Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005.
- [RKM05] P. Rantakari, V. Kaajakari, T. Mattila, J. Kiihamaki, A. Oja, I. Tittonen, and H. Seppa. “Low noise, low power micromechanical oscillator.” *Solid-State Sensors, Actuators and Microsystems, 2005. Digest of Technical Papers. TRANSDUCERS '05. The 13th International Conference on*, **2**:2135–2138 Vol. 2, June 2005.
- [SC01] Kwang-Sig Shin and Wan-Young Chung. “Automatic TCXO frequency-temperature test chamber using thermoelectric device array [temperature compensated crystal oscillator].” *Industrial Electronics, 2001. Proceedings. ISIE 2001. IEEE International Symposium on*, **3**:1624–1627 vol.3, 2001.
- [Sch] SS Schodowski. “Resonator self-temperature-sensing using a dual-harmonic-modecrystal oscillator.” *Proceedings of the 43rd Annual Symposium on Frequency Control 1989*, pp. 2–7.
- [SH93] K. Satou and T. Hara. “Temperature detector and a temperature compensated oscillator using the temperature detector.”, May 1993. US Patent 5,214,668.
- [SKL06] J. Sallai, B. Kusy, A. Ledeczki, and P. Dutta. “On the Scalability of Routing Integrated Time Synchronization.” *EWSN*, 2006.
- [SML04] G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. “Sensor network-based countersniper system.” In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 1–12. ACM New York, NY, USA, 2004.
- [SW09] Philipp Sommer and Roger Wattenhofer. “Gradient Clock Synchronization in Wireless Sensor Networks.” In *IPSN*, 2009.
- [Vig92] John R. Vig. “Introduction to Quartz Frequency Standards.” *Army Research Laboratory, Electronics and Power Sources Directorate*, 1992.
- [VXS07] M. Varshney, D. Xu, M. Srivastava, and R. Bagrodia. “sQualNet: A Scalable Simulation and Emulation Environment for Sensor Networks.” *IPSN '07*, 2007.

- [YS08] S. Yoon and M. L. Sichitiu. “Power Consumption Models for Wireless Sensor Networks MAC Protocols.”, 2008.
- [YSH06] W. Ye, F. Silva, and J. Heidemann. “Ultra-low duty cycle MAC with scheduled channel polling.” *SenSys*, pp. 321–334, 2006.
- [ZZX05] Wei Zhou, Hui Zhou, Zongqiang Xuan, and Wenqing Zhang. “Comparison among precision temperature compensated crystal oscillators.” *Frequency Control Symposium and Exposition, 2005. Proceedings of the 2005 IEEE International*, 29-31 Aug. 2005.