TEACHING DESIGN PATTERNS IN A REAL-TIME EMBEDDED SYSTEMS INTERFACING COURSE

Witold Kinsner

Cognitive Systems Laboratory
Department of Electrical and Computer Engineering
University of Manitoba, Winnipeg, MB, Canada R3T 5V6
witold.kinsner@umanitoba.ca

***Abstract**: This paper presents an overview of design patterns for teaching an undergraduate course on interfacing of microcontrollers, microprocessors and microcomputers for real-time systems. Such design patterns are useful because the course must cover a wide range of topics for both wired and wireless systems, and is intended for a variety of microcontrollers. Without the patterns, teaching all the material from ground up might not be feasible.*

***Keywords**:* Real-time interfacing; embedded systems; design patterns; microcontrollers.

1. INTRODUCTION

1.1 Challenges with Teaching Interfacing

This paper presents a few design patterns for teaching an undergraduate course on interfacing of microcontrollers, microprocessors and microcomputers for real-time systems [20], [21]. The course is not based on one microcontroller (usually taught prior to a course on interfacing and embedded systems), but on a wider range of microcontrollers and microprocessors. The course is also intended to generalize the architectures and organizations in order to prepare the students for new technologies. Teaching such a course material from ground up is not feasible because of the extensive detail required.

The difficulty of teaching a course in this area stems from the diversity of topics that must be covered. Figure 1 shows the granularity of topics included in the course (only the two top levels of topics are shown).

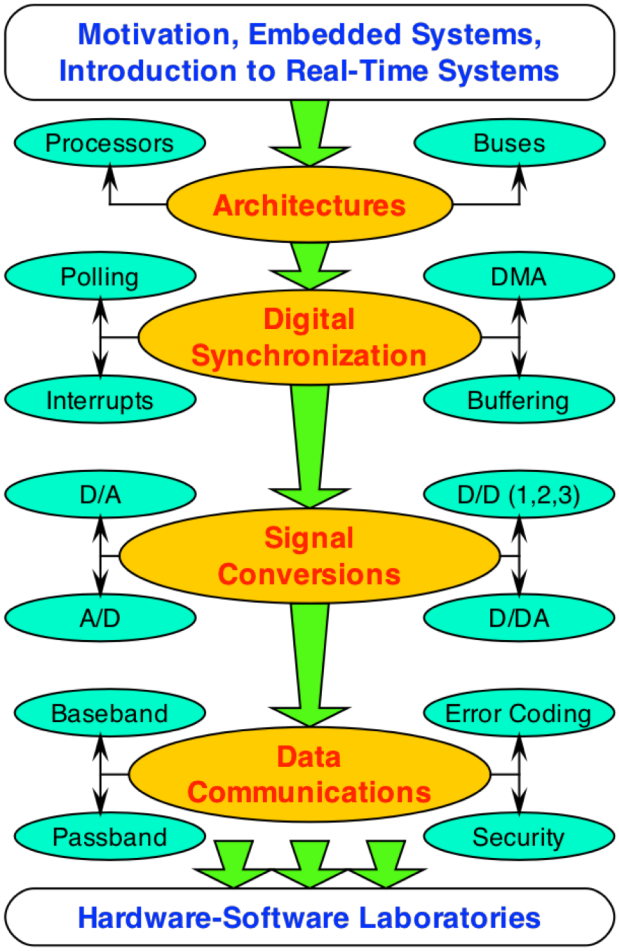

Fig. 1 Elements of an interfacing course.

1.2 Course Scope

The five top-level topic modules include: (i) an introduction to real-time computing, architectures, processors, and technologies, (ii) the **architectures** of the main microcontrollers and microprocessors, as well as of internal, external, peripheral and system buses, (iii) digital input/output (I/O) **synchronization** (e.g., different classes of polling and interrupts, direct memory access (DMA), context switching, and the major buffering techniques for isolation of input from the output, and for de-synchronization between the input and output), (iv) digital-to-analog (D/A), analog-to-digital (A/D), and digital-to-digital (D/D) **signal conversions**, as well as the core types of converters (e.g., parallel flash, serial counting (single ramp), tracking, successive approximation (SA), integrating (dual slope, quad slope), voltage-to-frequency (V/F)-based converters, and delta-sigma converters [22]), and finally (v)

interfacing aspects in **data communications** (e.g., major wired (baseband) and wireless (passband) data communications protocols, data encoding, signal conditioning, forward error detection and correction).

The D/D1-3 **digital-to-digital signal conversion** refers to source coding, security coding, and channel coding, while D/DA refers to line coding. **Source coding** (compression) changes the original digital bit stream to another more compact bit stream by removing redundancy from the original data. This increases data throughput by transmitting or storing the relevant information only. On the other hand, compression makes the data stream more vulnerable to errors on the channel or in storage. **Security coding** refers to protecting data against their inappropriate use, and is necessary in all digital systems now [19].

As an integral part of interfacing, the course also presents some aspects of **channel coding** with its **error detection and correction** [18]. It distinguishes between error detection and correction techniques that are suitable for real-time systems. It covers error correction techniques that use *retransmission* (the automatic retransmission request, ARQ) and *forward error correction* for real-time data communications such as the Hamming family of codes for memoryless channels. The benefits of channel coding come at a price: the error protection requires some extra redundancy in the original bit stream, thus reducing its throughput. To cover the topics properly in the course, the D/D conversions are presented using design patterns.

Finally, **line coding** translates the data bit stream into another bit stream in order to achieve several objectives: (i) to reduce the DC component of the signal due to long runs of 1s (or 0s) in order to reduce line wandering, (ii) to embed the data clock into the data bits, thus achieving self-clocking codes, and (iii) to make the bit stream differential, thus making it independent of wire polarity [17]. The benefits of line coding also come at a price: the bandwidth of the signal may be increased, thus affecting its power to maintain the signal to noise ratio and error protection. This paper presents an example of line coding design patterns in Sec. 3.

1.3 Course Lab Structure

The previous implementations of our lab experiments in this course utilized various microprocessors and microcontrollers, including the 6800, 68000, 6805, HC08, and the HC11, all in printed-circuit boards designed and implemented by us in house. To keep up with newer technologies and architectures, a new set of labs based on the HCS12 microcontroller development board has been implemented [23]. The new laboratories have been developed and presented to the students twice, followed by numerous improvements and corrections of the lab manual. A new tutorial on the HCS12 has also been developed. Selection of topics in the tutorial, their presentation and structure are all quite novel, and should be helpful in reinforcing the knowledge of this important pipelined microcontroller that is compatible with many legacy microprocessors and microcontrollers.

2. INTRODUCTION TO DESIGN PATTERNS

2.1 What is a Pattern?

A pattern is a regularity in either the physical, or man-made, or abstract world as discerned by our senses or by mathematical analysis. The patterns often include symmetry, spirals (often logarithmic), trees, waves, bubbles, stripes, fractures. Those patterns can be either static, or dynamic, or dynamical, and originate from deterministic or stochastic processes. Many of the rich patterns are self-affine (fractal and multifractal) [37], [26], [28].

Discovering patterns and their dynamics in both nature and the abstract world is an important part of life. It is important in mathematics (transformations and modelling) [1], [31], [35], [34], physics (growth, percolation, materials) [36], [5], chemistry (pattern in space such as symmetry in a molecule affecting its infrared spectrum) [3], geology and geophysics [37], engineering (statics and dynamics) [12], [4], stocks (trends) [33], biology and medicine (patterns of health) [38], [11], [39], and other areas.

2.2 What is a Design Pattern?

Design patterns are intended to capture the best practices in a specific domain. They are formally-documented solutions to design problems, often found in real-world applications, and presented in a compact form that can be easily communicated to those who need the knowledge. It is a description or a template for how to solve a problem that can be used in many different situations. “Patterns are about reusable designs and interactions of objects. A pattern documents a recurring problem–solution pairing within a given context. A pattern, however, is more than either just the problem or just the solution structure: it includes both the problem and the solution, along with the rationale that binds them together” [10].

The term "design pattern" is usually attributed to Christopher Alexander and his colleagues in the context of architecture, urban design, and community liveability, where it signifies a way of capturing and communicating critical design ideas [2].

A design pattern has the following attributes:

(i) It must explain why a particular situation causes problems, and why the proposed solution is considered a good one;

(ii) It does not provide a single solution to a problem, but proposes a guide and a stencil toward a best design for a particular application under specific constraints;

(iii) It must also explain the range of situations in which a pattern can be used (i.e., the context); and
(iv) Patterns enhance the different perspectives on a solution [9]

2.3 What is a Design Pattern Language?

A pattern language is a set of design patterns that work together to generate complex behaviour and complex artefacts, while each pattern within the language is simple in itself. Patterns in isolation provide only incremental improvements to software systems, processes, organizations, and design. A pattern language may provide a more fundamental and more lasting improvement.

2.4 Design Patterns In Different Disciplines

Patterns in Software Engineering

In software engineering, design patterns are very important and have been used for a long time (e.g., Gamma and colleagues [15] who were inspired by Alexander's work). Design patterns can speed up the development process by providing tested, proven development paradigms.

Patterns in Human-Machine Interaction (HMI)

Design patterns in the human-computer interaction (HCI) are often used for developing user interfaces because they render the communication among stakeholders efficient, and allow for quicker design of the interfaces [13], [30].

Collaboration patterns

Patterns in collaborative working environments (CWE) have also been defined at different levels of granularity and in relation to different application contexts [32]. They are intended to enhance collaboration and its impact on projects.

Patterns in Education

These examples of patterns are closely related to the design patterns discussed in this paper. Pedagogical patterns have been developed to capture expert knowledge (the essence) of the practice of teaching and learning [7], [6]. Recurring teaching problems include: choosing and sequencing materials, evaluating students, motivating students, and instilling life-long learning. Every new instructor can benefit from a senior faculty member and from a pattern language. Design patterns have been developed for good seminars and their delivery [14]. Lecture design patterns have been described to increase the efficiency of this form of education through a foundation of good lectures [29]. Experiential learning design patterns have also been developed [8]. Another set of patterns has been developed to gain different perspectives in the class [9].

3. An Example of Interfacing Patterns

This example is taken from the data communications module (Fig. 1) which is very rich in content and includes: (i) source coding, (ii) security coding, (iii) channel coding, (iv) line coding, and (v) digital modulation for wireless.

3.1 Scope of Line-Coding in the Course

The source, security and channel coding stages are required in preparing data for storage and/or transmission, but are not sufficient to transmit data because the bits are still in the internal representation of the computing node. Line coding is concerned with translating the abstract state representations (binary or non-binary symbols) into a physical digital temporal form (a boxcar signal) for transmission over digital communications channels, either wired (baseband) or wireless (passband).

The objectives of the line coding are: (i) the maximum data rate with the smallest signal rate, (ii) the smallest bandwidth, (iii) no DC component, (iv) self-synchronization, (v) immunity to noise and interference, and (vi) built-in error detection. Those objectives are often conflicting.

3.2 Types of Line Codes

Line coding includes various types of codes, as shown in Fig. 2.

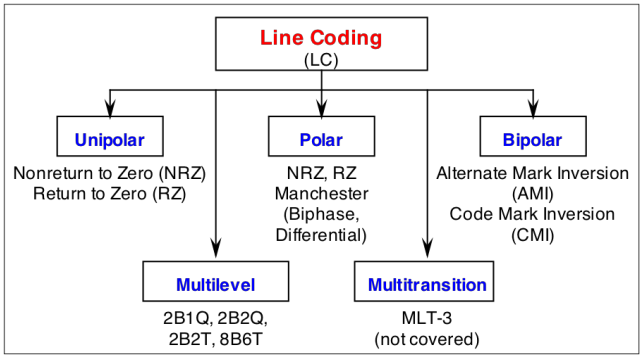

Fig. 2 Types of line codes. (After [Kins15b, p. 16])

Unipolar Codes: All the signal levels $\Sigma s = \{LO,HI)$ for all the symbols in the message alphabet (e.g., $\Sigma m = \{0,1)$) are either positive or negative, not both. Recall that an alphabet is a set of non-repeating symbols.

Polar Codes: Signal levels for the message symbols are both positive and negative (at different times).

Bipolar Codes: Signal levels for a single message symbol alternates sign (at different times) while the other symbol is zero. Example: AMI.

Multilevel Codes (mBnL): This line code can transmit one m-bit message symbol at a time. The m-bit symbol message alphabet is $\Sigma m = \{m1,m2,\cdots\}$, with cardinality M. For example, if each message symbol has $m = 3$ bits, $M = 2^m = 8$. To transmit each symbol in one signal symbol slot, the signal symbol would have to have $L = 8$ levels. This code would be denoted as (8B1O) where O stands for "octal." However, detecting eight signal levels is not easy in the presence of noise.

In order to reduce the number of signal levels to $L = 4$, we must split the message symbol cell into two slots. Since both slots have the same number of signal symbols, the total number of signal symbols is $N = L^n = 4^2 = 16$. Since we need only 8 distinct signals, the remaining 8 can serve as alternatives for code balancing, or as error detection. This code would be denoted as (3B2Q) where Q stands for "quad."

We might further simplify the decoding by reducing the number of signal levels to three ($L = 3$). Since the number of distinct signals in the two slots is $N = L^n = 3^2 = 9$, we can eliminate the signal condition 00, and still have the required 8 signal symbols to transmit the eight message symbols. The code is denoted by (3B2T) where T stands for "ternary."

All these schemes belong to a broad class of pulse amplitude modulation (PAM) signal encoding. Each signal symbol requires a period of time (symbol cell), T_s, to be formed for transmission. Thus, the signalling rate (the number of signal changes per unit of time expressed in baud) is $f_s = 1/T_s$. If the symbol is a bit, then the signalling rate is the same as bit rate in bits per second, bps.

3.3 Basic Line-Coding Design Patterns

The course provides an explanation of the codes structure, with emphasis on the reasons for each new code. For example, the minimization of the DC component is demonstrated through the changes from the unipolar non-return-to-zero level (NRZ-L) signal to the polar NRZ-L, then from the unipolar return-to-zero (RZ) to polar RZ and to the bipolar alternate-mark-inversion (AMI) signal. This is followed by demonstration of why embedding a data clock in the data stream is necessary, and what the conditions for self-clocking codes are, with examples of the construction of the pulse-width modulation (PCM), double-frequency code (DFC), Manchester biphase, and the code-mark-inversion (CMI). The clock recovery techniques are also discussed (e.g., the phase-locked loop, PLL, and the maximum lakelihood estimation, MLE). The conditions for polarity independence are also discussed, followed by the construction of a Manchester differential code and its derivatives. Code scrambling is discussed, including the bipolar with 8-zero substitution (B8ZS) and the high-density bipolar 3 (HDB3) as a modification of the AMI code.

Notice that all the codes were explained using a corresponding design pattern. The collection of the design patterns constituted a design-pattern language (DPL) for the Line codes. Within the language, the principle of code construction evolved naturally. For example, the binary signalling scheme transmits a message (a binary bit stream) by converting it into a suitable analog boxcar signal form such as the Manchester differential. The message is constructed from an alphabet consisting of two symbols $\Sigma m = \{0,1\}$. The signal is also constructed from an alphabet consisting of two symbols $\Sigma s = \{s1,s2\} = \{LO, HI\}$, where LO may be either 0, or $-A$, or $-A/2$ (denoted as LO = $\{0 \mid -A \mid -A/2\}$ where the vertical bar denotes OR), and HI = $\{A \mid A/2\}$. For example, the NRZ code is taken from {0, A}, and Manchester from {–A/2, A/2}. The amplitude, A, can be either 15 volts (V), 5 V, 3.3 V or some other value.
In the context of this design pattern language, we can ask if there is any other code to perform better than the binary code?

3.4 The Multilevel mBnL Design Patterns

The multilevel signalling (mBnL) is introduced by rephrasing the NRZ signal as a two-level PAM (2PAM) and its equivalent 1-bit message symbol and 1-bit binary signal symbol (1B1B), as shown in Fig. 3.

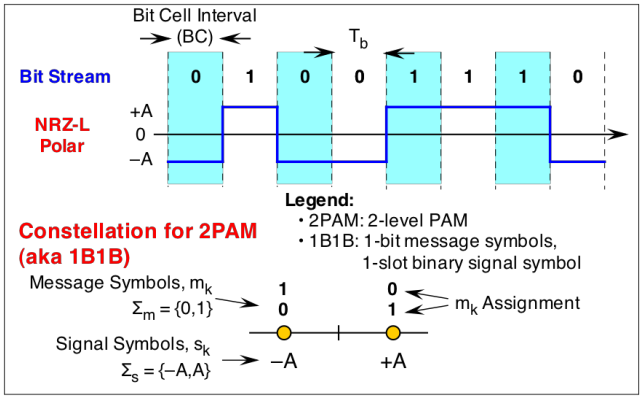

Fig. 3 NRZ constellation. (After [Kins15b, p. 35])

The 4PAM (2B1Q) signal is shown in Fig. 4. The features of the new code are discussed, with emphasis on its inability to control the DC component, even though it has some self-clocking capabilities. What should be done in order to achieve a better self-clocking and DC control?

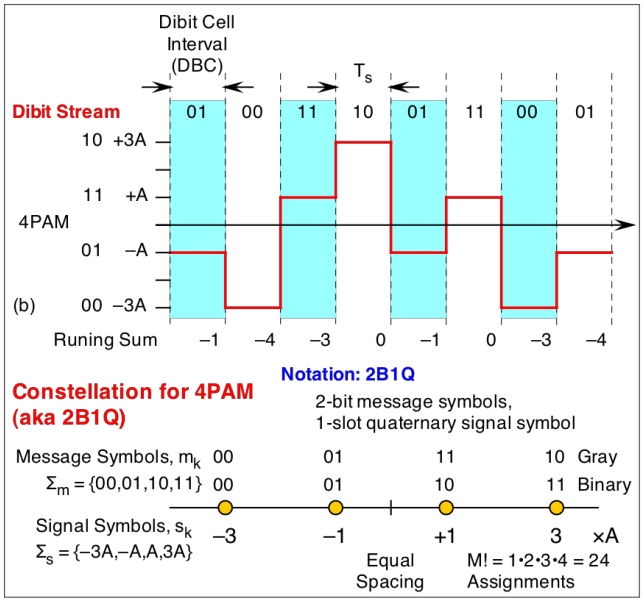

Fig. 4 2B1Q line code. (After [Kins15b, p. 36])

This analysis resulted in a second rephrasing of the RZ code: The bit cell for a 1 in the RZ code can be considered as having two time slots: the first has a HI, and the second has a LO. Thus, the signal symbol for a 1 is (HI,LO), with the transition in the middle of the bit cell, while the signal symbol for a 0 is (LO,LO), as shown in Fig 5.

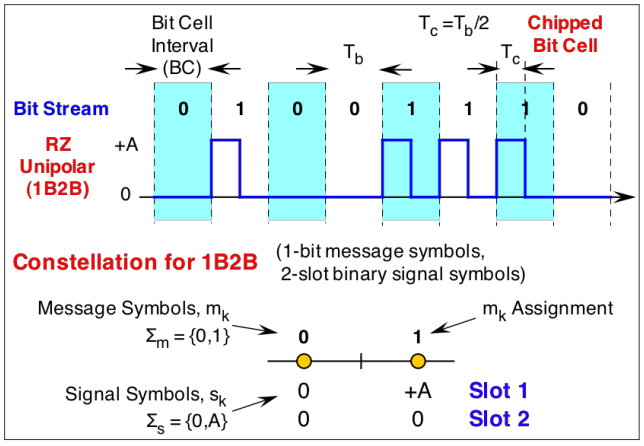

Fig. 5 RZ constellation. (After [Kins15b, p. 39])

By changing the signal symbol for a 0 from (LO,LO) to (LO,HI), the original Manchester code is re-discovered, as shown in Fig 6! This (1B2B) code is self-clocking, and it has error-detection capabilities because the (LO,LO) and (HI,HI) symbols are illegal. Notice that the contemporary definition of the Manchester code reverses its polarity.

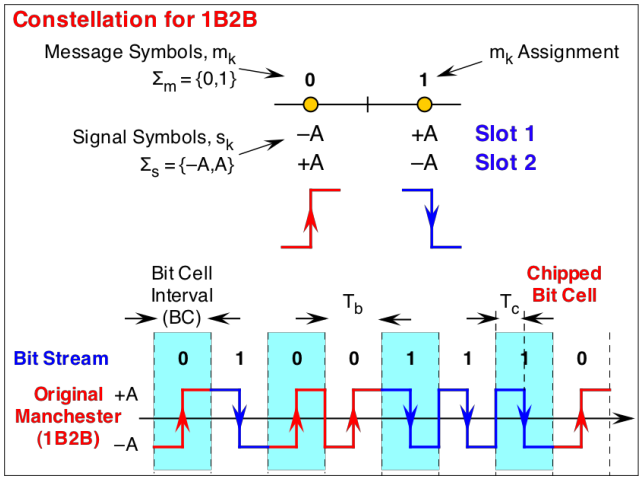

Fig. 6 Manchester code. (After [Kins15b, p. 40])

Having established the principles behind the mBnL codes, the discussion continues with a (2B2Q) code which is a perfectly-balanced code, but is highly underutilized (4 symbols used out of the total of 16), as shown in Fig 7. To increase the code utilization, a code 3B2Q (tree bits, two slots, four signal levels) is introduced and discussed. Since at this point, the understanding of the codes is quite complete, the discussion proceeds with the code 4B3T (four message bits, three slots, each with ternary signal levels) and its extensions.

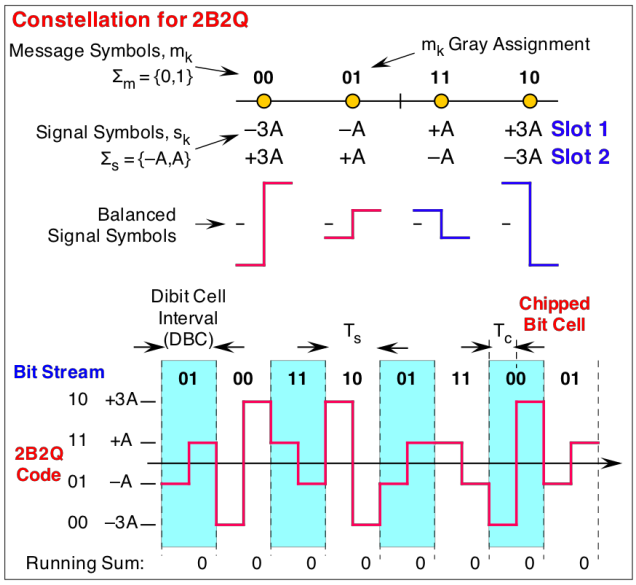

Fig. 7 2B2B line code. (After [Kins15b, p. 41])

This baseband encoding design pattern language is also used to explain passband encoding with various shift-keying schemes (ASK, FSK, PSK, and QAM) [17].

5. CONCLUDING REMARKS

This paper presents an example of **engineering and educational design patterns** situated in an undergraduate course on real-time interfacing of embedded systems. This is a one-term four-credit hour course with a laboratory. The material that must be covered in such a course is very diverse, with many details on hardware and software. Without design patterns, the material covered would appear to be very disjoint, and would emphasize the "how-to implement" a specific circuit, or module, or subsystem, or system. Such a "how-to" exposure has its merits for the first-time designers and system maintainers, but is inadequate for engineers who must solve problems with new and emerging technologies.

A major part of engineering education is (i) the extraction of general patterns from a variety of specific materials presented, (ii) the ability to discern between detail and a time/space-invariant feature of the system, (iii) "connecting the dots" (generalization) between different elements of the design, and (iv) the ability to deal with new technologies and algorithms that emerge in the future. Engineering and educational design patterns have helped significantly in achieving those goals in the real-time interfacing of embedded systems course.

Design patterns may also be helpful in teaching other courses such as cognitive systems [27], [25], as well as in the development of new bodies of knowledge for practitioners (BoK4P) [16], and in the modification of existing courses to include humanitarian engineering [24].

Acknowledgements

We would like to thank the Department of Electrical and Computer Engineering at the University of Manitoba for partial financial support of this project. I would like to thank many students from my undergraduate and graduate courses who expressed the need for engineering design patterns. I would also like to thank many of my colleagues in education, and particularly Ken Ferens, for discussions on how to teach for the future.

References

[1] John A. Adam, *Mathematics in Nature: Modeling Patterns in the Natural World*. Princeton, NJ: Princeton University Press, 2003, 360 pp. {ISBN 0-691-11429-3, pbk}

[2] Christopher Alexander, Sara Ishikawa, and Murray Silverstein, *A Pattern Language: Towns, Buildings, Construction*. Oxford, UK: Oxford University Press, 1977, 1171 pp.. {ISBN 0-19-501919-9}

[3] Bruce A. Averill and Patricia Eldredge, *General Chemistry: Principles, Patterns, and Applications*. Washington, DC: Saylor Academy, 2015, 2365 pp. {ISBN 13: 978-1-4533223-0-7, PDF, Open Textbook} Retrieved May 9, 2015 from the Saylor Academy at https://open.umn.edu/opentextbooks/BookDetail.aspx?bookId=69 http://www.saylor.org/books

[4] Soumitro Banerjee and George C. Verghese, *Nonlinear Phenomena in Power Electronics: Attractors, Bifurcations, Chaos, and Nonlinear Control*. Piscataway, NJ: IEEE Press, 2001, 441 pp. {ISBN 0-7803-5383-8, hbk}

[5] Albert-László Barabási and H. Eugene Stanley, *Fractal Concepts in Surface Growth*. Cambridge, UK: Cambridge University Press, 1995, 366 pp. {ISBN 0-521-48318-2, pbk}

[6] Joseph Bergin, *Some Pedagogical Patterns*. New York, NY: Pace University, 2015. Retrieved May 4, 2015 from Pace University at http://csis.pace.edu/~bergin/patterns/fewpedpats.html#unp http://csis.pace.edu/~bergin

[7] Joseph Bergin, Jutta Eckstein, Markus Völter, Marianna Sipos, Eugene Wallingford, Klaus Marquardt, Jane Chandler, Helen Sharp, Mary Lynn Manns (eds.), *Pedagogical Patterns: Advice for Educators*. Joseph Bergin Software Tools, 2012, 190 pp. {ISBN-13: 978-1479171828} Retrieved May 4, 2015 from Joseph Bergin at http://www.pedagogicalpatterns.org/

[8] Joseph Bergin, Klaus Marquardt, Mary Lynn Manns, Jutta Eckstein, Helen Sharp, and Eugene Wallingford, "Patterns for experiential learning," in *Proc. 6th European Conference on Pattern Languages of Programs, EuroPLoP01*. Andreas Rüping, Jutta Eckstein, and Christa Schwanninger (eds.) (Kloster Irsee, Bavaria, Germany; July 4-8, 2001), July 2002, 580 pp. {ISBN 978-3-87940-780-4; 59.00€} 19 pp., 2001. Retrieved May 4, 2015 from UVK Verlagsgesellschaft at http://www.agile-practice.com/pedagogicalPatterns/experientiallearning.pdf

[9] Joe Bergin, Jutta Eckstein, Mary Lynn Manns, and Eugene Wallingford, "Patterns for gaining different perspectives: A part of the Pedagogical Patterns Project pattern language," in *Proc. 6th European Conference on Pattern Languages of Programs, EuroPLoP01*. Andreas Rüping, Jutta Eckstein, and

Christa Schwanninger (eds.) (Kloster Irsee, Bavaria, Germany; July 4-8, 2001), July 2002, 580 pp. {ISBN 978-3-87940-780-4; 59.00€} 17 pp., 2001. Retrieved May 4, 2015 from UVK Verlagsgesellschaft at http://www.europlop.net/content/past-europlops http://www.europlop.net/content/annual-proceedings http://www.europlop.net/

[10] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt, *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. Vol. 5. Hoboken, NJ: Wiley, 2007, 490 pp. {ISBN: 978-0-471-48648-0, hbk; $77} Retrieved May 4, 2015 from UVK Verlagsgesellschaft at http://media.wiley.com/product_data/excerpt/28/04700590/0470059028.pdf

[11] Scott Camazine, Jean-Louis Deneubourg, Nigel R. Franks, James Sneyd, Gyu Theraulaz, and Eric Bonabeau, *Self-Organization in Biological Systems*. Princeton, NJ: Princeton University Press, 2001, 538 pp. {ISBN 978-0-691-11624-2, pbk}

[12] Guanrong Chen (ed.), *Controlling Chaos and Bifurcations in Engineering Systems*. Boca Raton, FL: CRC Press, 1999, 648 pp. {ISBN 0-8493-0579-9, hbk}

[13] Andy Dearden & Janet Finlay, "Pattern languages in HCI: a critical review," *Human Computer Interaction*, vol. 21, no. 1, pp. 49-102, 2006. Retrieved May 4, 2015 from UVK Verlagsgesellschaft at http://shura.shu.ac.uk/22/2/DeardenFinlayFormatted.pdf

[14] Astrid Fricke and Markus Völter, "Seminars: A pedagogical pattern language about teaching seminars effectively," in *Proc. Fifth European Conference on Pattern Languages of Programs, EuroPLoP 2000* (Kloster Irsee, Germany; July 5-9 2000) 2000, 36 pp. Retrieved May 4, 2015 from UVK Verlagsgesellschaft at http://www.coldewey.com/europlop2000/papers/voelter+fricke.zip http://www.coldewey.com/europlop2000/contents.html http://www.voelter.de/seminars

[15] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994, 395 pp. {ISBN 0-201-63361-2}

[16] Witold Kinsner, "Expanding the Body of Knowledge Concept for Professional Practitioners (BoK4P)," in *Proc. 2015 Canadian Engineering Education Association Conference, CEEA15* (Hamilton ON: May 31 to June 3, 2015), Paper 172, 12 pp, 2015.

[17] Witold Kinsner, *Line Coding: Encoding and Modulation of Analog Signals*. Technical report. Winnipeg, MB: University of Manitoba, Electrical and Computer Engineering, Version 327, February 2015, 77 pp.

[18] Witold Kinsner, *Error Detection and Correction: ARQ and FEC*. Technical report. Winnipeg, MB: University of Manitoba, Electrical and Computer Engineering, Version 323, March 2015, 76 pp.

[19] Witold Kinsner, *Security: Protection of Embedded Systems*. Technical report. Winnipeg, MB: University of Manitoba, Electrical and Computer Engineering, Version 329, April 2015, 84 pp.

[20] Witold Kinsner, *Microcontroller, Microprocessor, and Microcomputer Interfacing for Real-Time Systems*. Lecture Notes. Winnipeg. MB: University of Manitoba, 2014, 643 pp.

[21] Witold Kinsner, *Laboratories for Microcontroller, Microprocessor, and Microcomputer Interfacing for Real-Time Systems*. Lab Notes; Winnipeg. MB: University of Manitoba, 2014, 102 pp.

[22] Witold Kinsner, "Teaching delta-sigma conversion in an interfacing course," in *Proc. 2014 Canadian Engineering Education Association Conference, CEEA14* (Canmore, AB: June 8-11, 2014), Paper 107, 8 pp, 2014.

[23] Witold Kinsner, "Selecting a microcontroller development systems for a laboratory in a real-time interfacing course," in *Proc. 2014 Canadian Engineering Education Association Conference, CEEA14* (Canmore, AB: June 8-11, 2014), Paper 113, 10 pp, 2014.

[24] Witold Kinsner, "Humanitarian engineering education: Examples," in *Proc. 5th Conference of the Canadian Engineering Education Association, CEEA 2014* (Canmore, AB; June 8-11, 2014). Paper 121, 6 pp., 2014.

[25] Witold Kinsner, Simon Haykin, Yingxu Wang, Witold Pedrycz, Ivo Bukovsky, Bernard Widrow, Andrzej Skowron, Piotr Wasilewski, and Menahem Friedman, "Challenges in engineering education of cognitive dynamic systems," in *Proc. of the Canadian Engineering Education Association Conference, CEEA 2012*, (Winnipeg, MB, Canada; June 17–20, 2012), Paper 119, pp. 51-62, 2012. Accessed Mar 2015 from CEEA at http://library.queensu.ca/ojs/index.php/PCEEA/article/

viewFile/4633/4615
http://www.ceea.ca/images/content/ceea12-proc-complete-v35s.pdf

[26] Witold Kinsner, "System complexity and its measures: How complex is complex," in Yingxu Wang, Du Zhang, and Witold Kinsner (eds.), *Advances in Cognitive Informatics and Cognitive Computing*. Berlin: Springer Verlag, Vol SCI 323, pp. 265-295, 2010. {ISBN 978-3-642-16082-0; eISBN 978-3-642-16083-7}

[27] Witold Kinsner, "Challenges in the design of adaptive, intelligent and cognitive systems," *Intern. J. Software Science & Computational Intelligence*, vol. 1, no. 3, pp. 16-35, July-Sept. 2009.

[28] Witold Kinsner, "A unified approach to fractal dimensions," *Intern. J. Cognitive Informatics and Natural Intelligence*, vol. 1, no. 4, pp. 26-46, Oct-Dec 2007.

[29] Christian Köppe and Joost Schalken-Pinkster, Lecture Design Patterns: Laying the Foundation," in *Proc. 18th European Conference on Pattern Languages of Programs, EuroPLoP13* (Kloster Irsee, Germany; July 10-14, 2013) 2013, 27 pp.
http://koeppe.nl/publications/LecturePatterns-Foundation-authorversion.pdf
http://www.europlop.net/content/europlop-2013
http://www.europlop.net/

[30] Christian Kruschitz and Martin Hitz, "Analyzing the HCI design pattern variety," in *Proc. 1st Asian Conference on Pattern Languages of Programs, AsianPLoP 2010* (Tokyo, Japan; March 16-17, 2010) Paper #6, 2010. {doi>10.1145/2371736.2371745} Retrieved May 2015 from the ACM Library at
http://scholar.google.ca/scholar_url?url=http%3A%2F%2Fdl.acm.org%2Fcitation.cfm%3Fid%3D2371745&hl=en&sa=T&ct=res&cd=0&ei=o0lQVZzDMdLtqQGvyoHYCg&scisig=AAGBfm3WHauXsQwgxQdwU8SCRrnDq2QZwg&nossl=1&ws=1234x1002

[31] Klaus Mainzer, *Thinking in Complexity: The Computational Dynamics of Matter, Mind, and Mankind*. New York, NY: Springer, 2004 (4th ed.), 456 pp. {ISBN 3-540-62555-0, hbk}

[32] Jonas Pattberg and Matthias Fluegge, "Towards an ontology of collaboration patterns," in *Proc. 5th International Workshop on Challenges in Collaborative Engineering, CCE'07* (Cracow, Poland; April 11-13, 2007) Lecture Notes in Informatics, Vol. P-120, pp. 85-96, 2007.
http://cs.emis.de/LNI/Proceedings/Proceedings120/gi-proc-120-007.pdf
http://subs.emis.de/LNI/Proceedings/Proceedings120.html
http://subs.emis.de/LNI/Proceedings.html
http://www.springer.com/computer/lncs?SGWID=0-164-2-470309-0

[33] Edgar E. Peters, *Fractal Market Analysis: Applying Chaos Theory to Investment and Economics*. New York, NY: Wiley, 1994, 315 pp. {ISBN 0-471-58524-6, hbk}

[34] Przemyslaw Prusinkiewicz and Aristid Lindenmayer, *The Algorithmic Beauty of Plants*. New York, NY: Springer, 1990, 228 pp. {ISBN 0-387-97297-8, hbk}

[35] Manfred Schroeder, *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise*. New York, NY: W.H. Freeman, 1991, 429 pp. {ISBN 0-7167-2136-8, hbk}

[36] Tamás Vicsek, *Fractal Growth Phenomena*. Singapore: World Scientific, 1992 (2nd ed.), 488 pp. {ISBN 9810206690, pbk}

[37] Donals L. Turcotte, *Fractals and Chaos in Geology and Geophysics*. Cambridge, UK: Cambridge University Press, 1997, 398 pp. {ISBN 0-521-56733-5, pbk}

[38] Jan Wallechek (ed.), *Self-Organized Biological Dynamics and Nonlinear Control*. Cambridge, UK: Cambridge University Press, 2000, 428 pp. {ISBN 0-521-62536-3, hbk}

[39] Arthur T. Winfree, *The Geometry of Biological Time*. New York, NY: Springer, 2006 (2nd ed.), 777 pp. {ISBN 0-387-98992-7, hbk}