The Design of a Deep Space Transponder Regenerative Ranging Unit

by

David Samuel Warren

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1995

Author .....
Department of Electrical Engineering and Computer Science, May 26, 1995

Certified by .....
James K. Roberge, Prof. of Elec. Eng., Thesis Supervisor (Academic)

Certified by .....
Arthur W. Kermode, Company Supervisor (Jet Propulsion Laboratory)

Accepted by .....
F. R. Morgenthaler, Chair, Department Committee on Graduate Students

The Design of a Deep Space Transponder Regenerative Ranging Unit

by

David Samuel Warren

Submitted to the Department of Electrical Engineering and Computer Science on May 26, 1995, in partial fulfillment of the requirements for the degree of Master of Science

Abstract

Ranging signals are transmitted from the ground to deep space spacecraft, then returned to the ground in order to determine the spacecraft's distance from the earth. Currently, the transponders used on these spacecraft have no capability to regenerate the ranging signals. A scheme for implementing this regeneration, which leads to a dramatic reduction in the ranging signal's noise power (up to 40 dB) is proposed. The regeneration system was designed, and implemented with Actel field programmable gate arrays (FPGAs). After several iterations, the design was simulated successfully.

Thesis Supervisor: Professor James Roberge
Title: Professor of Electrical Engineering

Acknowledgments

The author gratefully acknowledges the assistance of several people. Arthur Kermode and Prof. J. K. Roberge supervised this work, and made this project possible. John Smith and Glenn Johnson provided crucial information on the sequential and PN ranging systems, as well as providing suggestions on how they could be modified. Renee Watson and Dr. Selahattin Kayalar provided assistance with Comdisco SPW, and made many suggestions involving design improvements. Todd Mackett provided the adapter needed to program the Actel FPGAs. Jeff Wong and Prof. V. M. Bove allowed the use of facilities at the MIT Media Laboratory to complete some of the hardware construction.

Table of Contents

Chapter 1. Introduction 15
1.1 General Overview 15
1.2 Satellite Communication 15
1.3 Why regenerate? 16
1.4 My Thesis Project 16
Chapter 2. Theoretical Analysis 19
2.1 Ranging 19
2.1.1 Sequential Ranging 19
2.1.2 PN Ranging 20
Chapter 3. Design 25
3.1 Overview 25
3.2 Comdisco Designs 25
3.2.1 Generation Unit 25
3.2.2 Regeneration Unit 26
3.3 Mentor Graphics Designs 28
3.3.1 Generation Unit 28
3.3.2 Regeneration Unit 28
3.4 Actel FPGA Design 29
3.5 Physical Hardware Construction 29
Chapter 4. Results and Discussion 33
4.1 Comdisco Simulation Results 33
4.2 Mentor Graphics Simulation Results 34
4.3 Post-layout Timing Simulation Results 37
Chapter 5. Conclusion 41
5.1 Summary 41
5.2 Recommendations 41
Appendix A. Graphs of PN component Auto- and Cross-correlation functions 43
A.1 Graphs of Autocorrelation Functions 43
A.2 Graphs of Cross-correlation Functions 47
Appendix B. Comdisco Block Diagrams 55
Appendix C. Systems used to Test Comdisco Designs 171
Appendix D. FSM description files 205
D.1 FSM State Diagram 205
D.2 Fsmc input file 206
D.3 FSM state file 207
D.4 FSM equation file 210
Appendix E. Mentor Graphics Schematics 211
Appendix F. Nonstandard Actel Macros 257
Appendix G. Revised Mentor Graphics designs for Actel FPGAs 281

Appendix H. Test Vector Generation Code 297
H.1 C Code Used to Generate Test Vectors 297
H.2 Header File Prepended to Output of C Code 300
H.3 Modified C Code Used to Create Test Vectors for Modified Timing 301
H.4 Modified Header File Prepended to Output of Modified C Code 304
References 305

List of Figures

Figure 2.1: Spectrum of combined PN sequence. 21
Figure 2.2: FFT of clock component. 22
Figure 2.3: Clock component and combined PN sequence (time domain) 22
Figure 3.1: Simplified Block Diagram of Regeneration Unit (with transponder). 26
Figure 3.2: Generation Unit Wirewrap Board Schematic 30
Figure 3.3: Regeneration Unit Wirewrap Board Schematic. 31
Figure 4.1: Comdisco simulation output - no noise on input 33
Figure 4.2: Comdisco simulation output - additive white noise on input. 34
Figure 4.3: Mentor functional simulation without additive noise 35
Figure 4.4: Mentor functional simulation with additive noise 36
Figure 4.5: Mentor timing simulation - Clock speed too high 38
Figure 4.6: Mentor timing simulation - Clock speed slow enough 39
Figure A.1: Graph of Autocorrelation Function for Length 2 PN component. 43
Figure A.2: Graph of Autocorrelation Function for Length 7 PN component. 44
Figure A.3: Graph of Autocorrelation Function for Length 11 PN component. 44
Figure A.4: Graph of Autocorrelation Function for Length 15 PN component. 45
Figure A.5: Graph of Autocorrelation Function for Length 19 PN component. 45
Figure A.6: Graph of Autocorrelation Function for Length 23 PN component. 46
Figure A.7: Graph of Cross-correlation Function for Length 2 and 7 PN Components. 47
Figure A.8: Graph of Cross-correlation Function for Length 2 and 11 PN Components. 47
Figure A.9: Graph of Cross-correlation Function for Length 2 and 15 PN Components. 48
Figure A.10: Graph of Cross-correlation Function for Length 2 and 19 PN Components. 48
Figure A.11: Graph of Cross-correlation Function for Length 2 and 23 PN Components. 49
Figure A.12: Graph of Cross-correlation Function for Length 7 and 11 PN Components. 49
Figure A.13: Graph of Cross-correlation Function for Length 7 and 15 PN Components. 50
Figure A.14: Graph of Cross-correlation Function for Length 7 and 19 PN Components. 50
Figure A.15: Graph of Cross-correlation Function for Length 7 and 23 PN Components. 51
Figure A.16: Graph of Cross-correlation Function for Length 11 and 15 PN Components. 51
Figure A.17: Graph of Cross-correlation Function for Length 11 and 19 PN Components. 52
Figure A.18: Graph of Cross-correlation Function for Length 11 and 23 PN Components. 52
Figure A.19: Graph of Cross-correlation Function for Length 15 and 19 PN Components. 53
Figure A.20: Graph of Cross-correlation Function for Length 15 and 23 PN Components. 53
Figure A.21: Graph of Cross-correlation Function for Length 19 and 23 PN Components. 54
Figure B.1: 2code_combiner block diagram. 57
Figure B.2: clock_acq block diagram. 59
Figure B.3: code_combiner block diagram. 61
Figure B.4: code_convert block diagram. 63
Figure B.5: comp_rec_all block diagram. 65
Figure B.6: comp_rec_wctl block diagram. 67
Figure B.7: comp_recover block diagram. 69
Figure B.8: control_cnt block diagram. 71
Figure B.9: control_fsm block diagram. 73
Figure B.10: d_ff block diagram. 75

Figure B.11: d_latch block diagram. 77
Figure B.12: edge_true block diagram. 79
Figure B.13: int2vec block symbol. 81
Figure B.14: int2vec block parameters. 81
Figure B.15: int_dump block diagram. 89
Figure B.16: local_pn block diagram. 91
Figure B.17: local_pn2 block diagram. 93
Figure B.18: max_count block diagram. 95
Figure B.19: nodly block symbol. 97
Figure B.20: nodly block parameters. 97
Figure B.21: phase_error block diagram. 105
Figure B.22: pn_2comp block diagram. 107
Figure B.23: pn_comp_count block diagram. 109
Figure B.24: pn_delays block diagram. 111
Figure B.25: pn_gen2 block diagram. 113
Figure B.26: pn_generator block diagram. 115
Figure B.27: pn_regen block diagram. 117
Figure B.28: pn_source block diagram. 119
Figure B.29: pn_source2 block diagram. 121
Figure B.30: regen_ranging block diagram. 123
Figure B.31: regen_reset block diagram. 125
Figure B.32: rsum block diagram. 127
Figure B.33: single_pn block diagram. 129
Figure B.34: single_pn2 block diagram. 131
Figure B.35: vec2int block symbol. 133
Figure B.36: vec2int block parameters. 133
Figure B.37: vec_logic block symbol. 141
Figure B.38: vec_logic block parameters. 141
Figure B.39: vec_rotate block symbol. 151
Figure B.40: vec_rotate block parameters. 151
Figure B.41: vec_sca_logic block symbol. 159
Figure B.42: vec_sca_logic block parameters. 159
Figure B.43: xnor block diagram. 169
Figure C.1: 2comp_rec_test block diagram. 173
Figure C.2: acq_test.detail block diagram. 175
Figure C.3: acq_test.system block diagram. 176
Figure C.4: cmp_cnt_test block diagram. 178
Figure C.5: combiner_test block diagram. 180
Figure C.6: comp_rec_test block diagram. 182
Figure C.7: comp_rec_test2 block diagram. 184
Figure C.8: comp_rec_test3 block diagram. 186
Figure C.9: count_test block diagram. 188
Figure C.10: fsm_test block diagram. 190
Figure C.11: int2vec_test block diagram. 192
Figure C.12: local_pn_test block diagram. 194
Figure C.13: pn_gen_test block diagram. 196

Figure C.14: ranging_test block diagram. 198
Figure C.15: single_test block diagram. 200
Figure C.16: vec_logic_test block diagram. 202
Figure C.17: vec_rot_test block diagram. 204
Figure D.1: State Diagram of Control FSM. 205
Figure E.1: 7cnt schematic 213
Figure E.2: 11cnt schematic 213
Figure E.3: 15cnt schematic 213
Figure E.4: 19cnt schematic 214
Figure E.5: 23cnt schematic 214
Figure E.6: 7code schematic 216
Figure E.7: 11code schematic 216
Figure E.8: 15code schematic 217
Figure E.9: 19code schematic 217
Figure E.10: 23code schematic 218
Figure E.11: 7decode schematic 220
Figure E.12: 11decode schematic 220
Figure E.13: 15decode schematic 220
Figure E.14: 19decode schematic 221
Figure E.15: 23decode schematic 221
Figure E.16: 7gen schematic 223
Figure E.17: 11gen schematic 223
Figure E.18: 15gen schematic 224
Figure E.20: 23gen schematic 224
Figure E.19: 19gen schematic 225
Figure E.21: 7rec schematic 227
Figure E.22: 11rec schematic 227
Figure E.23: 15rec schematic 228
Figure E.24: 19rec schematic 228
Figure E.25: 23rec schematic 229
Figure E.26: 7reverse schematic 231
Figure E.27: 11reverse schematic 231
Figure E.28: 15reverse schematic 232
Figure E.29: 15reverse schematic 232
Figure E.30: 23reverse schematic 233
Figure E.31: 32cnt schematic 235
Figure E.32: and16 schematic 237
Figure E.33: cmp16 schematic 239
Figure E.34: combiner schematic 241
Figure E.35: control schematic 243
Figure E.36: generate schematic 245
Figure E.37: maxcount schematic 247
Figure E.38: negacc schematic 249
Figure E.39: reg16 schematic 251
Figure E.40: regenerate schematic 253
Figure E.41: test_system schematic 255

Figure F.1: 1rec_c schematic. 259
Figure F.2: 8reg_ec schematic. 261
Figure F.3: buf3 schematic. 263
Figure F.4: buf4 schematic. 263
Figure F.5: buf5 schematic. 263
Figure F.6: buf8 schematic. 264
Figure F.7: mux24 schematic. 266
Figure F.8: ttl00 schematic. 268
Figure F.9: ttl02 schematic. 268
Figure F.10: ttl04 schematic. 268
Figure F.11: ttl08 schematic. 268
Figure F.12: ttl10 schematic. 269
Figure F.13: ttl11 schematic. 269
Figure F.14: ttl20 schematic. 269
Figure F.15: ttl21 schematic. 270
Figure F.16: ttl32 schematic. 270
Figure F.17: ttl138 schematic. 270
Figure F.18: ttl161 schematic. 271
Figure F.19: ttl169 schematic. 271
Figure F.20: ttl194 schematic. 272
Figure F.21: ttl377 schematic. 272
Figure F.22: ttl175 schematic. 274
Figure F.23: ttl244 schematic. 276
Figure F.24: ttl257 schematic. 278
Figure F.25: ttl283 schematic. 280
Figure G.1: 11mux_1 schematic 281
Figure G.2: 11mux_2 schematic 282
Figure G.3: 11rec_1 schematic 283
Figure G.4: 11rec_2 schematic 284
Figure G.5: 15mux_1 schematic 285
Figure G.6: 15mux_2 schematic 286
Figure G.7: 15rec_1 schematic 287
Figure G.8: 15rec_2 schematic 288
Figure G.9: generate2 schematic 289
Figure G.10: regen1 schematic 290
Figure G.11: regen2 schematic 291
Figure G.12: regen3 schematic 292
Figure G.13: regen4 schematic 293
Figure G.14: regen5 schematic 294
Figure G.15: regenerate2 schematic 295

List of Tables

Table 2.1: Component codes for JPL PN ranging system. 20

Chapter 1

Introduction

1.1 General Overview

For many deep space missions, the location of a spacecraft must be known precisely, so that its trajectory can be determined and/or corrected. This location can be determined from a combination of ranging data and doppler data. The doppler data allows operators to determine in which direction the spacecraft is located, as well as in which direction it is moving. The ranging data allows a determination of the distance between the spacecraft and the earth to be made.

The ranging data is collected by sending ranging signals to the spacecraft, phase modulated onto a carrier signal. The signals are returned to the earth through the spacecraft transponder, and the ground station is able to determine the time delay caused by the distance the signal travels. This time delay, called the round trip light time (RTLT), can be used to calculate the spacecraft's distance from the earth. Using the current system for ranging, this distance can be measured to within approximately 10 meters.

1.2 Satellite Communication

There are three types of signals sent between the spacecraft and the ground station. Command signals are sent from the ground to the spacecraft, and are the means by which operators on the ground control the spacecraft. These signals include instructions, such as "turn on thruster number three for 0.14 seconds", "power up the imaging system in camera number four", or "start sending recorded data now".

Telemetry signals are sent from the spacecraft to the ground, and are the means by which scientific data, and spacecraft status data is returned to the operators. These signals include data, such as "thruster six fired successfully", "imaging system is on", or scientific data from an onboard instrument.

Ranging signals are sent both from the ground to the spacecraft, and from the spacecraft to the ground. As explained above, these signals are used to determine the exact location of the spacecraft. These ranging signals are the focus of the remainder of this paper.

1.3 Why regenerate?

The ranging system that is currently being used is a turn-around system. In other words, the ranging signals are simply amplified on the spacecraft before being retransmitted to the ground. An alternative to this system is to regenerate the ranging signals onboard the spacecraft. In this fashion, a unit on the spacecraft would recreate the ranging signals, and those recreated signals would be transmitted to the ground. In order for the accuracy of the system to be maintained, these signals must be generated either in-phase with the received ranging signals, or differing from them by only a constant phase shift.

By regenerating the ranging signals onboard the spacecraft, it is possible to dramatically improve the signal-to-noise ratio (SNR) of the retransmitted signal. This improvement, in turn, allows the original ranging signals to be transmitted with less power, even though the confidence in their measurement accuracy remains the same. By using less power for ranging, there is more power available for command and telemetry, which allows these signals to use higher data rates, reducing the amount of antenna time required for a given transmission. Since antenna time is a scarce resource, this reduction will benefit most deep space missions.

1.4 My Thesis Project

I have designed a regenerative ranging system, based on pseudonoise (PN) ranging (the different types of ranging are described in Section 2.1). It provides a significant improvement in the ranging SNR, at the expense of some added complexity onboard the spacecraft. The SNR improvement is realized mainly through reducing the noise bandwidth of the ranging signals. In the turn-around ranging system, the ranging signals are amplified via a band-pass amplifier with a bandwidth of approximately 1.5 MHz. These signals, then, have a noise bandwidth of 1.5 MHz. In the regenerative system, the noise bandwidth is reduced to the loop bandwidth of a phase-locked loop at the input of the system. This loop bandwidth is approximately 100 - 200 Hz. This difference causes a theoretical reduction by a factor of approximately 10^4, or 40 dB, in the noise power. The actual improvement in ranging SNR approaches this number.

In the next several chapters, some of the theory behind the ranging systems, a detailed account of the design methods employed, and some results and conclusions will be presented.

Chapter 2

Theoretical Analysis

2.1 Ranging

Currently, there are two main systems that perform ranging: sequential ranging, and pseudonoise (PN) ranging. The transponder used for deep space missions, and the ground stations that are part of the Deep Space Network (DSN), currently use sequential ranging.

2.1.1 Sequential Ranging

In the sequential ranging system [1], [5], [8], the ranging signal consists of a sequence of square waves with successively doubling periods, each called a component. The amount of time for which each component is transmitted is determined from mission requirements, and can vary over the lifetime of a mission. The main factor affecting these transmission durations is the integration time required in the ground station. The integration time, in turn, is affected by the signal-to-noise ratio of the signal received on the ground. The lower the signal-to-noise ratio falls, the longer the integration time must become. Currently, the options for increasing the signal-to-noise ratio for this ranging system are difficult to implement and/or otherwise undesirable.

The accuracy of the system is determined by the highest frequency component sent to the spacecraft. Since the phase of the returned signal is measured to about 1 part in 100, the accuracy of the system is approximately 1/100 of the wavelength of the highest frequency component [7].

The system also introduces an ambiguity with a period equal to that of the lowest frequency component. This ambiguity is created since the only measurements that are taken are the phases of each component. Since these phases will all be the same when the spacecraft is at distance of ζ, as they will be when the spacecraft is at distance of ζ plus any integer multiple of the wavelength of the lowest frequency component, there is ambiguity in the measurement. To eliminate this ambiguity, it is necessary to know, *a priori*, where the spacecraft is located, to within one wavelength of the lowest frequency component. This estimate can be made by either forcing the above wavelength to be larger than the distance the spacecraft will travel from the earth, or through using principles of celestial

mechanics, trajectory estimation, or some other method, to determine the spacecraft's location to within the necessary accuracy (one wavelength of the lowest frequency component).

2.1.2 PN Ranging

In the PN ranging system [10], [11], [12], which is currently being considered as a replacement for the sequential ranging system, the ranging signal consists of a pseudorandom stream of bits. This bit stream is chosen such that it will have a sharp peak in its autocorrelation function. In addition, to increase the speed at which the PN code can be acquired, the bit stream is composed of several shorter sequences (or components) that are combined using boolean logic. These components are chosen to minimize their respective cross-correlations. In this fashion, the phase of each component can be accurately determined by correlating the component with the entire combined sequence.

To allow for minimized cross-correlations, the component lengths must all be relatively prime. In addition, to facilitate easy acquisition, one component typically has length two, and is called the clock component. The remaining component sequences are chosen such that on average they maintain high and low logic levels for approximately equal amounts of time. The actual components chosen by the ranging group at JPL are shown in Table 2.1. Graphs of their auto- and cross-correlation functions are shown in Figures A.1 – A.21, in Appendix A.

Component Length	Component Sequence
2 (clock)	10
7	1110100
11	11100010110
15	111100010011010
19	1111010100001101100
23	11111010110011001010000

Table 2.1: Component codes for JPL PN ranging system.

These components are combined into a single PN sequence through a simple boolean function, shown in Equation (2.1).

$$PN = F(C_i) = C_2 \bullet (C_7 + C_{11} + C_{15} + C_{19} + C_{23}) + (C_7 \bullet C_{11} \bullet C_{15} \bullet C_{19} \bullet C_{23}) \quad (2.1)$$

Where C_n represents the PN component of length n, • represents logical AND, and + represents logical OR.

This combination results in a 6.25% reduction in the power of the clock signal. This power is spread into sidebands around the frequency of the clock component, producing the spectrum shown in Figure 2.1. The spectrum of the pure clock component is shown in

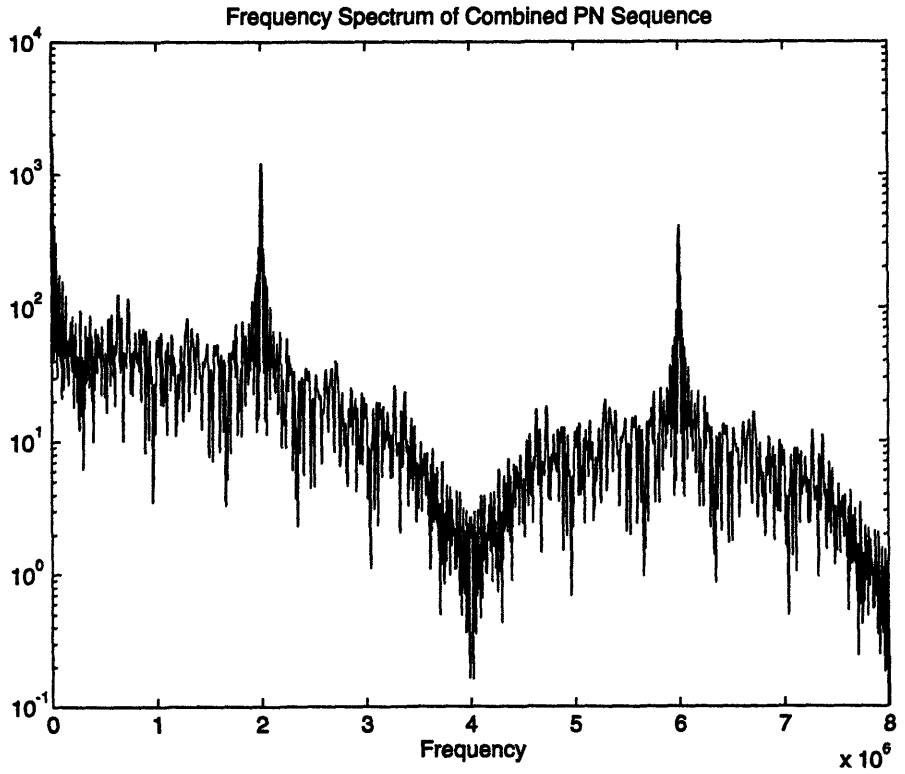

Figure 2.1: Spectrum of combined PN sequence.

Figure 2.2, for reference. In the time domain, the combined PN sequence appears very similar to the clock component, except that the combined sequence is missing a few pulses. In other words, the combined sequence will alternate between 1 and 0, except that there will be a few instances of three 1's in a row, or three 0's in a row. Both the clock component and the combined sequence are shown in the time domain, in Figure 2.3.

The accuracy of the PN system is comparable to that of the sequential system, since both systems depend on the highest frequency present (the clock component, in the case of the PN system) for taking their most accurate measurement. Both systems allow the use of up to 2 MHz clock components. The sequential system, however, currently uses only a

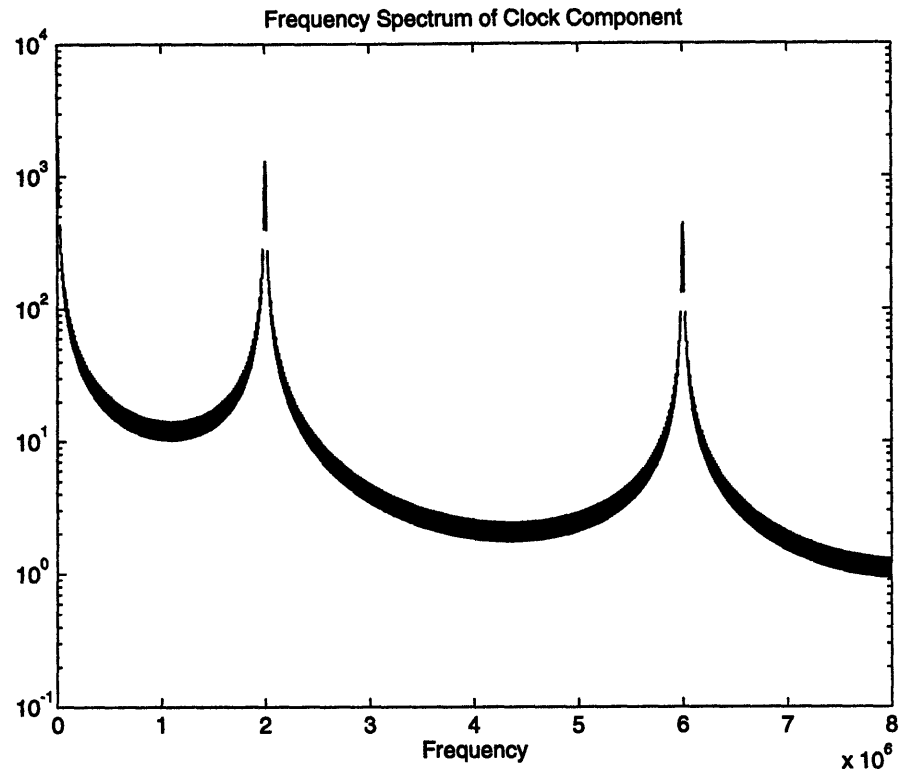

Figure 2.2: FFT of clock component.

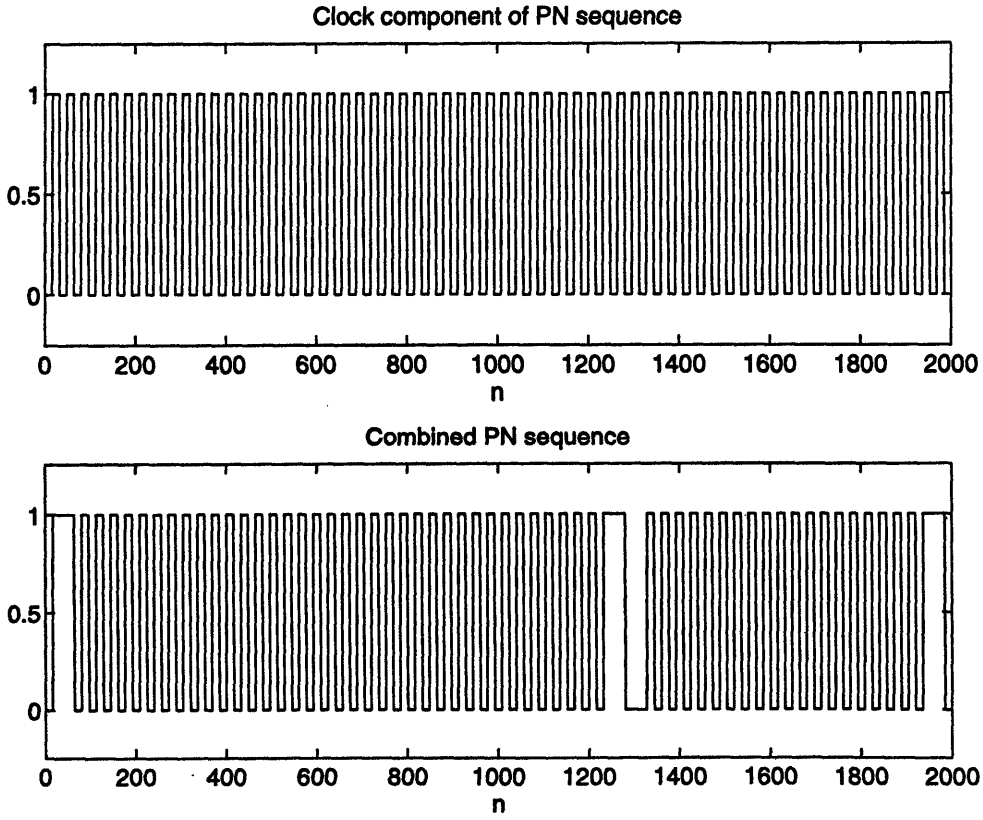

Figure 2.3: Clock component and combined PN sequence (time domain).

1 MHz clock component. Similar to the case of the sequential system, it is also possible for there to be ambiguity in the range measurement. In the PN system, however, this ambiguity results from the finite length of the PN sequence. The ambiguity in the measurement becomes the distance an EM wave travels in one cycle of the combined PN sequence.

Chapter 3

Design

3.1 Overview

The regenerative ranging system was designed as two separate units. One unit generates a PN ranging code, for testing purposes; the other performs the actual regeneration.

Both units were initially designed in Comdisco SPW (a signal processing CAD system), in order to develop functional algorithms. The units were developed concurrently, as there is a great deal of overlap in the Comdisco blocks used. Once the algorithm development was completed, the units were implemented in hardware using Mentor Graphics Design Architect, a schematic capture tool. After thorough simulation, the designs were laid out into Actel field programmable gate arrays (FPGAs) using the Actel Designer Advantage software. The FPGAs were then programmed, and wired onto a prototype circuit board for testing.

3.2 Comdisco Designs

The following sections (3.2.1 and 3.2.2) will explain only the general design of the generation and regeneration units. For a detailed description of the inputs, outputs, and function of each non-primitive Comdisco block used, as well as a copy of their lower-level block diagrams, see Appendix B. For a similar description of the systems used to test several of the Comdisco blocks, see Appendix C.

3.2.1 Generation Unit

The generation unit (shown in Appendix B as the *pn_source2* block, on Pages 120-121) is comprised of two main sections: The component generators, and the component combiner. The component generators are shift registers of length equal to the length of the component that they generate. They have load inputs that enable the bit sequences of a component to be initialized, and have both parallel and serial outputs. This design allows the blocks to be reused in the regeneration unit.

The component combiner implements a simple boolean function (Equation (2.1), in Section 2.1.2) that combines the 5 components and the clock into a single PN sequence.

In addition, the generation unit was designed to allow white Gaussian noise, of specified variance, to be added to the sequence before it is output. This additive noise provides for a more accurate model of what the regeneration unit will actually receive under normal operation.

3.2.2 Regeneration Unit

The regeneration unit (shown in Appendix B as the regen_ranging block, on Pages 122-123) consists of a clock recovery system, five independent component recovery systems, five matching component generators, and a component combiner circuit. A simplified block diagram of the system is shown in Figure 3.1.

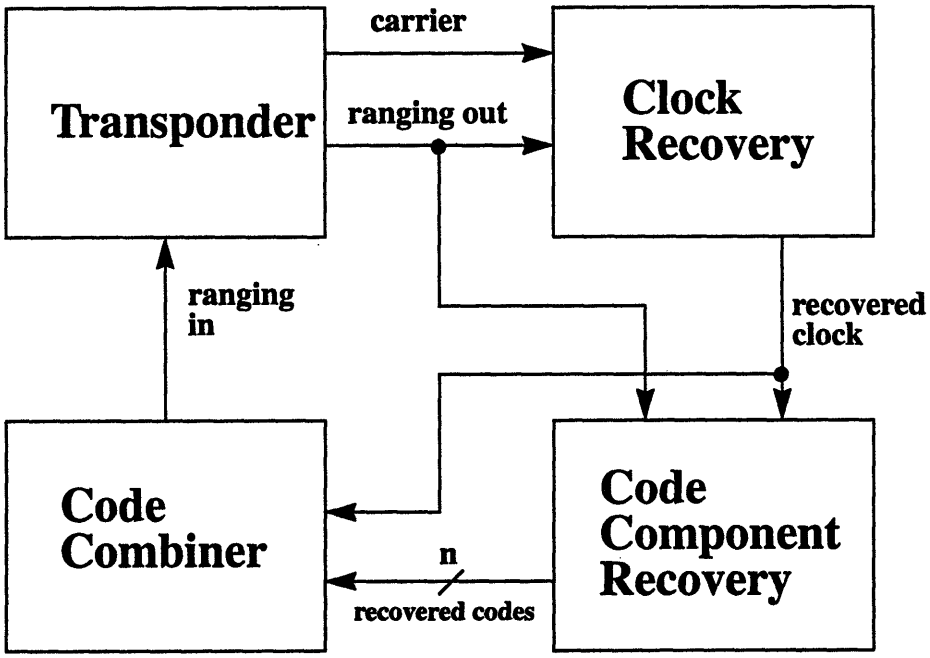

Figure 3.1: Simplified Block Diagram of Regeneration Unit (with transponder).

The clock recovery system (shown in Appendix B as the clock_acq block, on Pages 58-59) is implemented as a phase-domain model of a phase-locked loop. It consists of a phase error detector, a loop filter, and a signal generator. The phase error detector measures phase error by integrating the difference between an advanced version of the clock, and a delayed version of the clock, each multiplied by the incoming PN sequence. The integration is performed over one half the period of the clock. Since it is difficult to generate an advanced version of the clock, the comparison is actually performed on the clock, and a version of the clock that is delayed by $2\Delta T$. The recovered clock then becomes the original clock, delayed by ΔT.

Since the phase error detector performs an integration, its output is only valid at a lower sampling rate than its input. This difference in rates requires that the portion of the loop between the phase error detector and the repeat block run at a lower sampling rate than the rest of the system. This system is called a multirate simulation in Comdisco, and it requires careful handling of simulation parameters to insure correct output.

When scaling factors due to the sampling rate, and the method by which integrators are implemented in Comdisco are taken into account, the effective loop transmission has the form shown in Equation (3.1). This transfer function has a crossover frequency of $\omega_c = 89.6\frac{rad}{s}$, and a phase margin of 90.0°. It allows the loop to track step changes, as well as linear ramps, of the phase of the input clock, with zero error.

$$H(s) = 8.96 \times 10^{-2} \cdot \frac{(1000s + 1)}{s^2} \tag{3.1}$$

The five component recovery systems are all identical, except for their bit-width, and all five operate independently. Each recovery system consists of a local component generator, several correlators, logic to find the peak of the correlations, several registers, and control logic. The system computes the correlation between all possible phases of the locally generated component and the PN sequence that is input to the unit. It then finds the peak value of these correlations, and adjusts the phase of the local component so that this peak will correspond to a phase offset of zero.

The phase of the component generator in each component recovery system is then loaded into one of the five stand-alone component generators in the regeneration unit. These generators are the same as the ones used in the generation unit. The outputs of these generators are then fed into the component combiner circuit, which is identical to the combiner circuit in the generation unit. The output of the combiner circuit is the regenerated PN sequence that the system outputs.

A finite state machine (FSM) was designed to control the data paths that perform these functions, using the FSM design tools also used in 6.111 (Digital Systems Laboratory). The fsmc input file, and the equation and state output files are shown in Appendix D. A state diagram of the FSM is shown in Figure D.1.

3.3 Mentor Graphics Designs

The following sections (3.3.1 and 3.3.2) will explain only the general design of the generation and regeneration units. For a detailed description of the inputs, outputs, and function of each schematic building block used, as well as copies of their actual schematics, see Appendix E. For a similar description of some of the non-standard Actel macros used, see Appendix F.

3.3.1 Generation Unit

The generation unit (shown in Appendix E as the *generate* schematic, on pages 244-245) consists of five component generators, a component combiner, and an 8-bit adder. The component generators, as in the Comdisco design, are simply circular shift registers, sequentially outputting a component code that is input with a parallel load. Again, the component generator is a boolean logic function, obeying Equation (2.1), as stated above in Section 2.1.2. The output of the component generator then controls a multiplexor, letting the noiseless PN sequence take on an 8-bit value of either 96, or -96. An 8-bit noise input is then added to this value. The resulting quantity, after passing through an 8-bit D/A converter becomes the output.

3.3.2 Regeneration Unit

The regeneration unit (shown in Appendix E as the *regenerate* schematic, on pages 252-253) consists of five component recovery systems, five matching component generators, and a component combiner. The component generators and component combiner operate identically to those described above, for the generation unit (as shift registers, and a boolean function, respectively).

The regeneration unit takes its input as an 8-bit value that is the output of an A/D converter. The sampling rate on the A/D converter is set to four times the frequency of the clock component of the PN sequence. It is worth noting that the Mentor Graphics design of the regeneration unit does not contain a clock recovery circuit. This omission is purposeful, and is due to the fact that the clock recovery circuit is assumed to be implemented as an analog phase-lock loop before the A/D conversion takes place. In addition, this phase-lock loop outputs the sampling clock for the A/D converter (at four times the frequency of the PN clock component).

Each component recovery system consists of a local component generator, a set of correlators, and logic to find the peak of the correlation. In normal operation, the local generator will output a component, which will be correlated against the input PN sequence. This correlation is accumulated for several periods of the local component, to reduce the effect of noise present in the input PN sequence. The phase of the local generator is then adjusted so that the peak in the correlation moves to a phase difference of zero (between the local component and the input PN sequence). This component phase is then output to one of the stand-alone generators in the generation unit. After all the components have been acquired in this fashion, the output of the component combiner will be identically equal to the input PN sequence, except that the output sequence will be free from noise.

3.4 Actel FPGA Design

Due to the complexity of the design, FPGAs were chosen for the physical implementation. Actel FPGAs were used, since the required software had already been acquired. In addition, there were several Actel FPGAs available in the lab.

In order to use the Actel software, the Mentor Graphics designs had to be altered slightly. This alteration was necessary because the regeneration unit design wouldn't fit on one FPGA, and because the FPGAs require input and output buffers on all external signals. The revised designs are shown in Appendix G.

At this point in the design process, it was also decided that the final product would be a proof-of-concept, rather than an actual prototype. This decision permitted the Actel design to use only the shortest three PN components (lengths 7, 11, and 15) and the clock component, and to ignore the two longer components (lengths 19 and 23). This decision was made because each of the two longer PN components would require an additional three FPGAS to be used in the design, and it was determined that the additional gain in functionality would not justify the additional expense.

3.5 Physical Hardware Construction

After completing the design work, the generation and regeneration units were implemented as wirewrap circuit boards. The schematics for these boards are shown in Figures 3.2 and 3.3, respectively. It may be noted that there is no phase-lock loop present

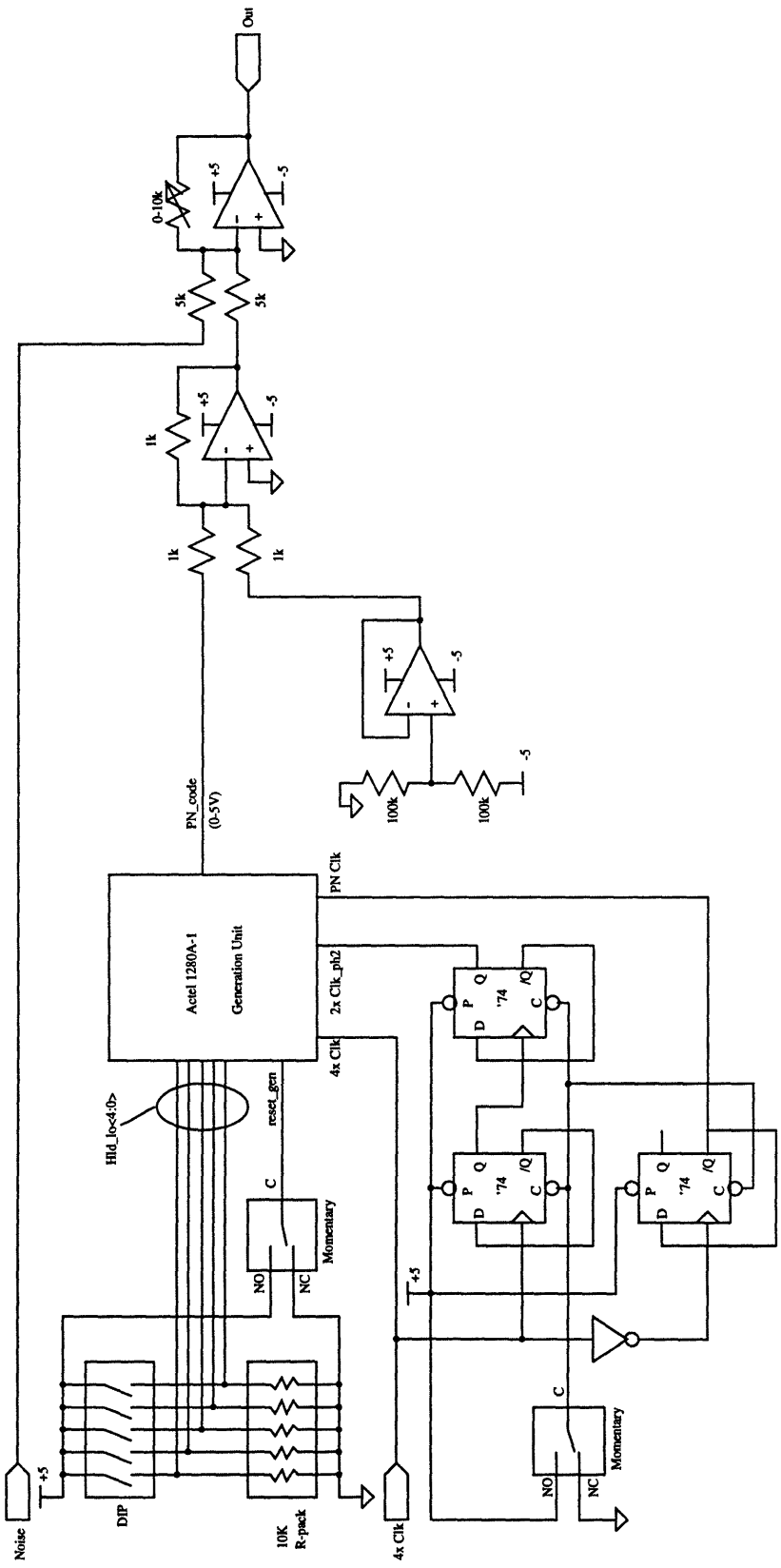

Figure 3.2: Generation Unit Wirewrap Board Schematic.

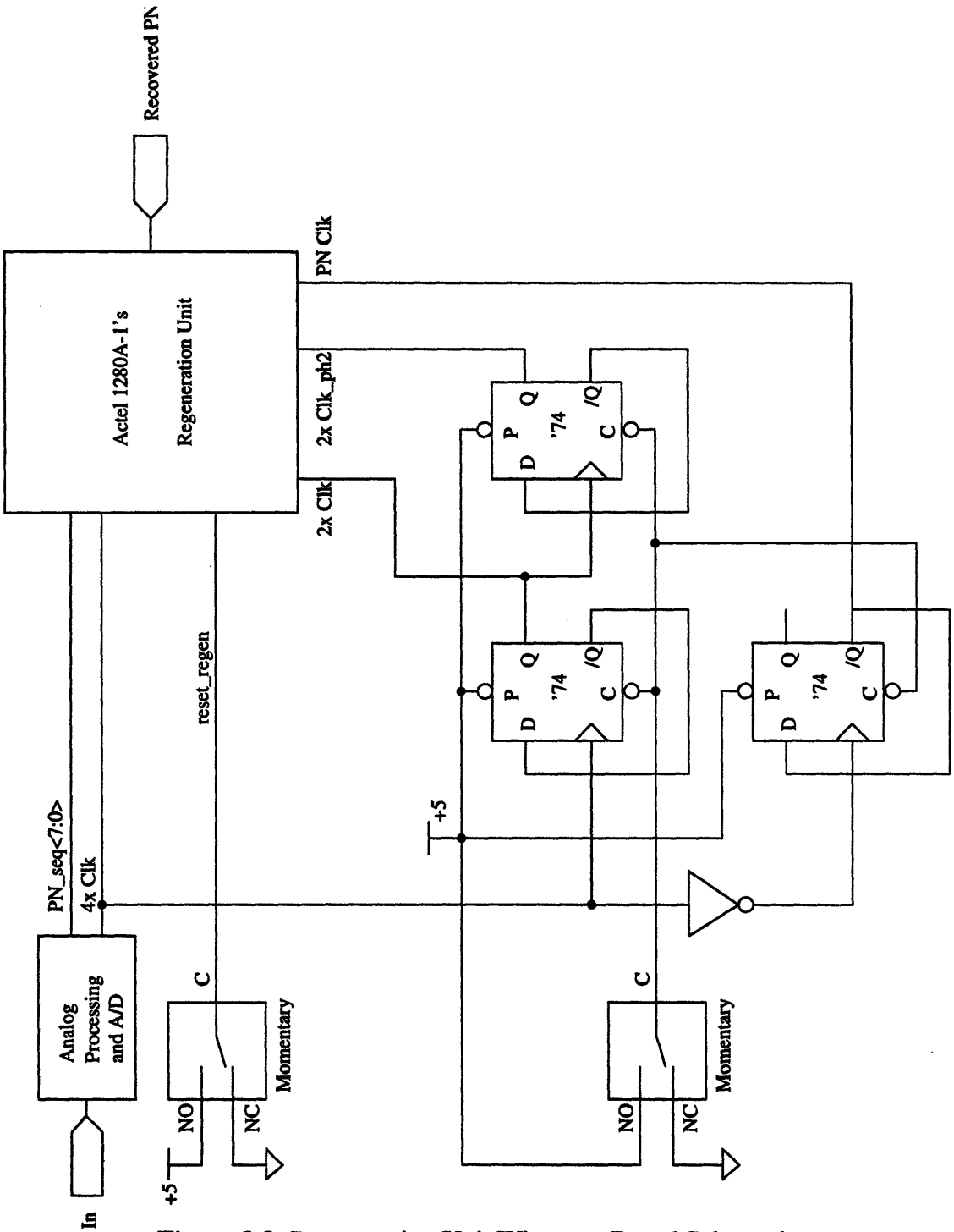

Figure 3.3: Regeneration Unit Wirewrap Board Schematic.

on the regeneration board. Since my expertise lies outside the area of analog phase-lock loop design, and since the design of a phase-lock loop is fairly simple for someone whose expertise includes this area, it was decided that space would be left on the regeneration board for a phase-lock loop, but it would not include one.

In order to attach the FPGAs, which have a 160-pin plastic quad-flat pack (PQFP) package, to the boards, a socket was found into which the FPGAs would fit. This socket, in turn had standard printed circuit board pins, so each socket was plugged into a set of female wirewrap header pins. In this fashion, it was possible to wirewrap to each pin of the FPGA.

Chapter 4

Results and Discussion

There are three sets of results; one for each of the different simulations or tests that were performed. These results include the output of several Comdisco simulations, and several Mentor Graphics simulations.

4.1 Comdisco Simulation Results

Once the algorithm design was completed in Comdisco, many simulations were performed, to ensure correct operation under all conditions. Output from two of these simulations is shown in Figures 4.1 and 4.2. From these figures, it can be seen that the regenerated output sequence locks onto the input after some finite time, then stays locked onto it for the duration of the simulation.

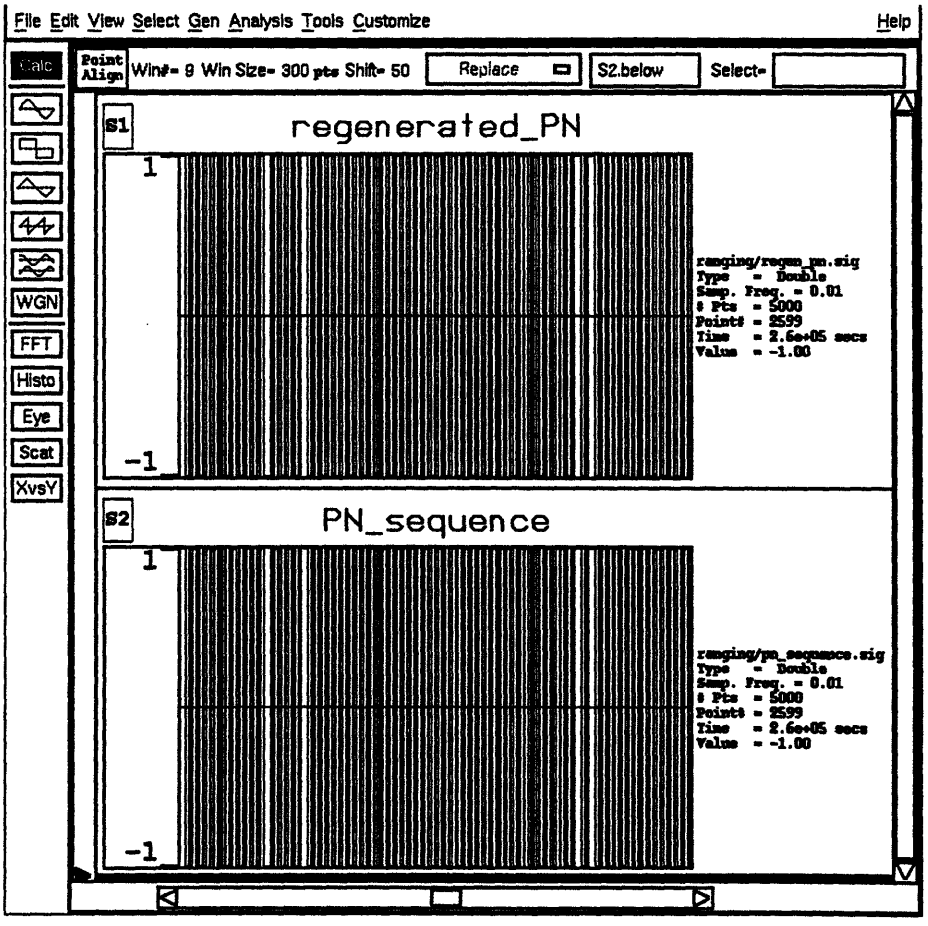

Figure 4.1: Comdisco simulation output - no noise on input.

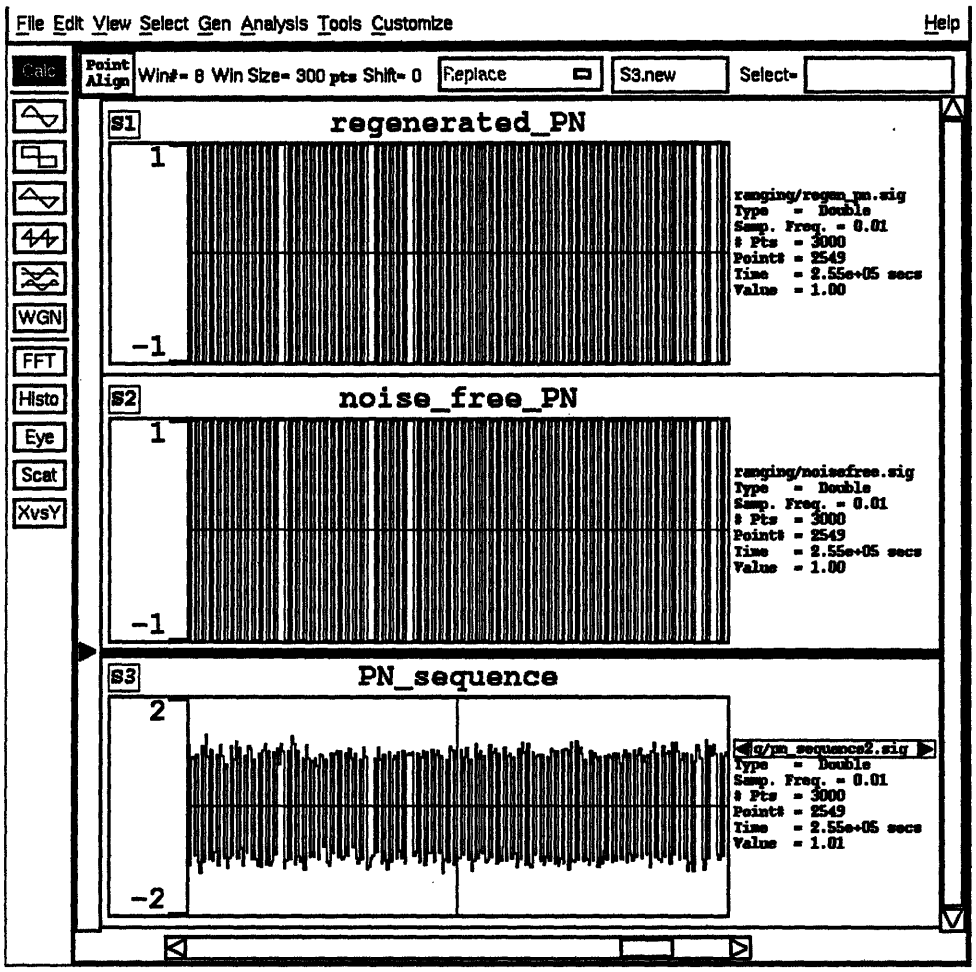

Figure 4.2: Comdisco simulation output - additive white noise on input.

4.2 Mentor Graphics Simulation Results

Similar to the case of Comdisco, above, the Mentor Graphics gate-level designs were thoroughly tested. Output from two of these simulations is shown in Figures 4.3 and 4.4. Again, it can be seen that the regenerated sequence locks onto the input, then stays locked.

In order to fully test the hardware, random test vectors were used. These vectors were generated by a small C program, shown in Appendix H (Section H.1). The output of the program was appended to a header file, also shown in Appendix H (Section H.2). This combination was then loaded into Mentor Graphics as an input vector. These vectors allowed the simulations to include true random noise, as well as random phase offsets to the generation unit. The output of the regeneration unit was considered correct when it matched the output of the generation unit (before noise was added).

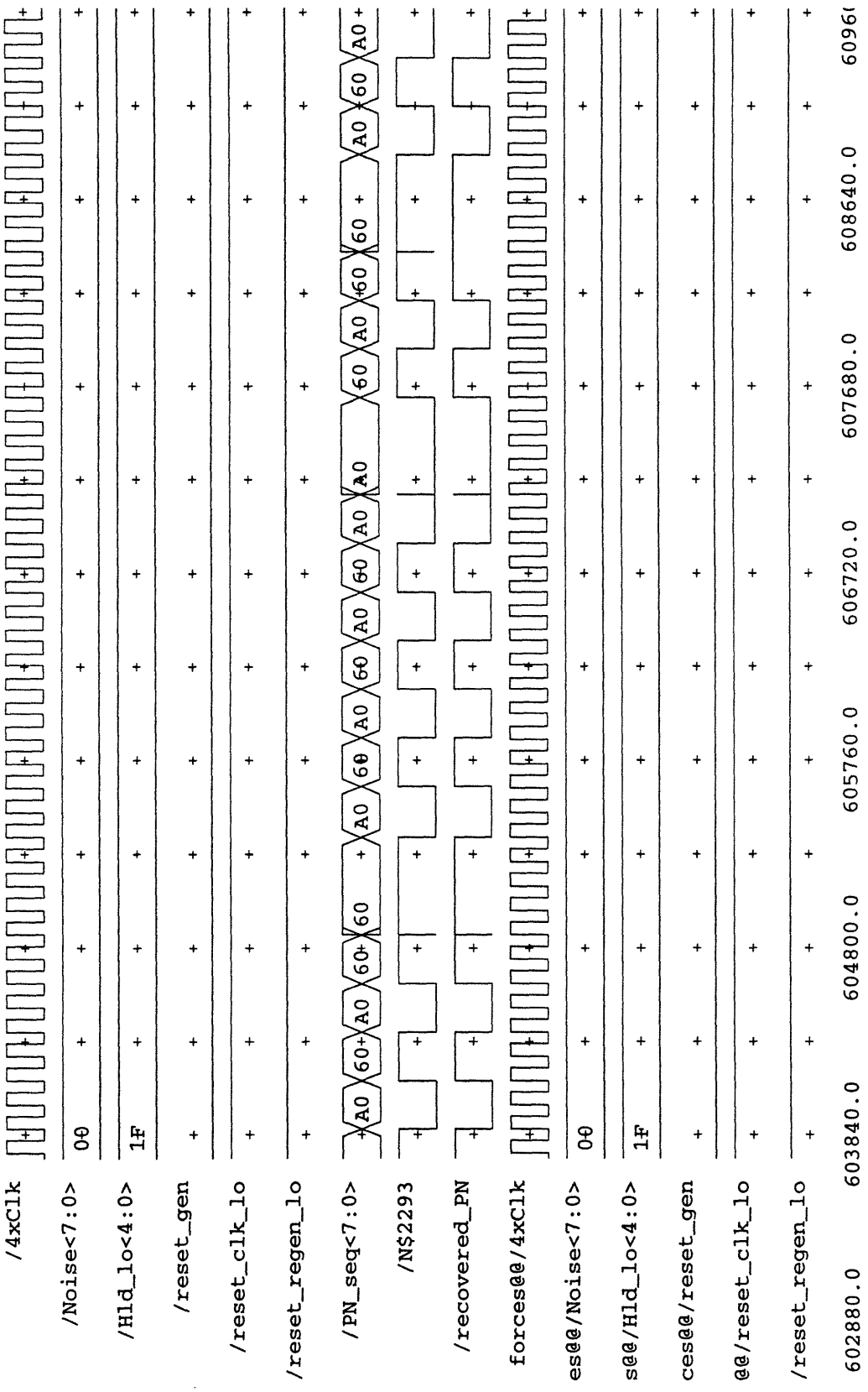

Figure 4.3: Mentor functional simulation without additive noise.

Figure 4.4: Mentor functional simulation with additive noise.

4.3 Post-layout Timing Simulation Results

Once the Actel FPGAs were designed and layout was completed, timing information was back-annotated into the Mentor Graphics simulations. The simulations were then performed again, in order to verify that the design still operated correctly. Output from two of these simulations is shown in Figures 4.5 and 4.6. The first simulation (Figure 4.5) shows that the design did not function correctly. This failure was traced to several timing violations inside the Actel FPGAs. Because of these violations, it was decided that the Actel FPGAs were too slow for an actual prototype. Since the goal of the project was to create a proof-of-concept, rather than an actual prototype, the clock frequency was simply lowered to the point that FPGAs would operate correctly. This slower speed was chosen to be 1 MHz. The second simulation (Figure 4.6) shows correct operation at this lower speed.

Test vectors for the original timing were generated using the same code as for the functional Mentor Graphics simulations. In order to create test vectors for the modified timing (reduced clock frequency), the vector generation code was modified. The revised code and the revised header file are shown in Appendix H (Sections H.3 and H.4, respectively).

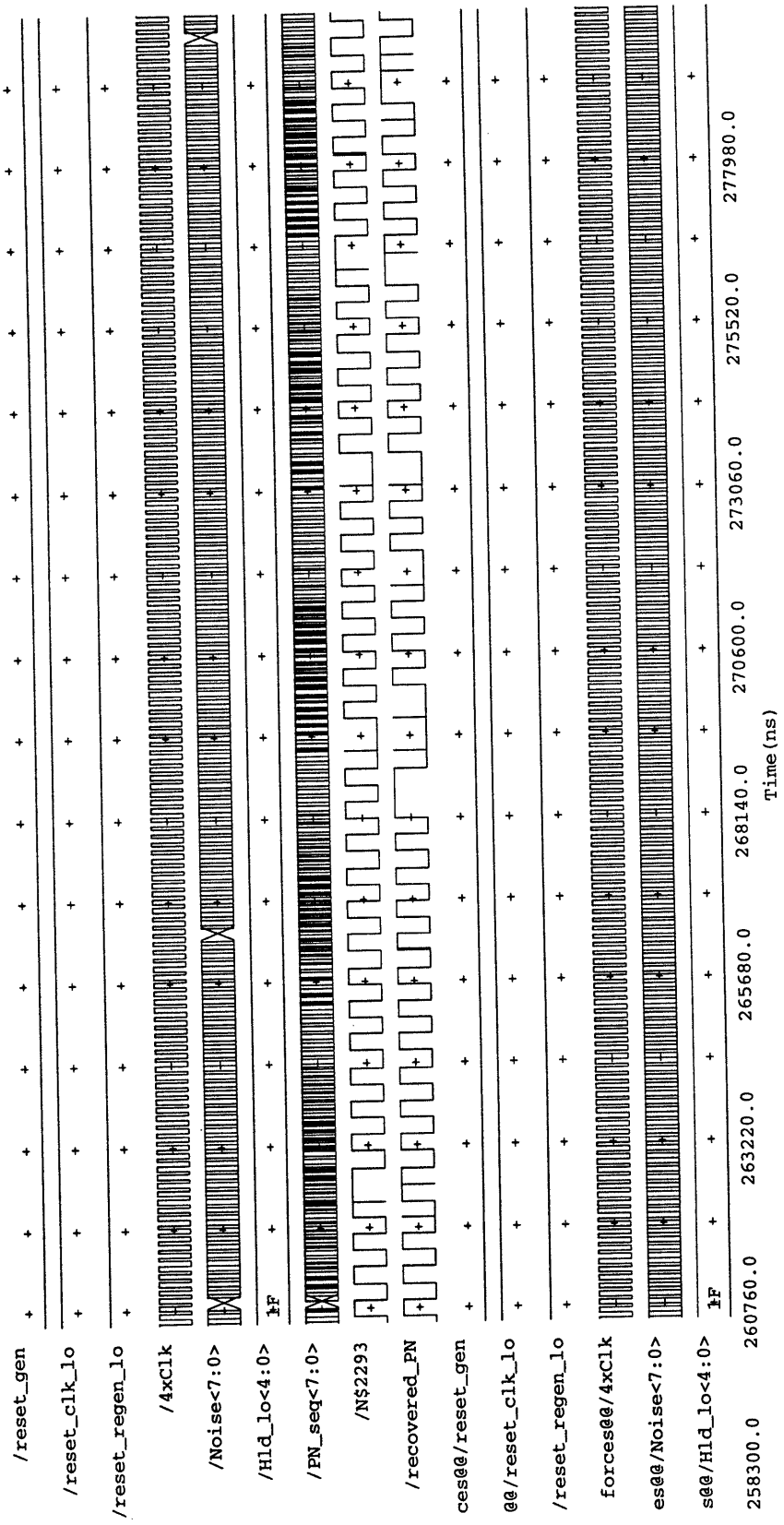

Figure 4.5: Mentor timing simulation - Clock speed too high.

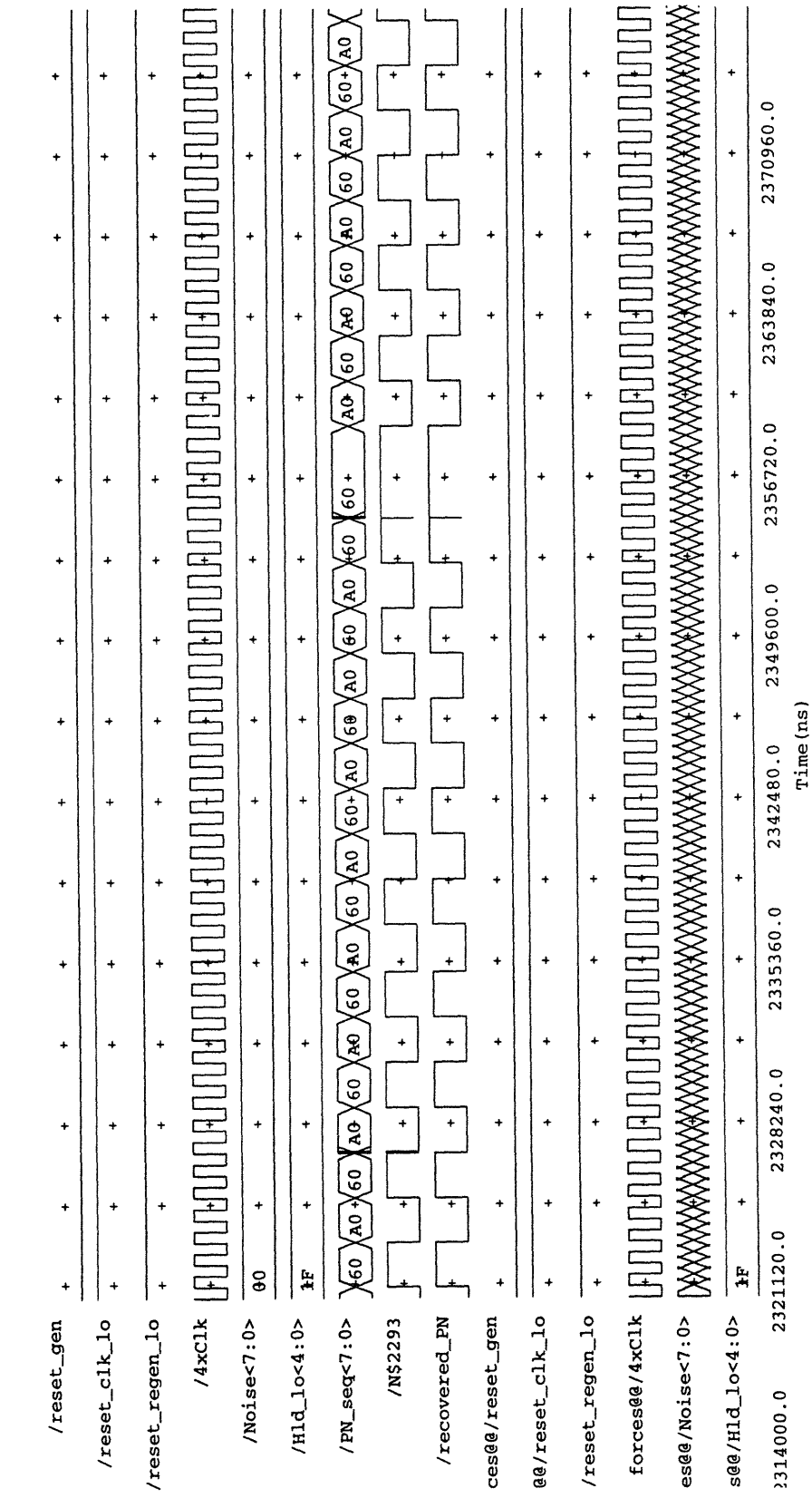

Figure 4.6: Mentor timing simulation - Clock speed slow enough.

Chapter 5

Conclusion

5.1 Summary

I have designed a regenerative ranging system to improve the SNR of ranging signals for deep space missions. This system approaches the maximum theoretical improvement of 40dB in ranging SNR. In doing so, the system allows higher data rates on the command and telemetry signals, which will benefit most deep space missions. Unfortunately, the Actel FPGAs that were chosen for the hardware implementation were not fast enough, and a proof-of-concept was designed, instead of an actual prototype. After slowing down the system, it functioned as desired, showing that regenerative ranging is possible.

5.2 Recommendations

Although a prototype was not built, the proof-of-concept performed well enough that further research should be conducted. Initially, the phase-locked loop that was left out of the hardware design should be designed and added to the system. The hardware design will then be complete.

Next, the breadboard model should be constructed, and thoroughly tested to determine if the actual hardware performs as well as the simulations. If this proves successful, the entire digital system should be redesigned as an application specific integrated circuit (ASIC). In this manner it can be made to operate at the required speed, and can be prepared for space qualification. Once this has been completed, the system will be able to fly on deep space spacecraft, reducing the amount of antenna time these spacecraft require for ranging operations. This reduction will lead to lower costs, and thus will benefit the entire unmanned space program.

Appendix A

Graphs of PN component Auto- and Cross-correlation functions

A.1 Graphs of Autocorrelation Functions

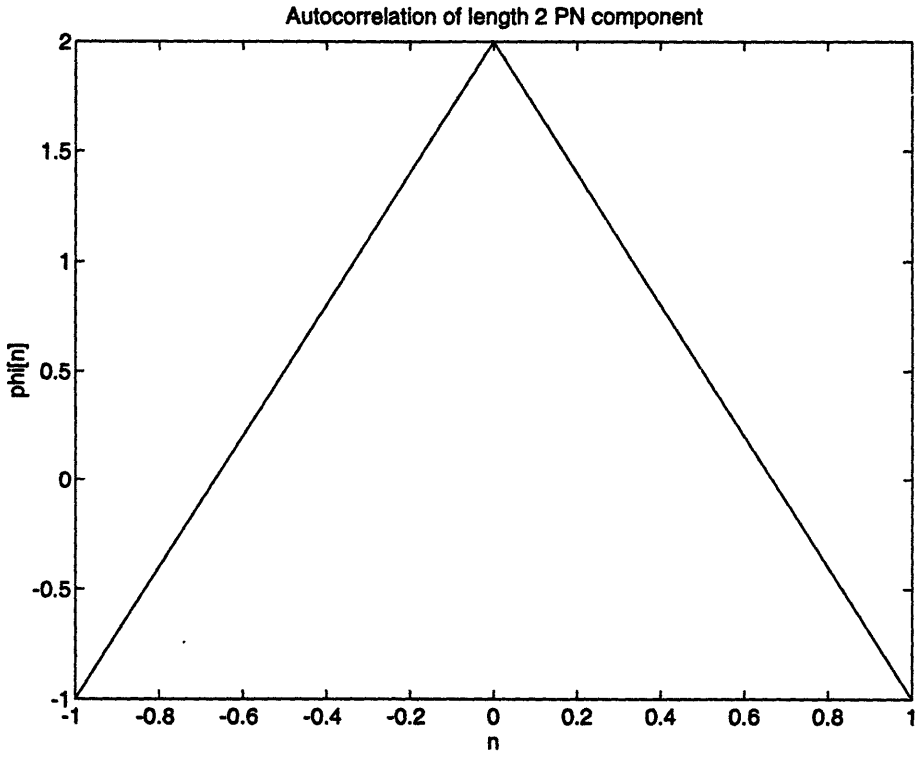

Figure A.1: Graph of Autocorrelation Function for Length 2 PN component.

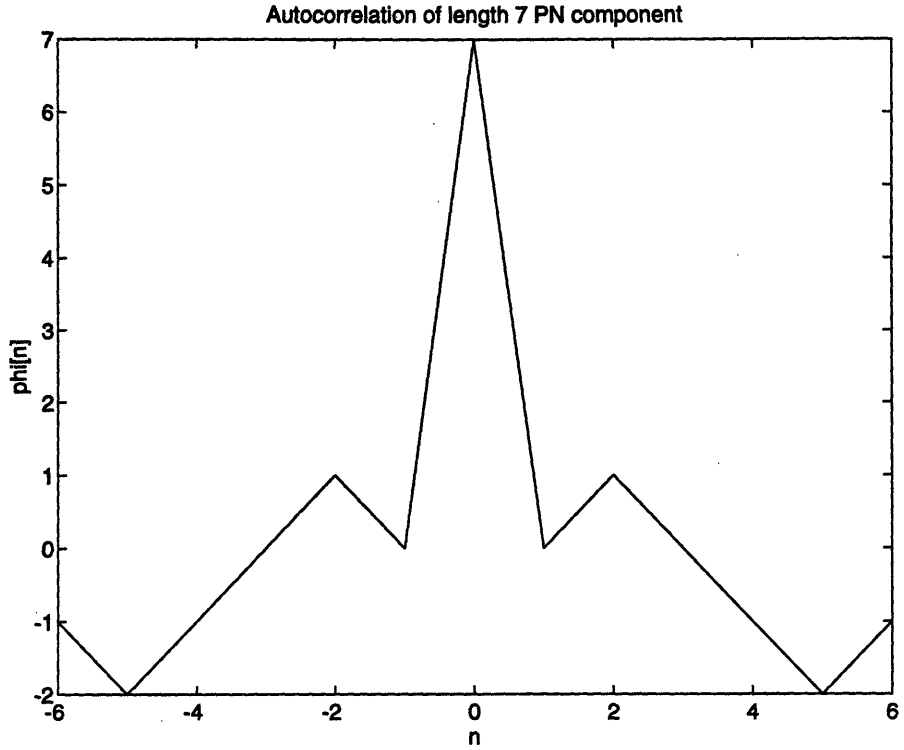

Figure A.2: Graph of Autocorrelation Function for Length 7 PN component.

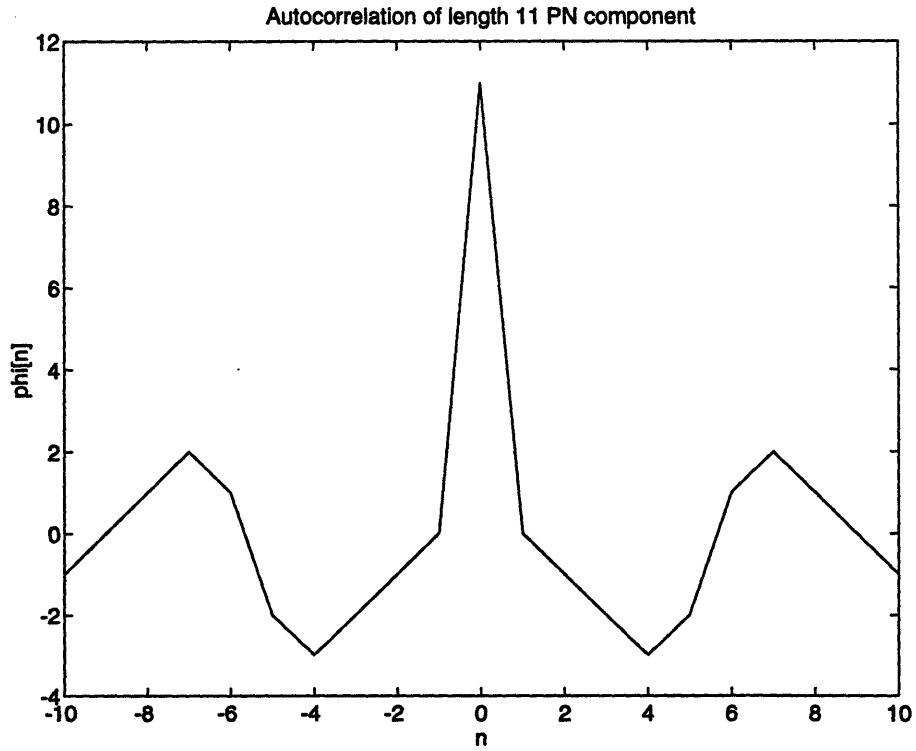

Figure A.3: Graph of Autocorrelation Function for Length 11 PN component.

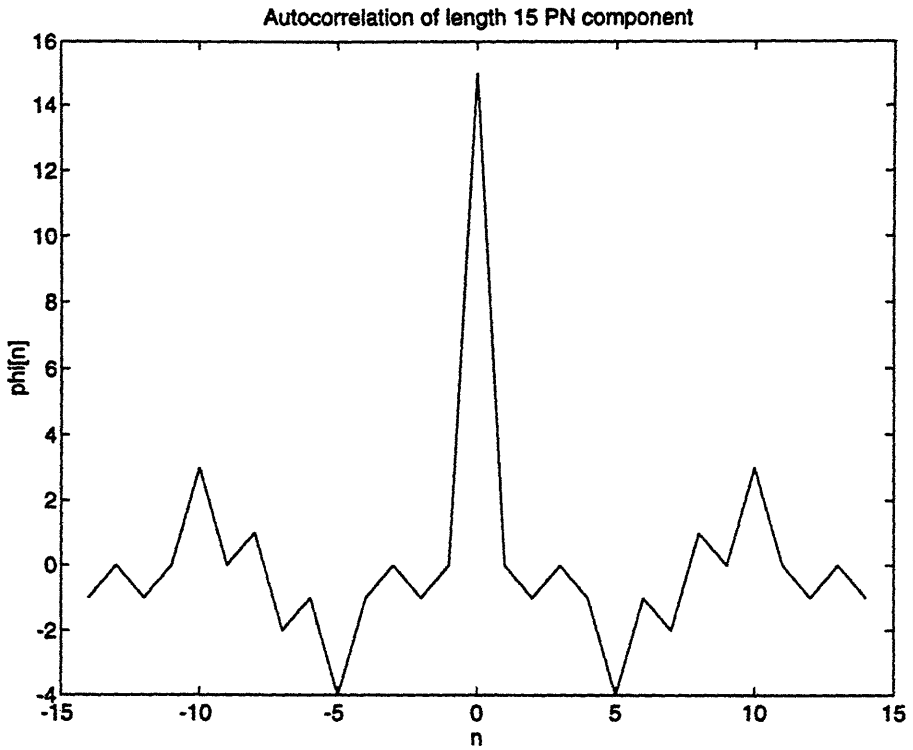

Figure A.4: Graph of Autocorrelation Function for Length 15 PN component.

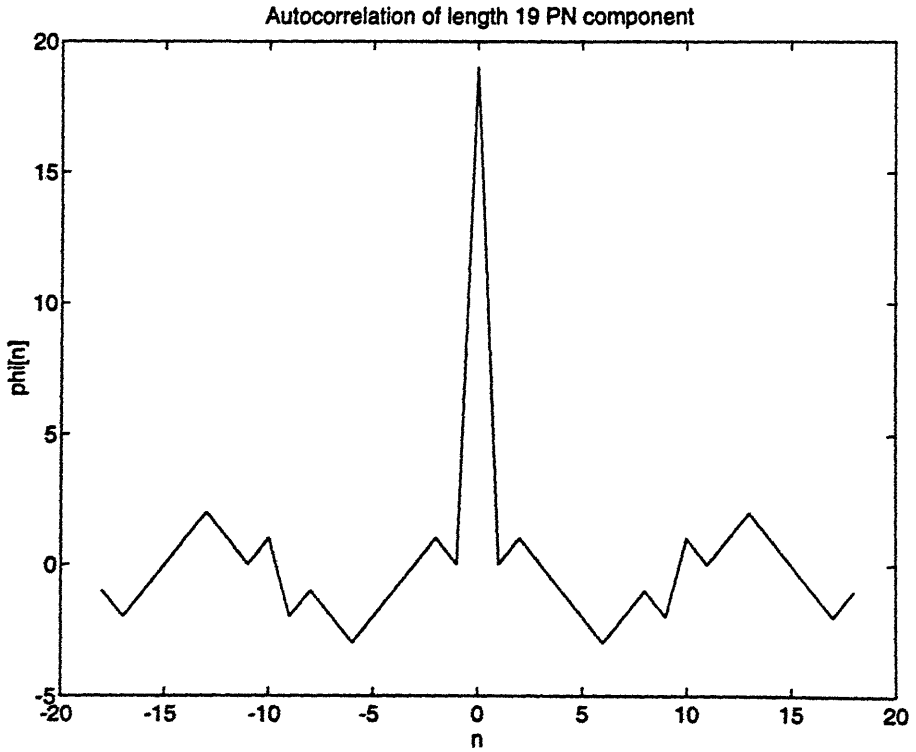

Figure A.5: Graph of Autocorrelation Function for Length 19 PN component.

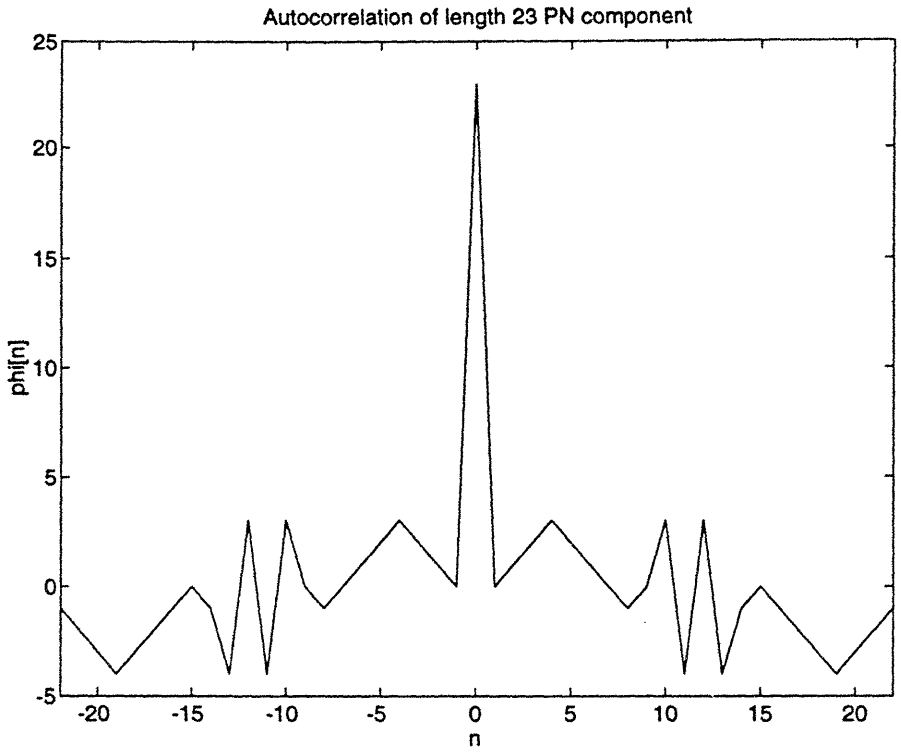

Figure A.6: Graph of Autocorrelation Function for Length 23 PN component.

A.2 Graphs of Cross-correlation Functions

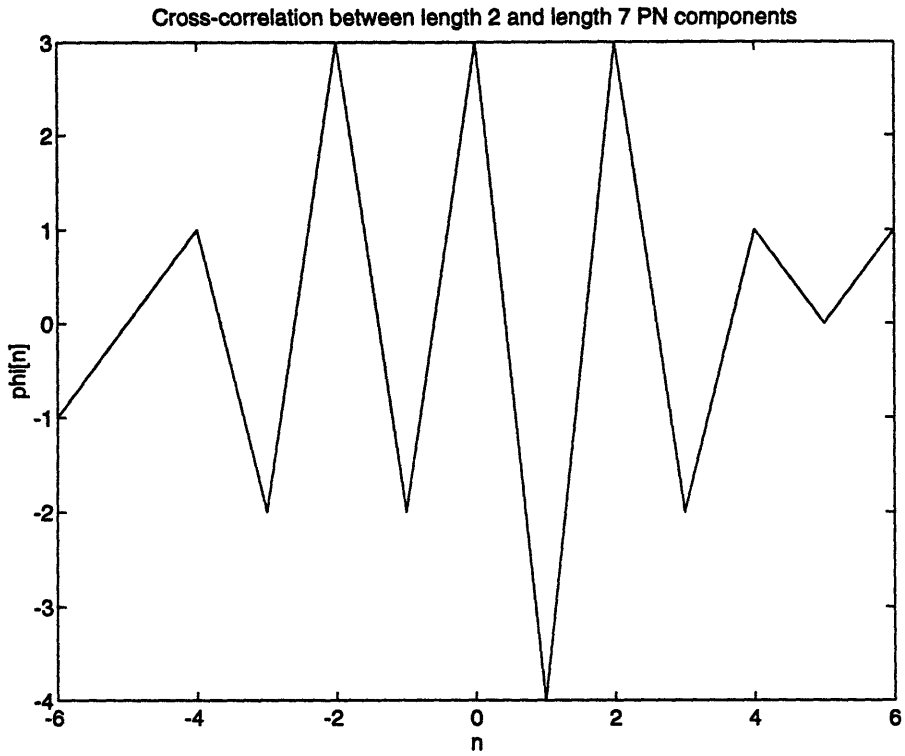

Figure A.7: Graph of Cross-correlation Function for Length 2 and 7 PN Components.

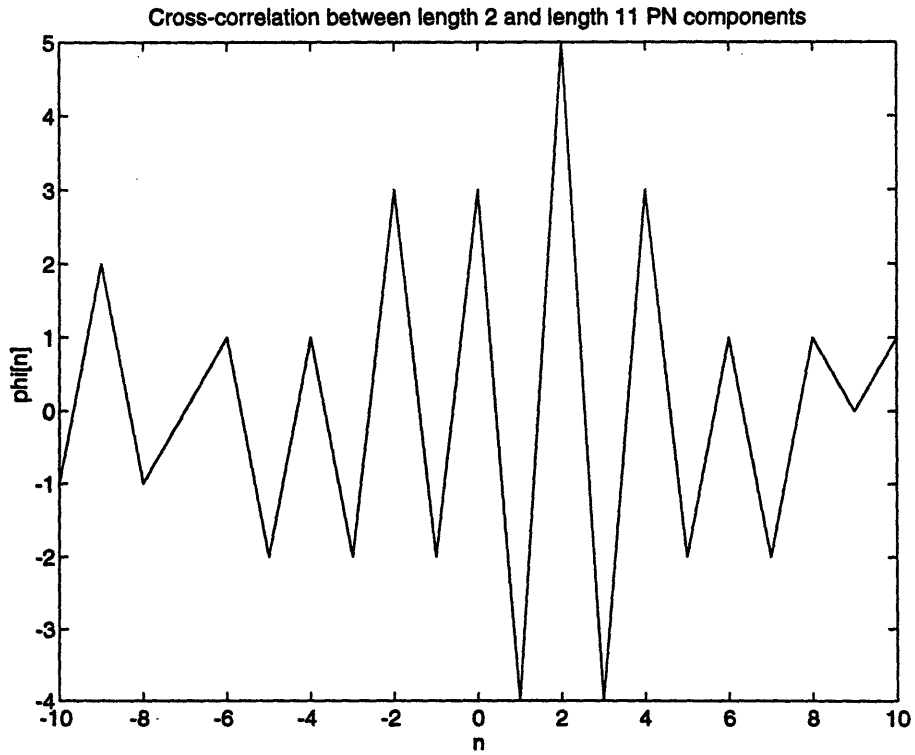

Figure A.8: Graph of Cross-correlation Function for Length 2 and 11 PN Components.

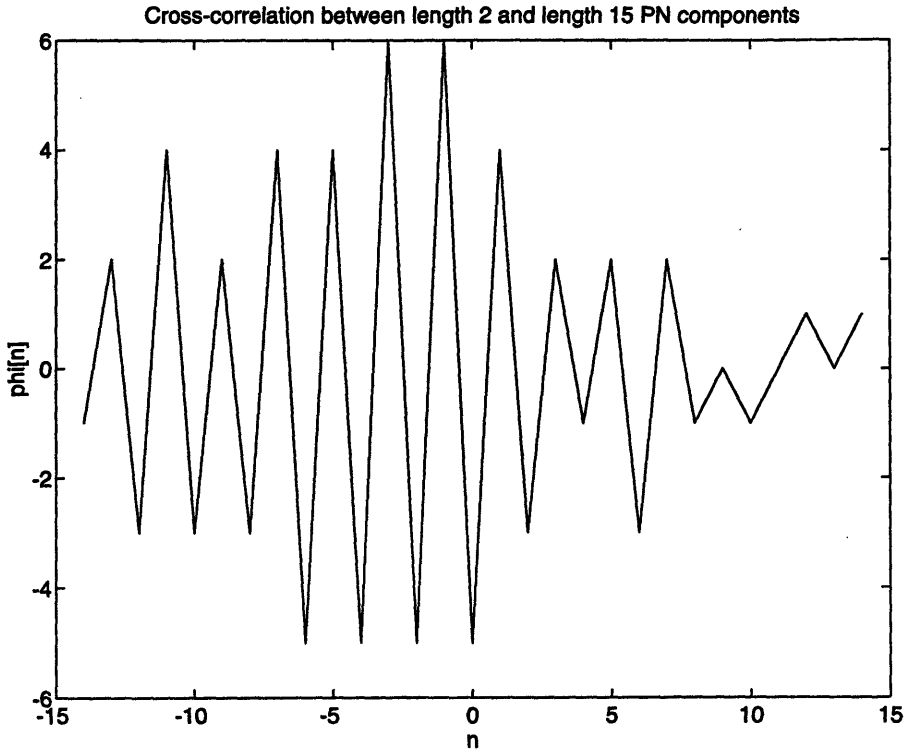

Figure A.9: Graph of Cross-correlation Function for Length 2 and 15 PN Components.

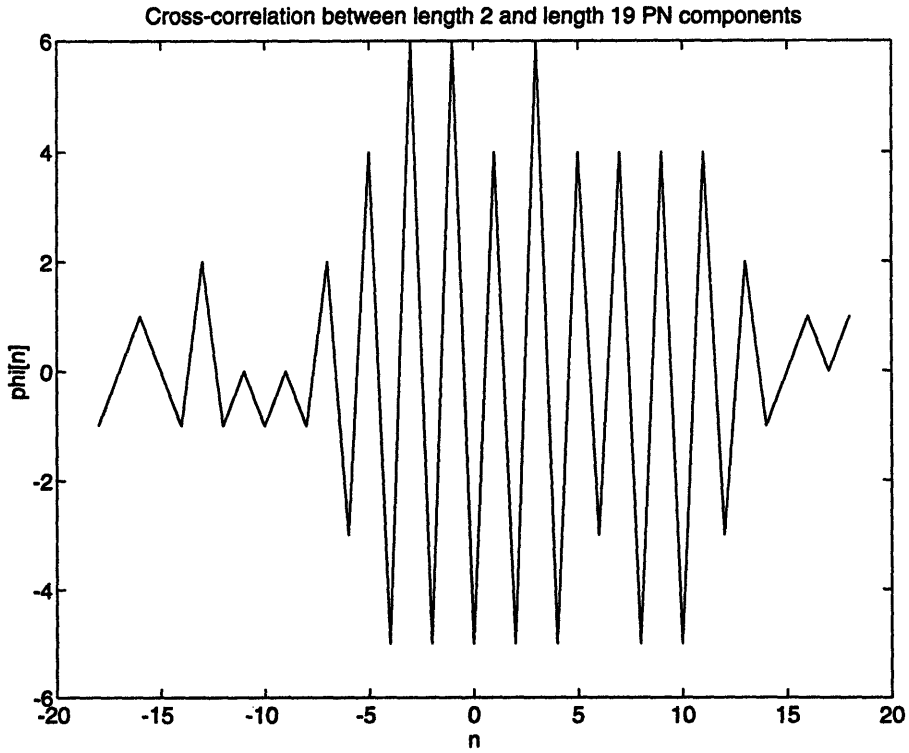

Figure A.10: Graph of Cross-correlation Function for Length 2 and 19 PN Components.

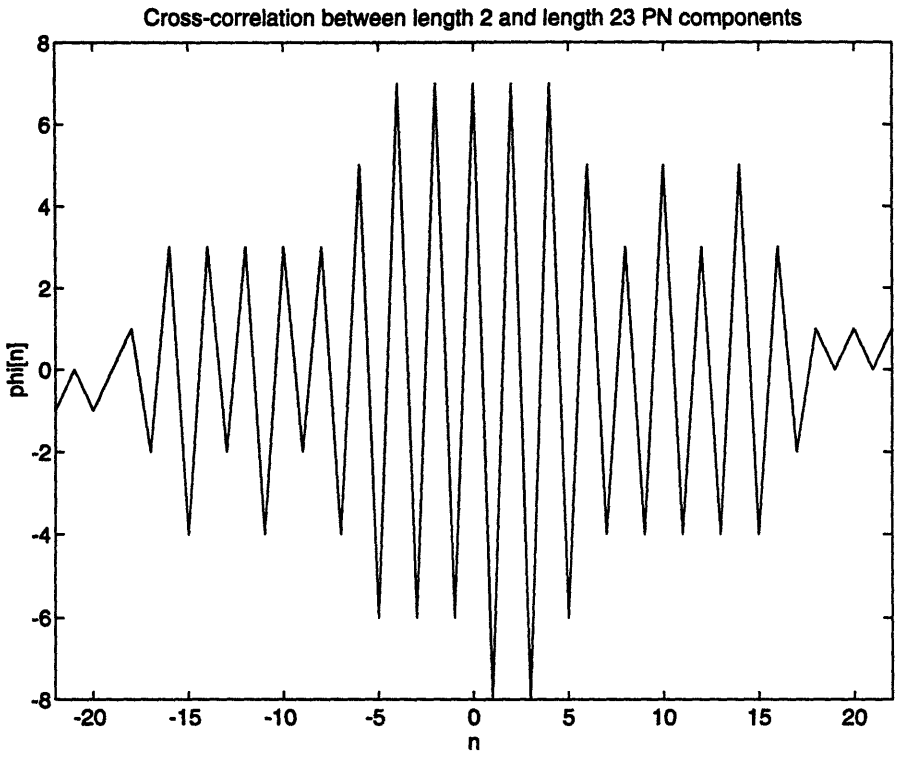

Figure A.11: Graph of Cross-correlation Function for Length 2 and 23 PN Components.

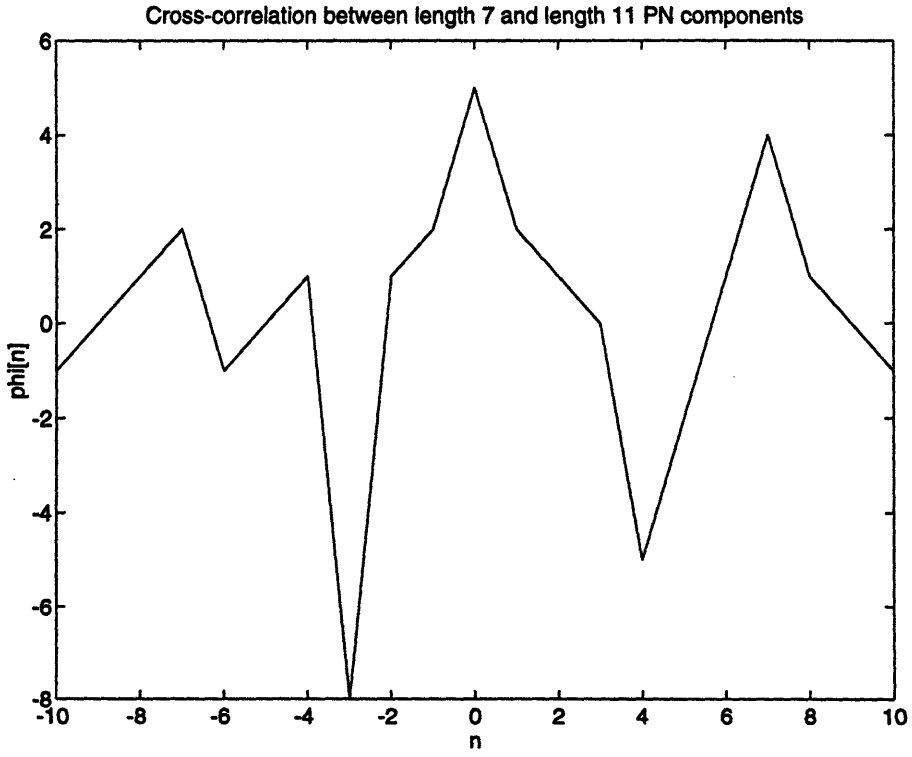

Figure A.12: Graph of Cross-correlation Function for Length 7 and 11 PN Components.

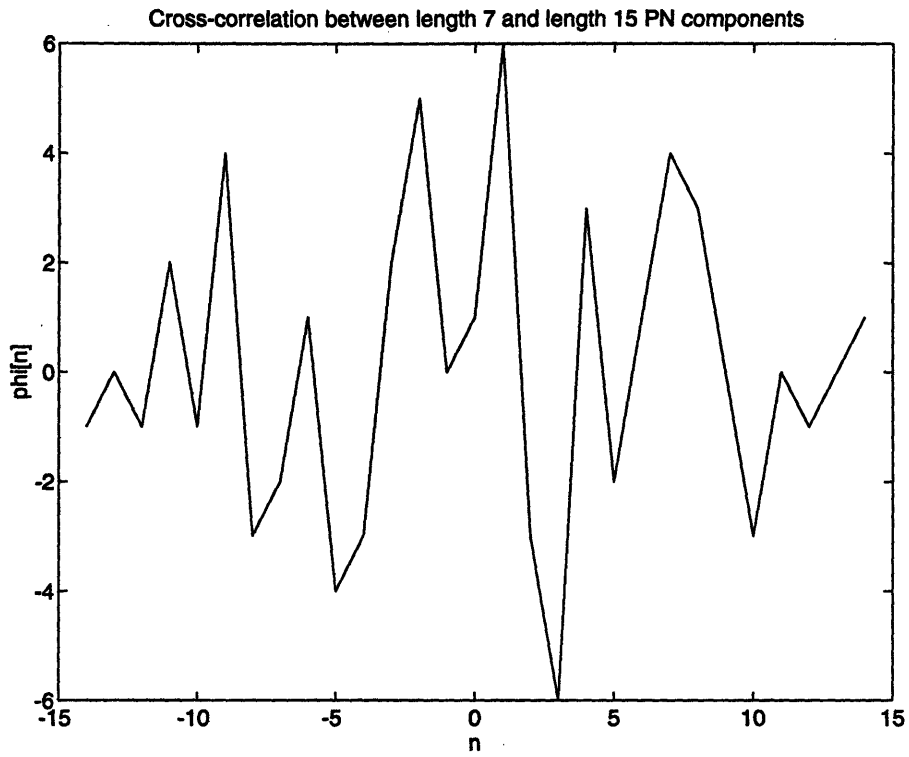

Figure A.13: Graph of Cross-correlation Function for Length 7 and 15 PN Components.

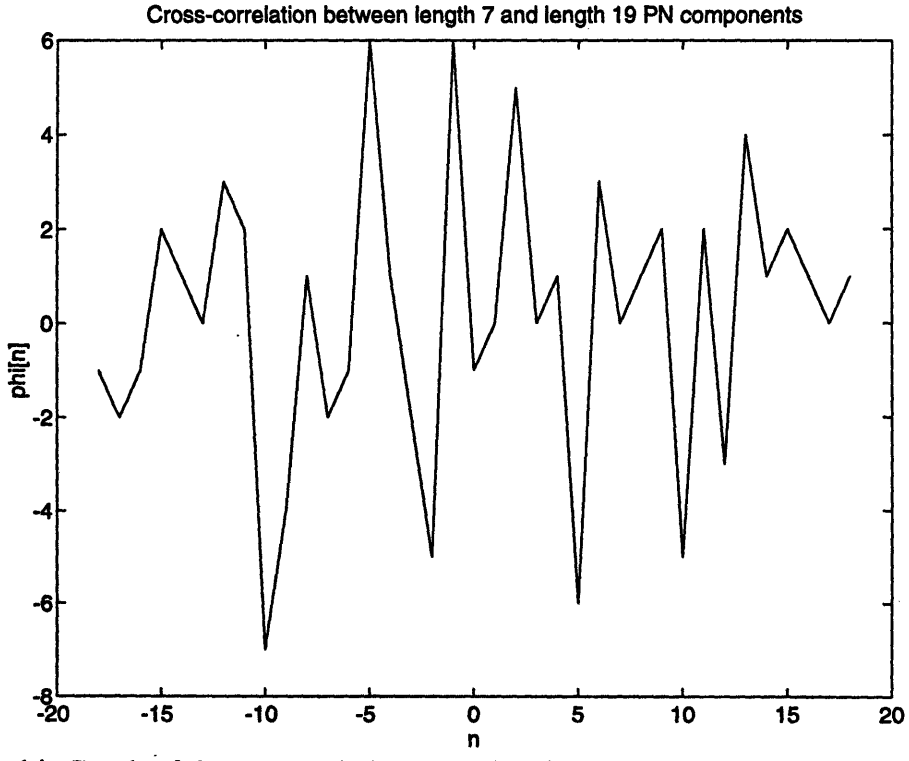

Figure A.14: Graph of Cross-correlation Function for Length 7 and 19 PN Components.

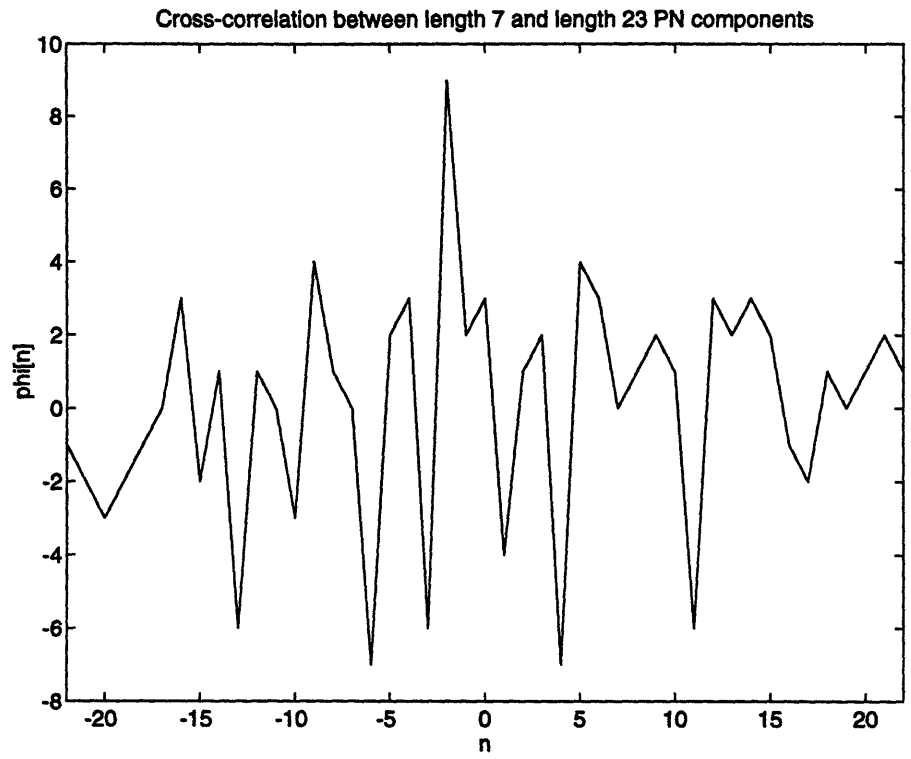

Figure A.15: Graph of Cross-correlation Function for Length 7 and 23 PN Components.

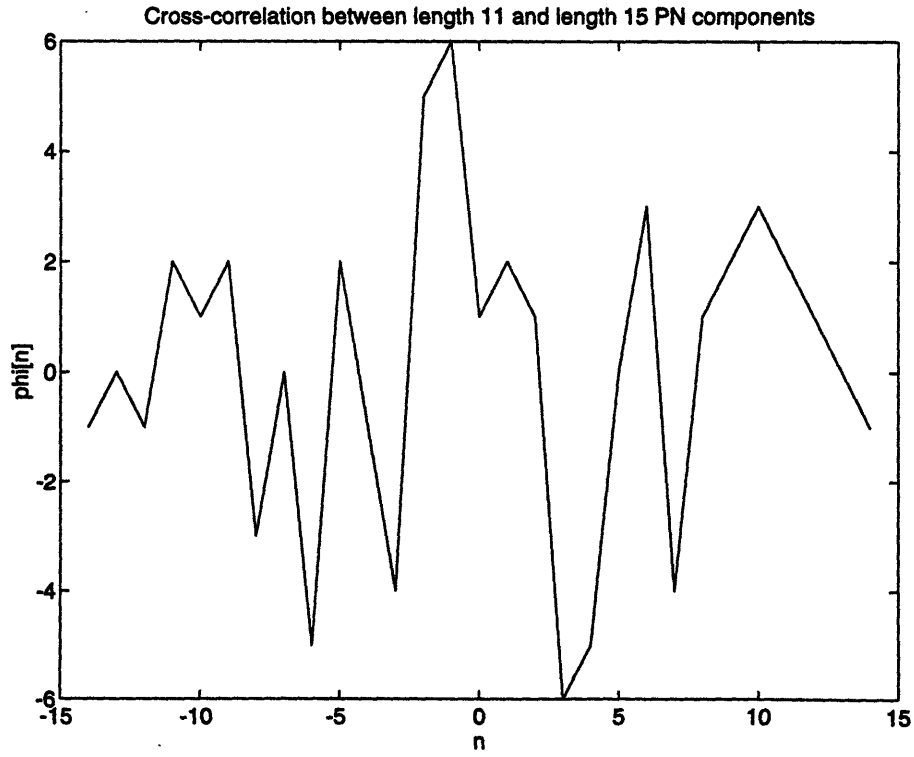

Figure A.16: Graph of Cross-correlation Function for Length 11 and 15 PN Components.

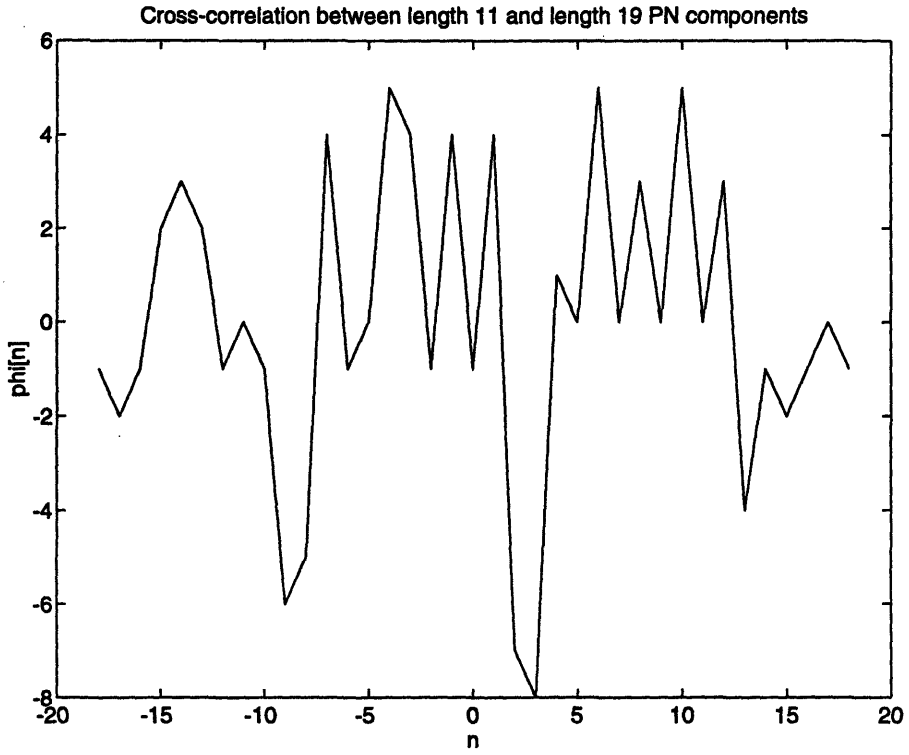

Figure A.17: Graph of Cross-correlation Function for Length 11 and 19 PN Components.

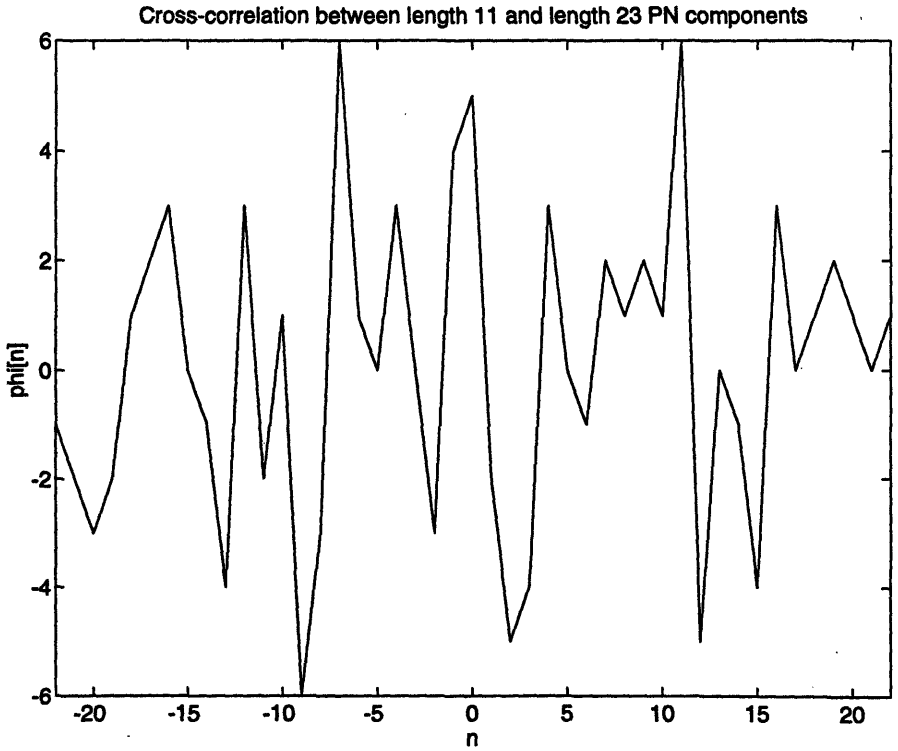

Figure A.18: Graph of Cross-correlation Function for Length 11 and 23 PN Components.

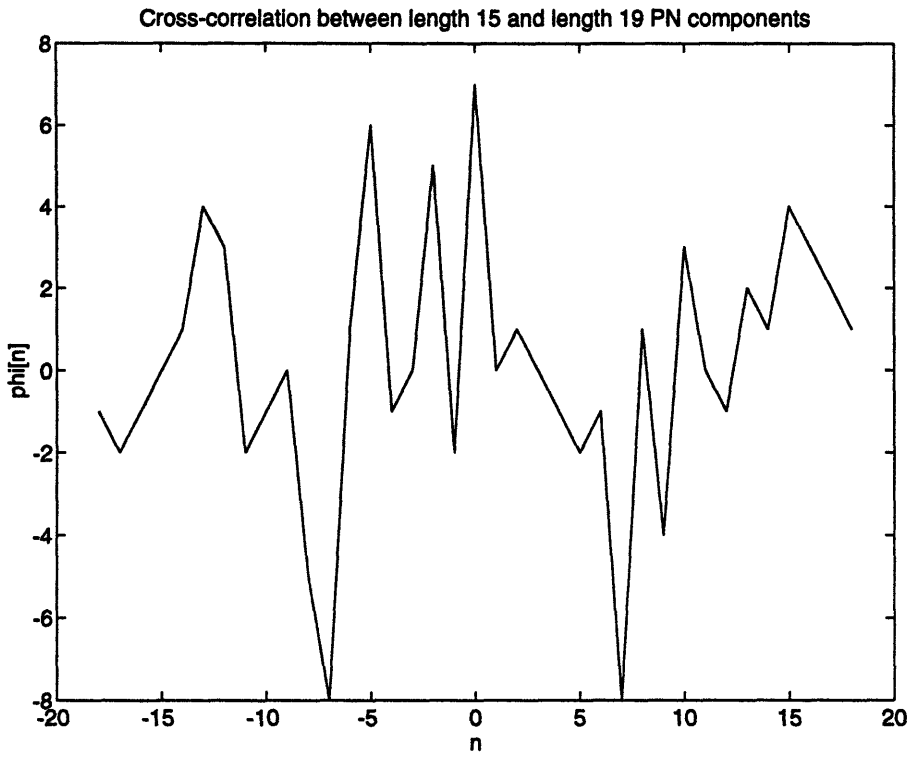

Figure A.19: Graph of Cross-correlation Function for Length 15 and 19 PN Components.

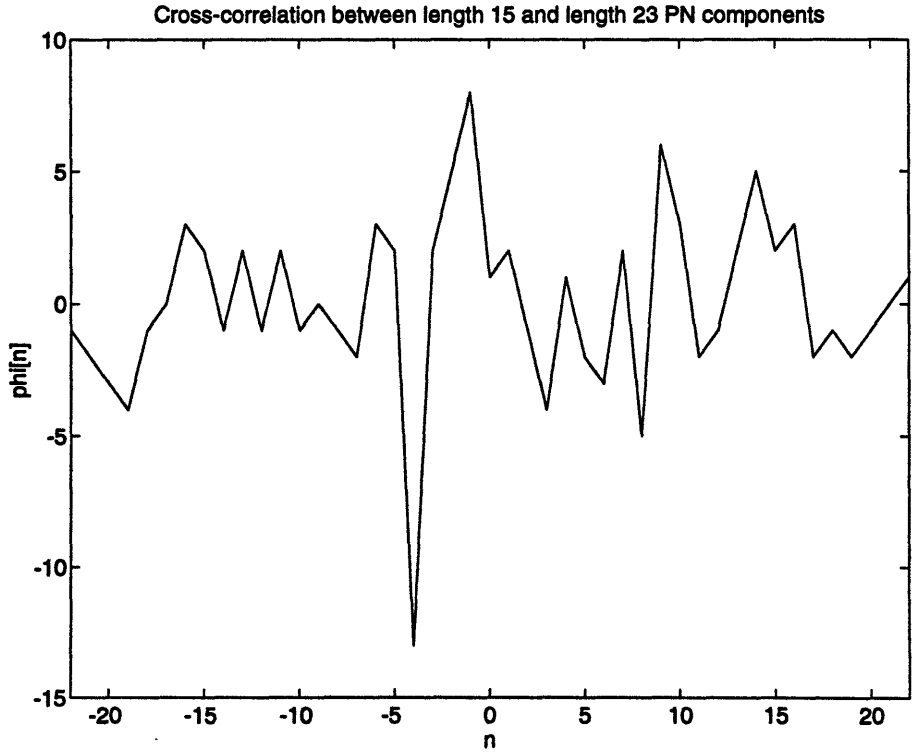

Figure A.20: Graph of Cross-correlation Function for Length 15 and 23 PN Components.

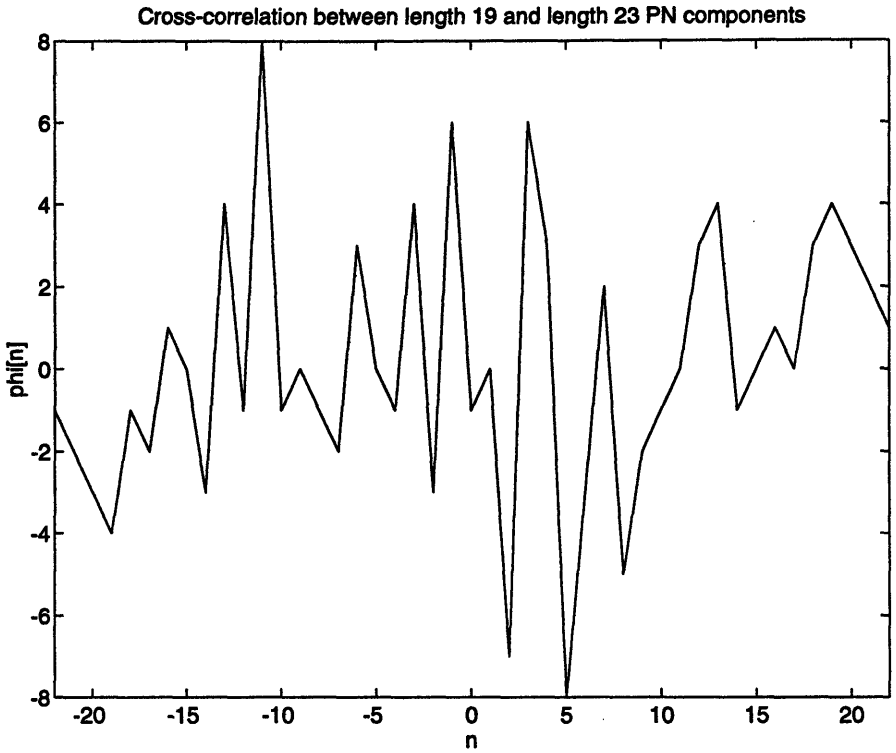

Figure A.21: Graph of Cross-correlation Function for Length 19 and 23 PN Components.

Appendix B

Comdisco Block Diagrams

This appendix contains descriptions of each Comdisco block used. The descriptions are formatted in the style of data sheets, with a listing of the inputs, outputs, and method of use for each block. Each description also contains a figure showing the underlying system of each block.

Block Name: **2code_combiner**

Synopsis:

Input Signals:

clock: The clock component of the PN code.

PN_codes: A vector containing all of the PN components, except the clock component.

Output Signals:

PN_sequence: The combined PN code sequence.

Parameters:

none

Functional Description:

This block functions exactly as the **code_combiner** block, with the exception that this block only combines two components of the PN code, plus clock. The **code_combiner** block combines five components of the code, plus clock.

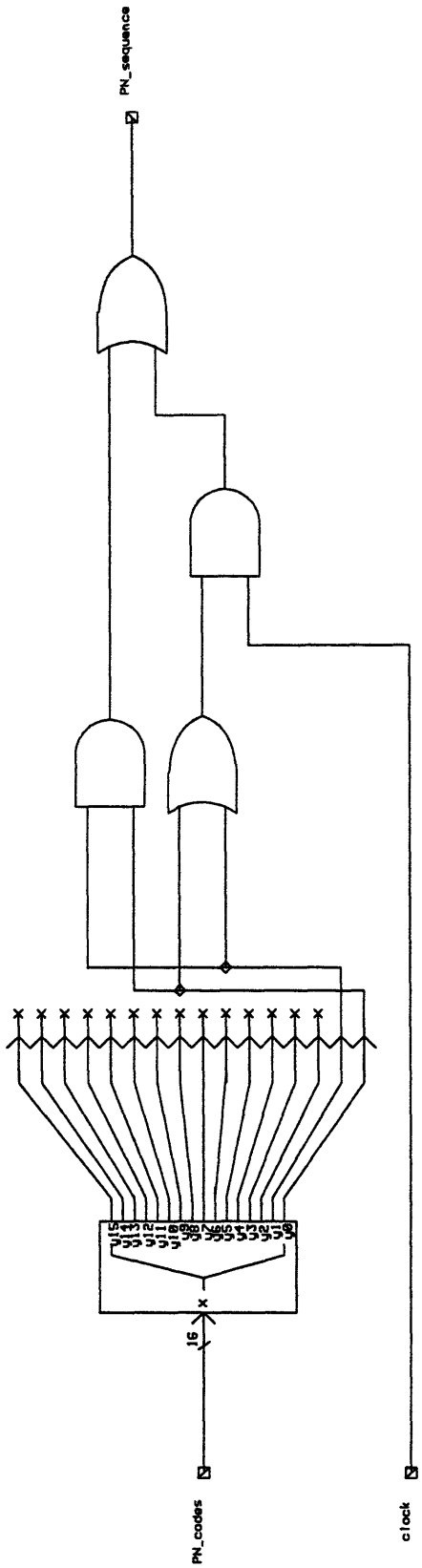

Figure B.1: 2code_combiner block diagram.

Block Name: clock_acq

Synopsis:

Input Signals:

PN_code: The pseudonoise sequence from which the clock is to be recovered.

Output Signals:

regen_clock: The recovered clock signal from the PN sequence.

Parameters:

c_freq: The carrier frequency from the receiver.
delta_t: Half of the separation, in samples, between the two inputs to the phase error (PE) processor.
div_ratio: The division ratio required to get the clock component frequency from the carrier frequency.
s_freq: The sampling frequency of the simulation.
win_length: The length of the integration window in the PE processor.

Functional Description:

This block attempts to recover the clock component from a PN sequence. It outputs this recovered clock, which is phase-locked, in-phase, to the PN clock component. It has no real capability to adjust the output frequency, since it should be a fixed ratio of the carrier frequency.

This block contains multirate blocks.

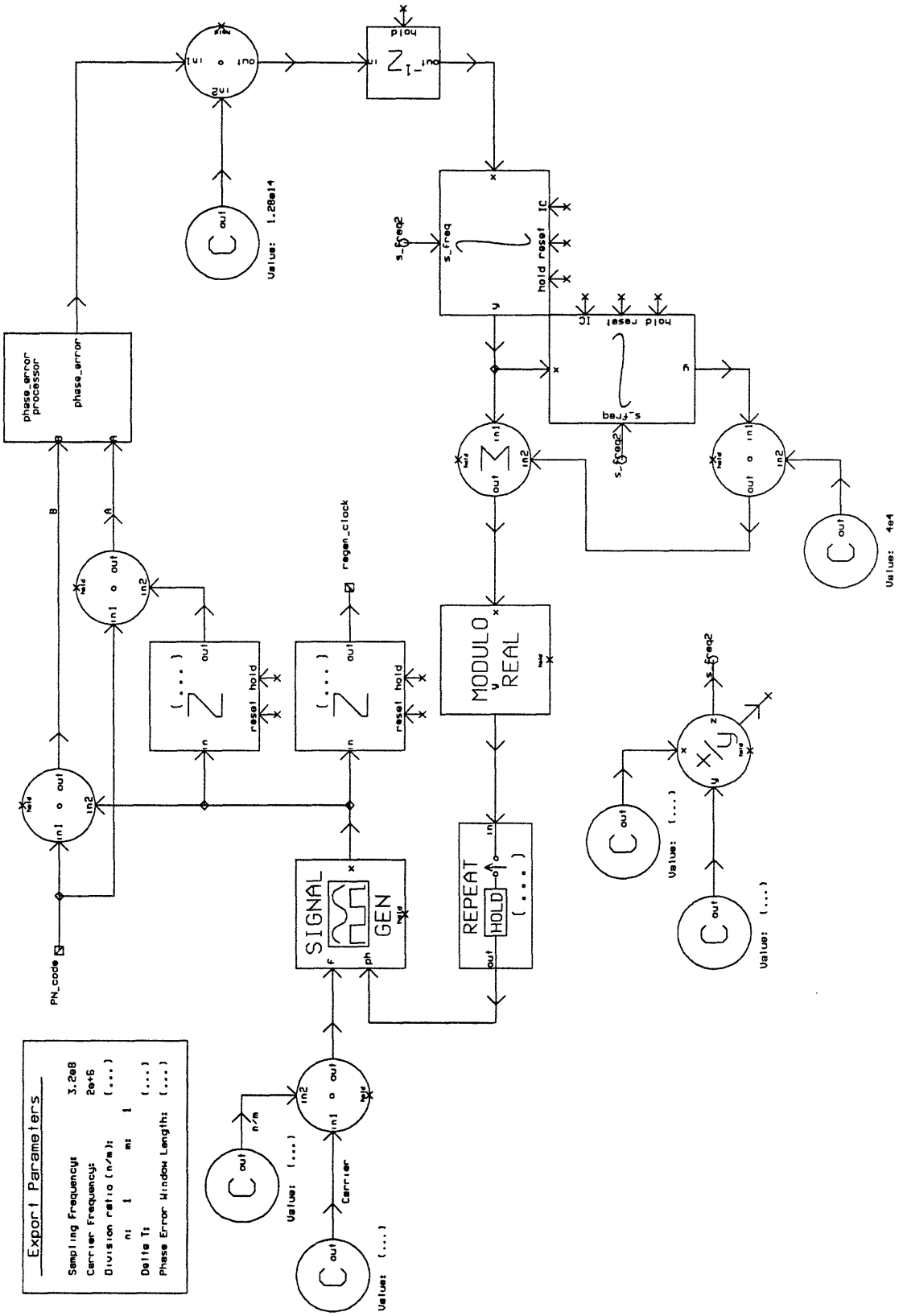

Figure B.2: clock_acq block diagram.

Block Name: code_combiner

Synopsis:

Input Signals:

clock: The clock component of the PN code.

PN_codes: A vector containing all of the PN components, except the clock component.

Output Signals:

PN_sequence: The combined PN code sequence.

Parameters:

none

Functional Description:

This block combines the PN code components and outputs the combined sequence.

The formula it follows is as follows:

$$PN = C_l \cdot (OR(C_i)) + (AND(C_i)) \quad \text{(B.1)}$$

Where C_l represents the clock component, C_i represents all the other components, $AND(...)$ represents the logical AND of all the signals inside the parenthesis, and $OR(...)$ represents the logical OR of all the signals inside the parenthesis.

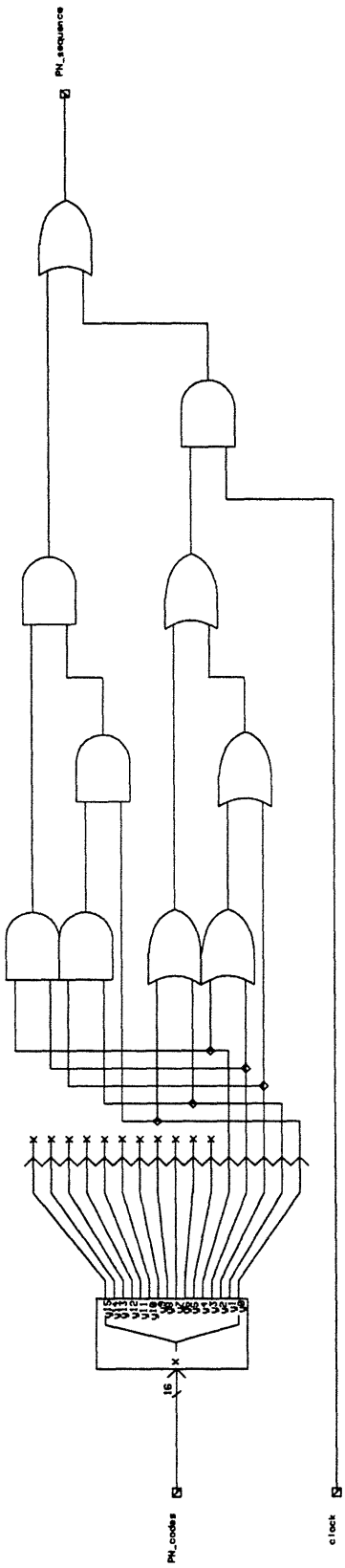

Figure B.3: code_combiner block diagram.

Block Name: code_convert

Synopsis:

Input Signals:

In: The unconverted PN sequence. High is 1, Low is 0.

Output Signals:

Out: The converted PN sequence. High is 1, Low is -1.

Parameters:

none

Functional Description:

This block rescales the PN sequence and converts it from a standard binary representation to a representation that is AC coupled. It changes the representation from high 1, low 0, to a representation of high 1, low -1.

The formula it follows is as follows:

$$Out = (In \times 2) - 1 \qquad \text{(B.2)}$$

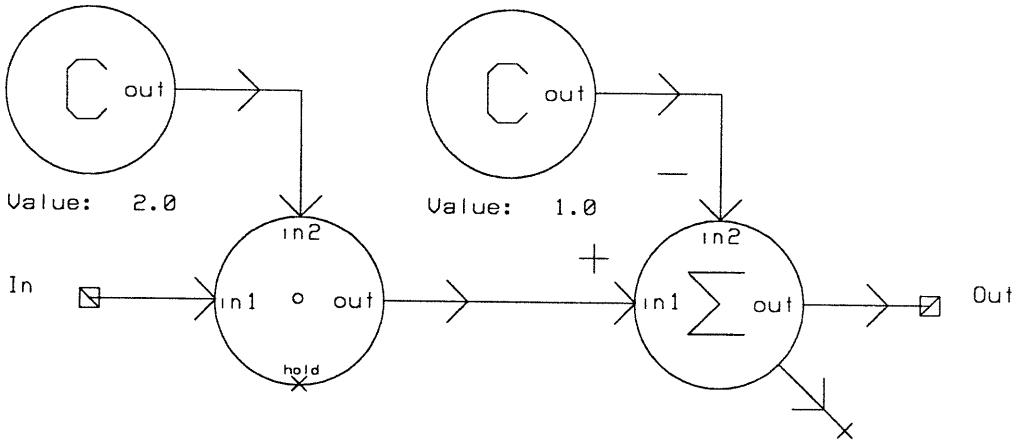

Figure B.4: code_convert block diagram.

Block Name: **comp_rec_all**

Synopsis:

Input Signals:

clk: The digital clock.
PN_sequence: The received PN sequence.
recovered_clock: The recovered clock component of the received PN sequence.
reset: A control signal allowing the block to be reset to its initial state.

Output Signals:

comp*X*_phase: The phase of component X, so an external generator can be aligned to the internal generator. X is between 1 and 5.
ld_c*X*: A control signal that tells the external generator when the **comp*X*_phase** signal is valid. X is between 1 and 5.

Parameters:

accum_per: The number of component periods over which to perform the correlation.
c*X*_int: An integer representation of the binary code for component X.
cX_len: The length (in bits) of component X.

Functional Description:

This block performs maximum-likelihood correlation detections on each of the 5 PN components. It then outputs the phase of each component, and begins again.

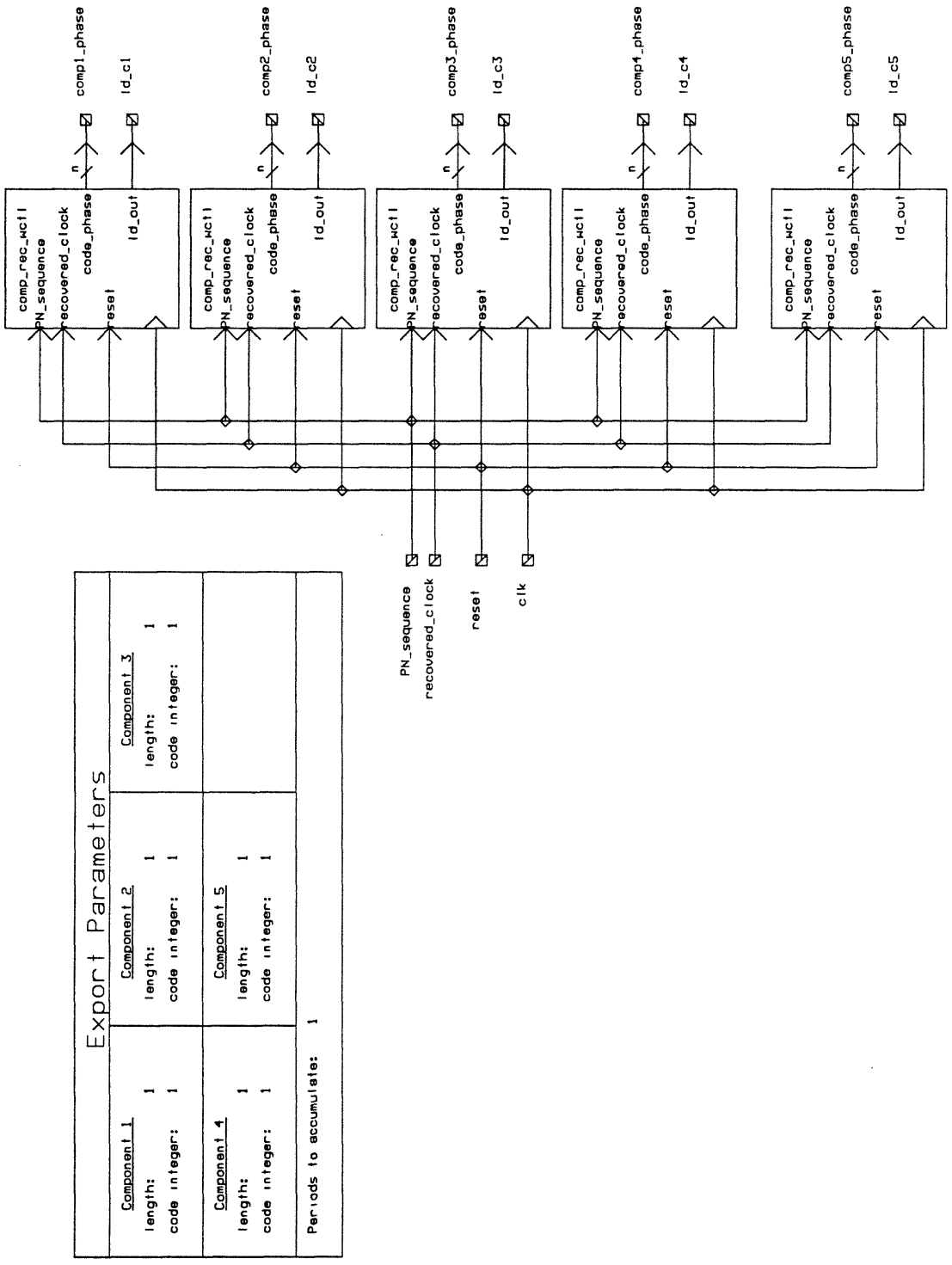

Figure B.5: comp_rec_all block diagram.

Block Name: comp_rec_wctl

Synopsis:

Input Signals:

clk: The digital clock.
PN_sequence: The received PN sequence.
recovered_clock: The recovered clock component of the received PN sequence.
reset: A control signal allowing the block to be reset to its initial state.

Output Signals:

code_phase: The phase of the component, so an external generator can be aligned to the internal generator.
ld_out: A control signal that tells the external generator when the **code_phase** signal is valid.

Parameters:

accum_per: The number of component periods over which to perform the correlation.
code_int: An integer representation of the binary code for the PN component.
num_bits: The length of the PN component (in bits).

Functional Description:

This block performs a maximum-likelihood correlation detection to determine the phase of a single PN component. It then outputs this phase, and begins again.

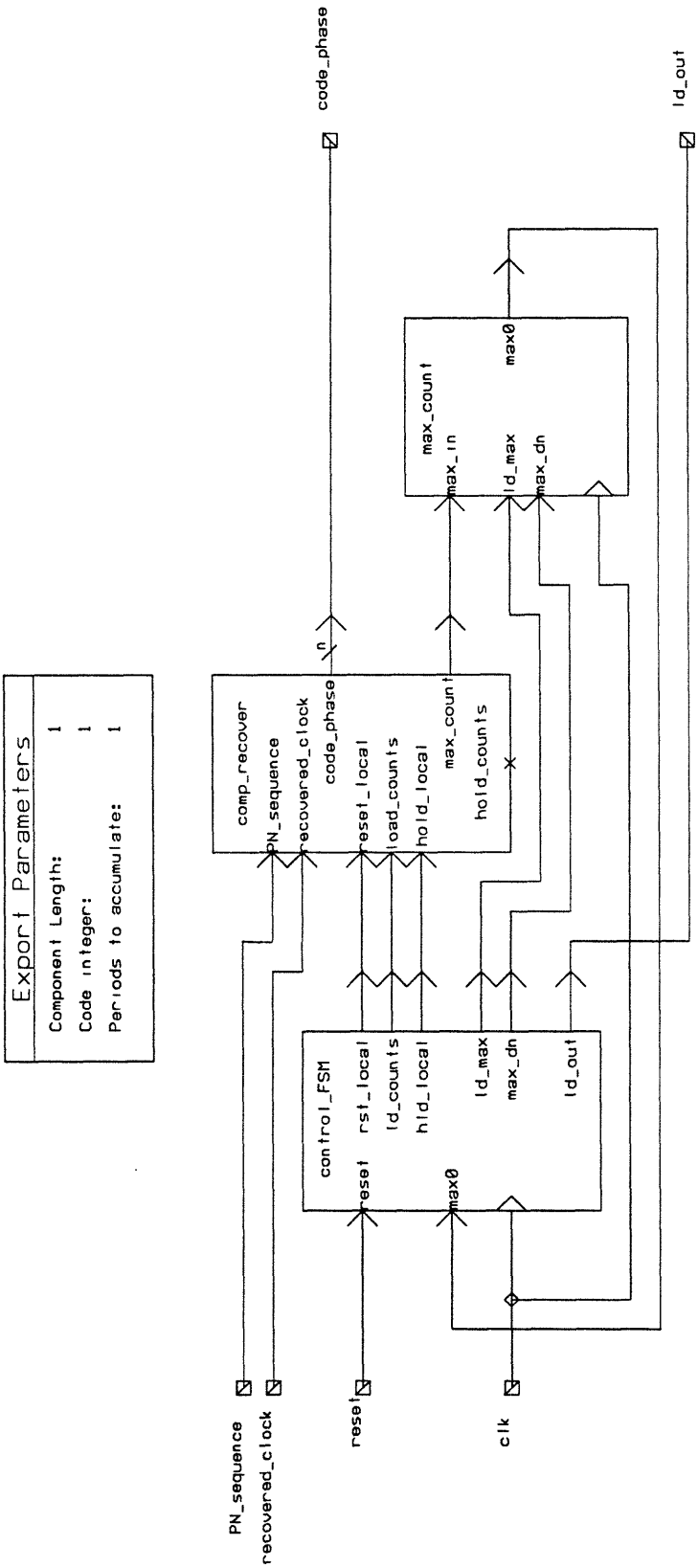

Figure B.6: comp_rec_wctl block diagram.

Block Name: comp_recover

Synopsis:

Input Signals:

hold_counts: A control signal that allows the correlation counters to be held.
hold_local: A control signal that allows the local PN generator to be held.
load_counts: A control signal that resets the correlation counters.
PN_sequence: The received PN sequence.
recovered_clock: The recovered clock component of the received PN sequence.
reset_local: A control signal that resets the local PN generator.

Output Signals:

code_phase: The phase of the local PN generator.
max_count: The location of the highest peak in the correlation.

Parameters:

code_int: An integer representation of the binary code for the PN component.
num_bits: The length of the PN component (in bits).

Functional Description:

This block is the correlation detector used by **comp_rec_wctl**. It takes a locally generated version of the PN sequence and correlates it against the incoming received PN sequence. It then outputs the index of the highest peak in the correlation.

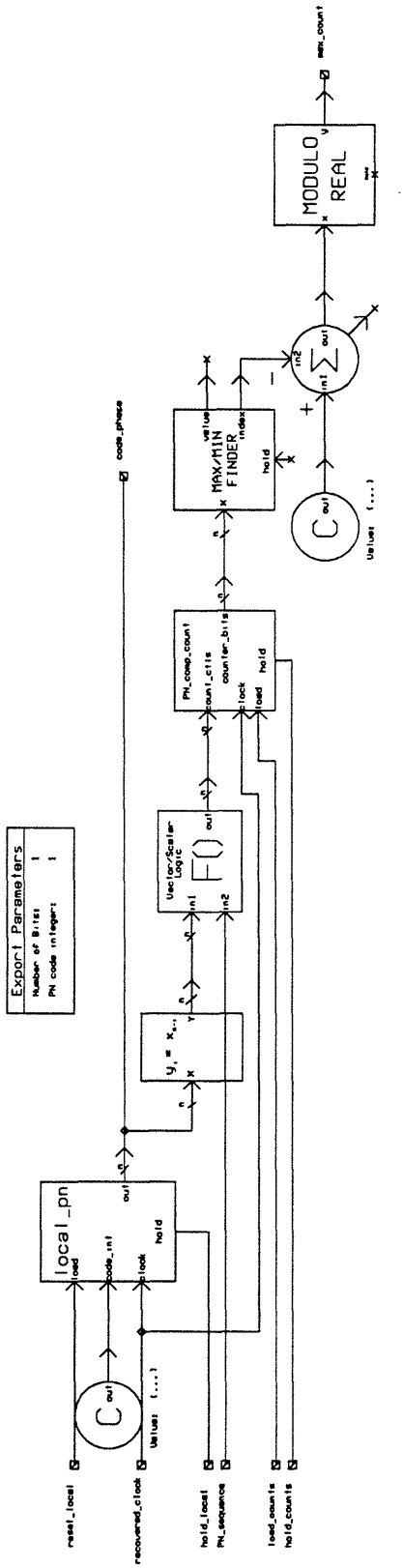

Figure B.7: comp_recover block diagram.

Block Name: control_cnt

Synopsis:

Input Signals:

clk: The digital clock.
count: A control signal that causes the counter to increment.
reset: A control signal that resets the counter to zero.

Output Signals:

done: A control signal that indicates that the counter has reached its highest value.

Parameters:

size_cnt: The size of the counter.

Functional Description:

This block is a counter with synchronous count enable and reset controls, and a carry flag. It can be of arbitrary size.

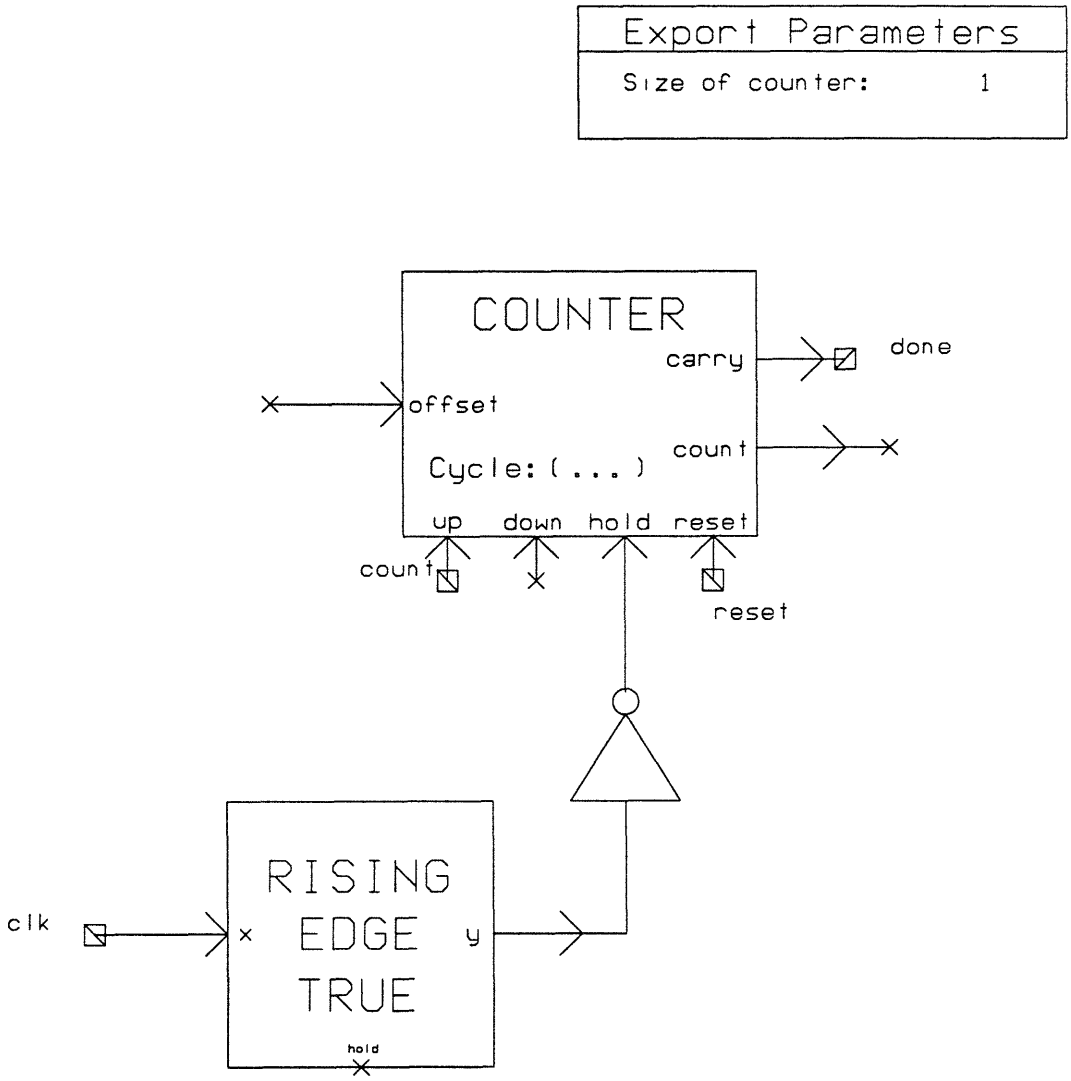

Figure B.8: control_cnt block diagram.

Block Name: **control_fsm**

Synopsis:

Input Signals:

clk: The digital clock.
max0: A control signal that indicates when the value stored in the **max_count** block reaches zero.
reset: A control signal that allows the block to be reset to its initial state.

Output Signals:

hld_local: A control signal that indicates that the local generator should be held.
ld_counts: A control signal that indicates that the correlation counters should be reset.
ld_max: A control signal that indicates that the **max_count** block should load.
ld_out: A control signal indicating that the external PN generator should load.
max_dn: A control signal to show that the **max_count** block should decrement.
rst_local: A control signal indicating that the local PN generator should be reset.

Parameters:

cnt1: The size of the PN component.
cnt2: The number of periods over which to correlate.

Functional Description:

This block provides the control logic for the maximum-likelihood correlation detector. It consists of a finite-state machine, and two counters.

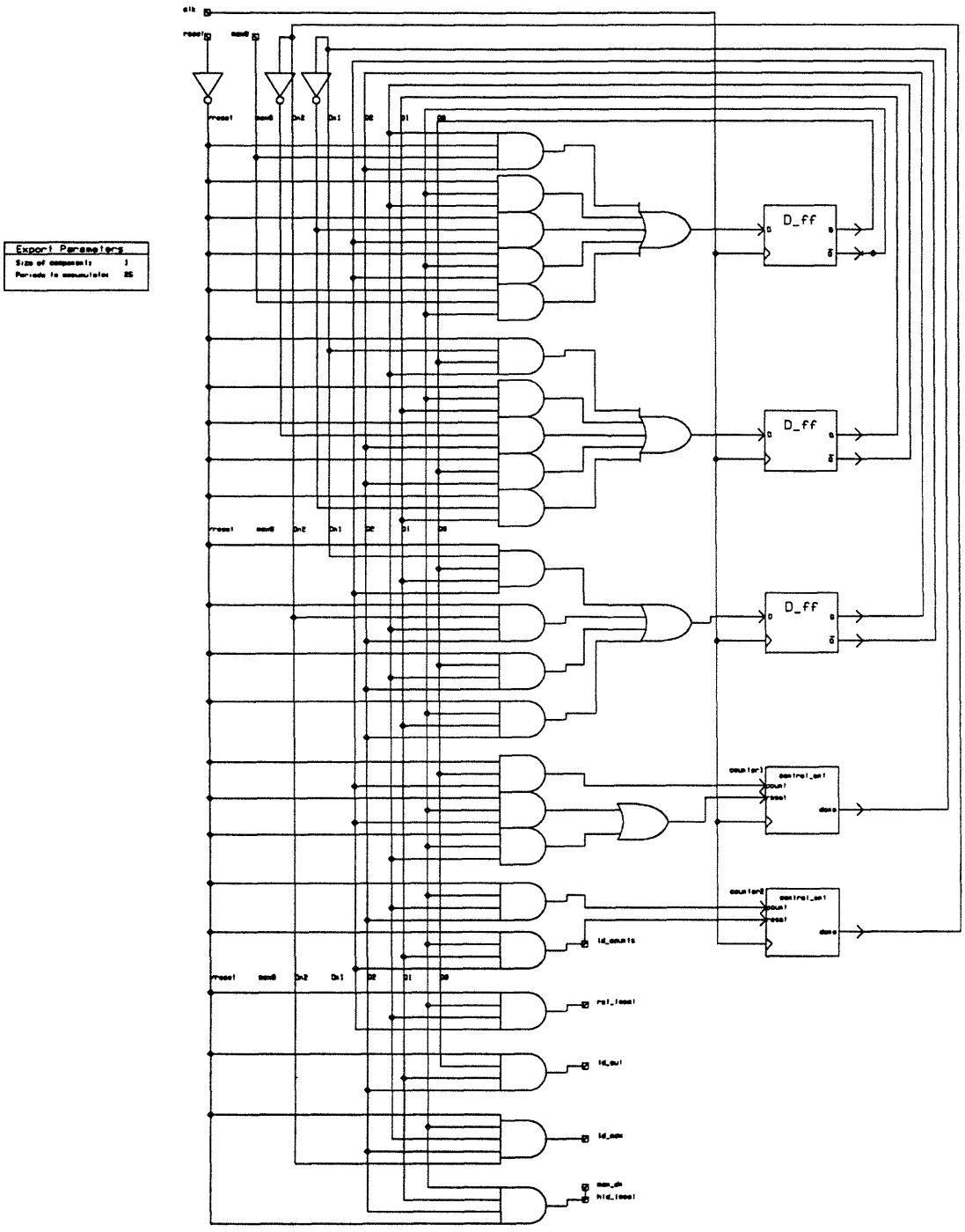

Figure B.9: control_fsm block diagram.

Block Name: **d_ff**

Synopsis:

Input Signals:

clk: The digital clock.
D: The input value.

Output Signals:

Q: The registered output value.
Q_bar: The inverted, registered output value.

Parameters:

none

Functional Description:

This is a standard, ordinary, D-type rising-edge-triggered flip-flop. The **D** input is copied to the **Q** output on each rising edge of the **clk** input. Between edges, the output is held.

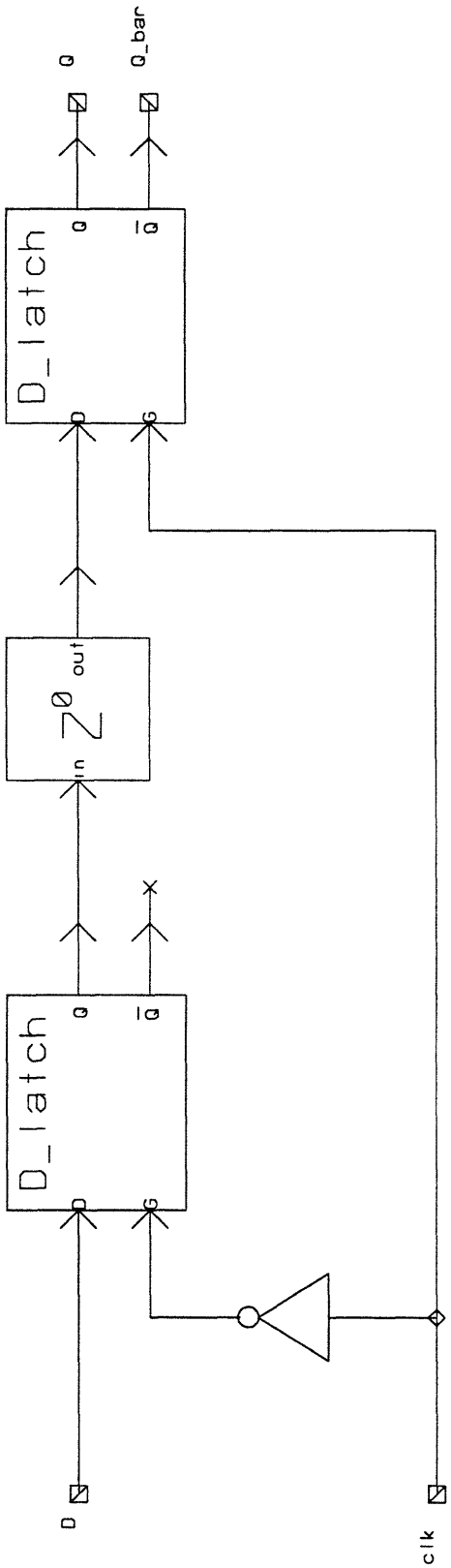

Figure B.10: d_ff block diagram.

Block Name: **d_latch**

Synopsis:

Input Signals:

D: The input value.
G: The gate control signal.

Output Signals:

Q: The latched output value.
Q_bar: The inverted, latched output value.

Parameters:

none

Functional Description:

This is a standard, ordinary, D-type latch. When the **G** input is high, the **D** input is copied to the **Q** output. When the **G** input is low, the output is held constant.

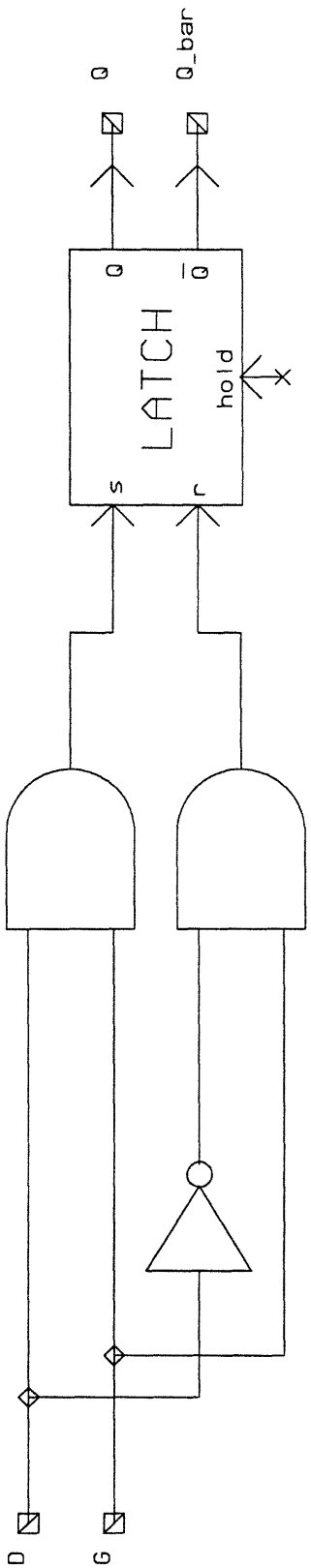

Figure B.11: d_latch block diagram.

Block Name: edge_true

Synopsis:

Input Signals:

hold: A control signal that allows the output to be held, regardless of changes in the input.

x: The signal in which to detect edges.

Output Signals:

y: A signal indicating where the edges of **x** are located.

Parameters:

none

Functional Description:

This block outputs a single high sample whenever the input signal transitions from low to high, or from high to low.

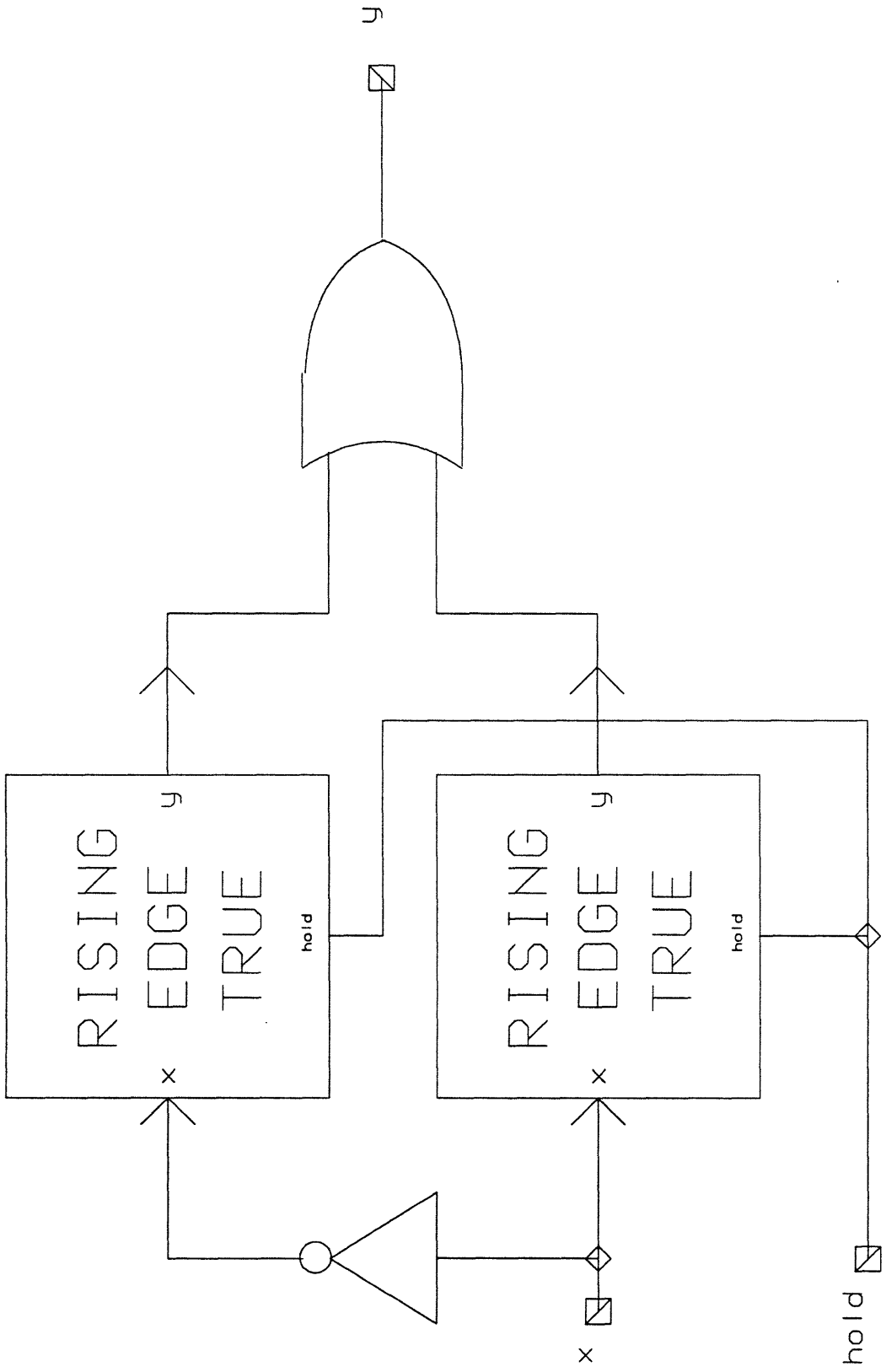

Figure B.12: edge_true block diagram.

Block Name: **int2vec**

Synopsis:

Input Signals:

in: The signal that is to be converted.

Output Signals:

out: A vector of the binary representation of the input signal.

Parameters:

hold_in_val: The initial value of each component of the vector.
high_low: The order in which the binary representation fills the vector (MSB or LSB to the low component).
out_IOVEC_LEN: The number of bits used in the binary representation.

Functional Description:

This block takes an integer as input, and outputs a vector containing a binary representation of that integer on the same simulation iteration. It fills the vector such that the low component of the vector contains either the LSB or the MSB of the binary representation, according to the **high_low** parameter.

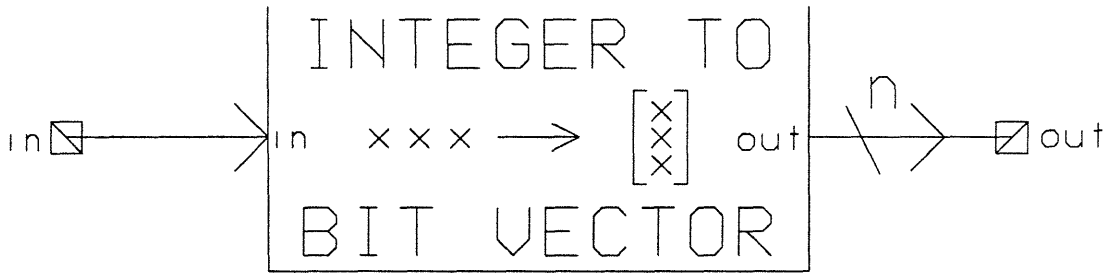

Figure B.13: int2vec block symbol.

INTEGER TO VECTOR BLOCK PARAMETERS	
MAIN PARAMETERS:	
Number of bits per integer (size of vector)	16
Low component of vector is (MSB/LSB):	'MSB'
MISCELLANEOUS PARAMETERS:	
Initial value	0.0
ALL_FEED_THROUGH out	

Figure B.14: int2vec block parameters.

int2vec block code

int2vec.c

```
#include "spw_platform.h"
#ifdef UNIX
#include "caedata/regen_ranging/int2vec/blockcode/int2vecu.c"
#else
#ifdef VAX_VMS
#include "[caedata.regen_ranging.int2vec.blockcode]int2vecu.c"
#endif VAX_VMS
#endif UNIX
static char *REVISION = "2.50";

/*
 *
 *       Block Function: int2vec
 *       Library: regen_ranging
 *       Date: Fri Aug 19 13:44:02 1994
 *
 */
```

```
/*****************************************************************/
/*                                                               */
/*    FEED_THROUGH_LIST INFORMATION:                             */
/*                                                               */
/*    --> FEED_THROUGH_TYPE IS NOT EDITABLE. The BDE parameter   */
/*        screen associated with the block must be edited to change */
/*        the block's FEED_THROUGH_TYPE.                         */
/*                                                               */
/*        FEED_THROUGH_TYPE = ALL_FEED_THROUGH.                  */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*    LINK_OPTIONS INFORMATION:                                  */
/*                                                               */
/*    --> The LINK_OPTIONS list is editable.  It contains all the */
/*        libraries which the code must be linked to.  Each item in */
/*        the list must be surrounded by double quotes and       */
/*        separated by commas.  The math library is automatically */
/*        linked, and does not need to be specified.  The paths  */
/*        may be specified as full paths or as paths relative to */
/*        the host.                                              */
/*        A link option can also be specified in the form "-lx"  */
/*        (where x is defined in the UNIX manual on "ld"         */
/*                                                               */
/*        IMPORTANT:  The entire LINK_OPTIONS list must be deleted */
/*        if it doesn't contain any elements.                    */
/*                                                               */
/*        Sample LINK_OPTIONS list:                              */
/*        (Actual list should be placed below this comment block) */
/*                                                               */
/*        LINK_OPTIONS = { "-lm",                                */
/*                         "//host/code/lib/sample.a" };         */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*    INCLUDE_DIRS INFORMATION:                                  */
/*                                                               */
/*    --> The INCLUDE_DIRS list is editable.  The list should    */
/*        contain all directory search paths needed to locate all */
/*        the include files used by this block.  It has the same */
```

```
/*        format as the LINK_OPTIONS list.                              */
/*                                                                      */
/*        IMPORTANT:  The entire INCLUDE_DIRS list must be deleted      */
/*        if it doesn't contain any elements.                           */
/*                                                                      */
/*        Sample INCLUDE_DIRS list:                                     */
/*        (Actual list should be placed below this comment block)       */
/*                                                                      */
/*        INCLUDE_DIRS = { "//host/u/code/include",                     */
/*                         "//host/lib/dir" };                          */
/*                                                                      */
/*                                                                      */
/************************************************************************/

/************************************************************************/
/*                                                                      */
/*       EDITABLE FUNCTIONS                                             */
/*                  --> In_int2vec_regen_ranging ()
/*                  --> Ro_int2vec_regen_ranging ()
/*                  --> Te_int2vec_regen_ranging ()
/*                                                                      */
/*       Structure use:                                                 */
/*         Typical input value reference                                */
/*           local_var = *(spb_input->var_name);                        */
/* **OR**    local_var = I_var_name;    */
/*         Typical output value update                                  */
/*           spb_output->var_name = local_var;                          */
/* **OR**    O_var_name = local_var;    */
/*         Typical parameter reference                                  */
/*           local_var = spb_parm->var_name;                            */
/* **OR**    local_var = P_var_name;    */
/*                                                                      */
/*       (See reference manual for further information)                 */
/*                                                                      */
/************************************************************************/

/*
 *             Initialize Function (must be present)
 *             --> If editing, modify only the lines within the
 *                 function's opening and closing brackets.
 *
 *             This function is used to initialize the state structure
 *                  and constant outputs of the block.  It is called once
 *                   for each block instance during simulation.
 *
 *             Function must always return either SYS_OK, SYS_TERM,
```

```
 *                  or SYS_FATAL by using the return() function.
 *                  User may modify the line containing
"return(SYS_OK);".
 */

In_int2vec_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_int2vec_regen_ranging *spb_parm;
STRUCT It_int2vec_regen_ranging *spb_input;
STRUCT Ot_int2vec_regen_ranging *spb_output;
STRUCT St_int2vec_regen_ranging *spb_state;
{
  int i;

  for (i = 0 ; i < O_out_iovec_len ; i++)
    VEC_SET(O_out, i, P_hold_in_val);

  return (SYS_OK);
}

/*
 *             Run Output Function (must be present)
 *             --> If editing, modify only the lines within the
 *                 function's opening and closing brackets.
 *
 *             This function is used to update the outputs and/or state
 *                  of the block.  It is called each iteration, for each
 *                  block instance during simulation.
 *
 *             Function must always return either SYS_OK, SYS_TERM,
 *                  or SYS_FATAL by using the return() function.
 *                  User may modify the line containing
"return(SYS_OK);".
 */

Ro_int2vec_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_int2vec_regen_ranging *spb_parm;
STRUCT It_int2vec_regen_ranging *spb_input;
STRUCT Ot_int2vec_regen_ranging *spb_output;
STRUCT St_int2vec_regen_ranging *spb_state;
{
  int i;
  int temp_in = (int) I_in;

  if (*P_high_low == 'L')
    for (i = 0 ; i < O_out_iovec_len ; i++)
      {
```

```
        VEC_SET(O_out,i,(temp_in & 1));
        temp_in = temp_in >> 1;
      }
  else
    for (i = (O_out_iovec_len - 1) ; i >= 0 ; i--)
      {
        VEC_SET(O_out,i,(temp_in & 1));
        temp_in = temp_in >> 1;
      }

  return (SYS_OK);
}

/*
 *              Termination Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to dump the final state of the
 *                   block.  It is called once for each block instance
 *                   during the simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Te_int2vec_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_int2vec_regen_ranging *spb_parm;
STRUCT It_int2vec_regen_ranging *spb_input;
STRUCT Ot_int2vec_regen_ranging *spb_output;
STRUCT St_int2vec_regen_ranging *spb_state;
{

        return (SYS_OK);
}

/*****************************************************************/
/*                                                               */
/*      Add any additional functions you need here.              */
/*                                                               */
/*****************************************************************/
```

int2vec.h

```
#include "FBCDEFS.h"

/*
 *
 *       Block Function: int2vec
 *       Library: regen_ranging
 *       Date: Fri Aug 19 13:44:02 1994
 *
 */

/************************************************************************/
/*                                                                      */
/*       EDITABLE USER DEFINED STATE STRUCTURE                          */
/*                   --> STRUCT St_int2vec_regen_ranging
/*                                                                      */
/************************************************************************/

/*
 *                State Structure (User Defined, editable)
 */
STRUCT St_int2vec_regen_ranging {
        int instance;
};

/************************************************************************/
/*                                                                      */
/*       UNEDITABLE SIMULATOR DEFINED STRUCTURES                        */
/*                   --> STRUCT Pt_int2vec_regen_ranging
/*                   --> STRUCT It_int2vec_regen_ranging
/*                   --> STRUCT Ot_int2vec_regen_ranging
/*                                                                      */
/************************************************************************/
/*
 *                Parameter Structure, Simulator Defined, uneditable
 */
STRUCT Pt_int2vec_regen_ranging {
        char * high_low;
        double hold_in_val;
};

/*
 *                Input Structure, Simulator Defined, uneditable
 */
STRUCT It_int2vec_regen_ranging {
```

```
        double *in;
};

/*
 *                Output Structure, Simulator Defined, uneditable
 */
STRUCT Ot_int2vec_regen_ranging {
        long out_iovec_len;
        double *out;
};

/*****************************************************************/
/*                                                               */
/*       The following #defines may be used to shorten           */
/*       references to members of the above structures.          */
/*                                                               */
/*****************************************************************/
#define P_high_low (spb_parm->high_low)
#define P_hold_in_val (spb_parm->hold_in_val)
#define I_in (*spb_input->in)
#define O_out_iovec_len (spb_output->out_iovec_len)
#define O_out (spb_output->out)
```

Block Name: **int_dump**

Synopsis:

Input Signals:

in: Input to be integrated.
tim_ov: Input that overrides internal integrator reset circuitry.

Output Signals:

out: Output integration at the end of the dump period.

Parameters:

period: Length of integration between dumps.
s_freq: The sampling frequency.

Functional Description:

This block integrates the input signal over the specified dump period then it outputs the integrated value. The integrator is reset with the current input value when the dump occurs. The integration is performed using a backwards difference algorithm {y(n)=y(n-1)+[x(n)/**s_freq**]}. The output of this block runs at 1/**period** of the normal sampling rate.

This is a multirate block.

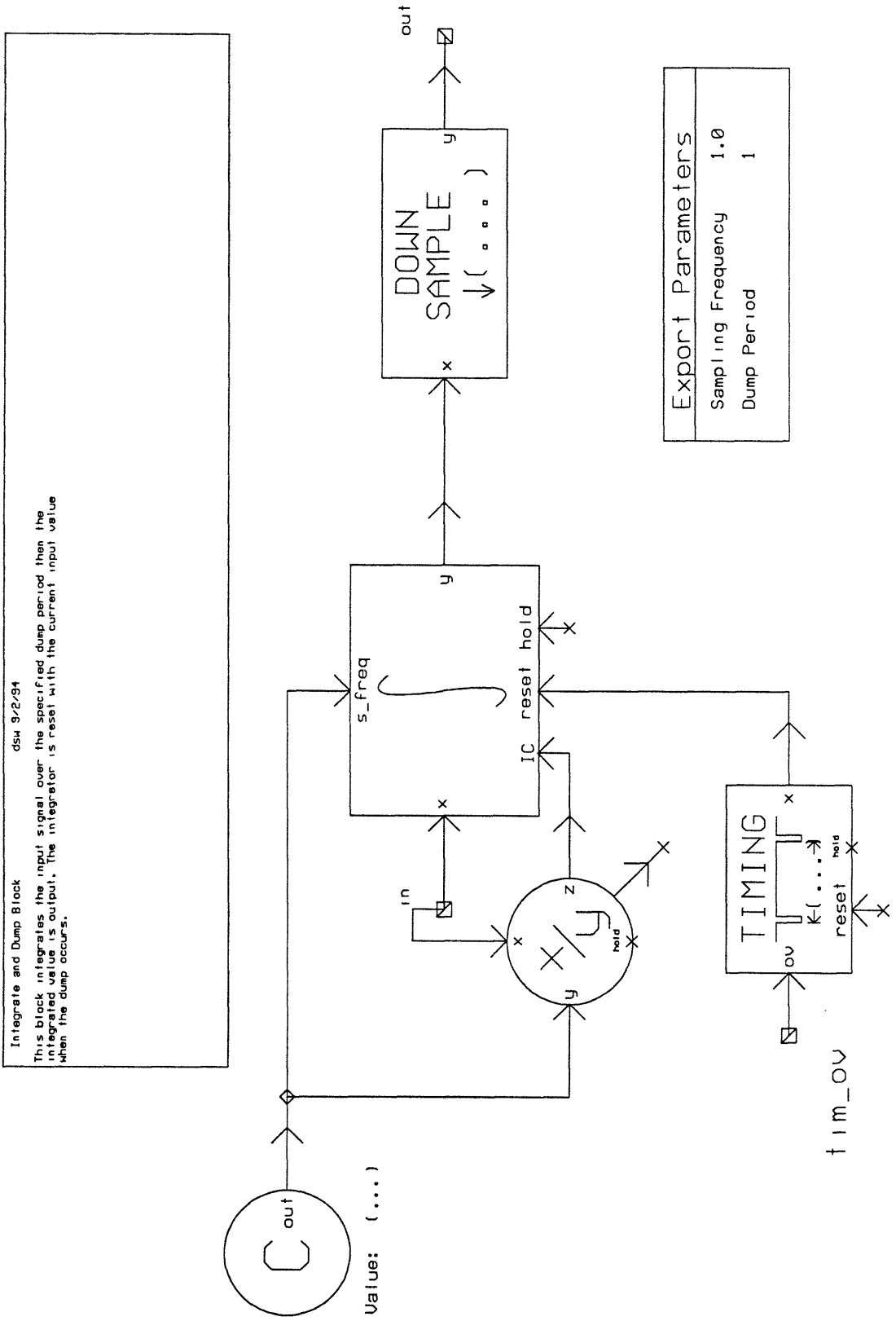

Figure B.15: int_dump block diagram.

Block Name: **local_pn**

Synopsis:

Input Signals:

clock: The PN clock component.
code_int: An integer representation of the binary PN component code.
hold: A control signal that allows the PN generator to be held.
load: A control signal that loads the PN component into the generator.

Output Signals:

out: The PN component output.

Parameters:

num_bits: The length of the PN component (in bits).

Functional Description:

This block generates a PN code component, and outputs the entire component as a vector, one bit per vector component.

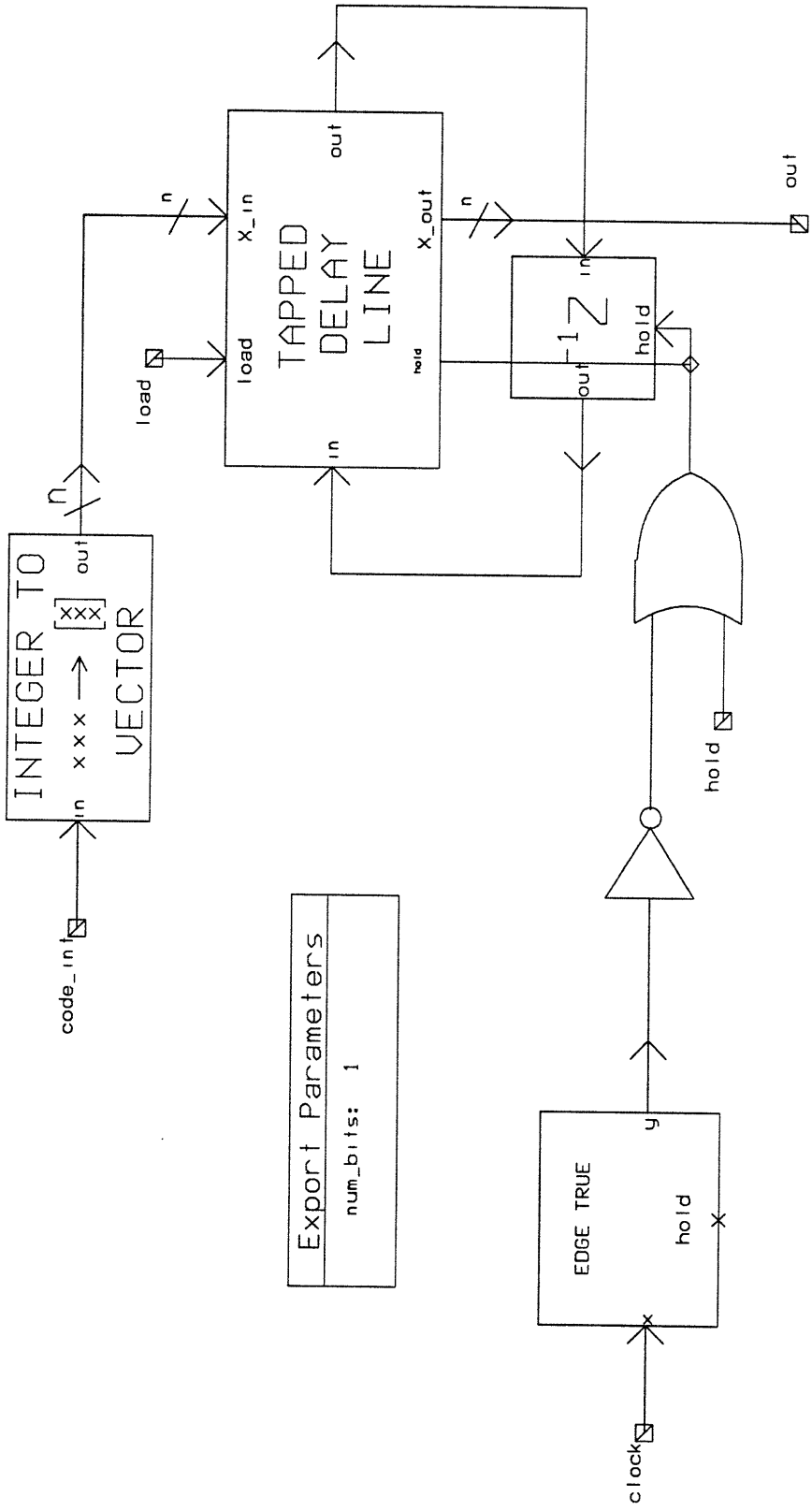

Figure B.16: local_pn block diagram.

Block Name: **local_pn2**

Synopsis:

Input Signals:

clock: The PN clock component.
code_phase: A vector representation of the PN component code. The phase of the component is also conveyed by the ordering of the vector.
hold: A control signal that allows the PN generator to be held.
load: A control signal that loads the PN component into the generator.

Output Signals:

out: The PN component output.

Parameters:

num_bits: The length of the PN component (in bits).

Functional Description:

This block generates a PN component, and outputs it serially. It accepts a vector input, representing both the value and phase of the PN component.

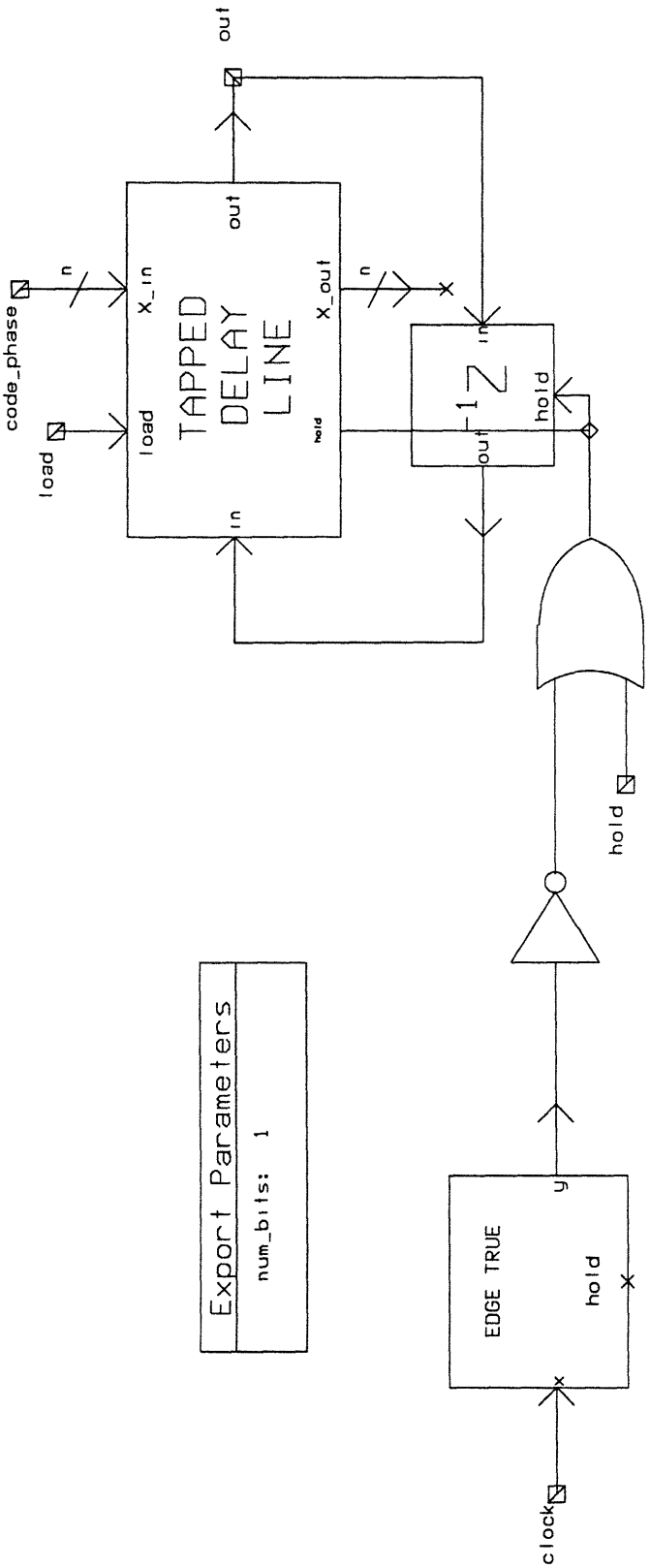

Figure B.17: local_pn2 block diagram.

Block Name: **max_count**

Synopsis:

Input Signals:

clk: The digital clock.
ld_max: A control signal indicating that the counter should be loaded with **max_in**.
max_in: The value to be loaded into the counter.
max_dn: A control signal allowing the counter to be decremented.

Output Signals:

max0: A signal that indicates when the counter contains the value zero.

Parameters:

none

Functional Description:

This block is a loadable down counter. A value can be loaded into the counter, and the value stored in the counter can be decremented. The counter indicates when its contents reach zero.

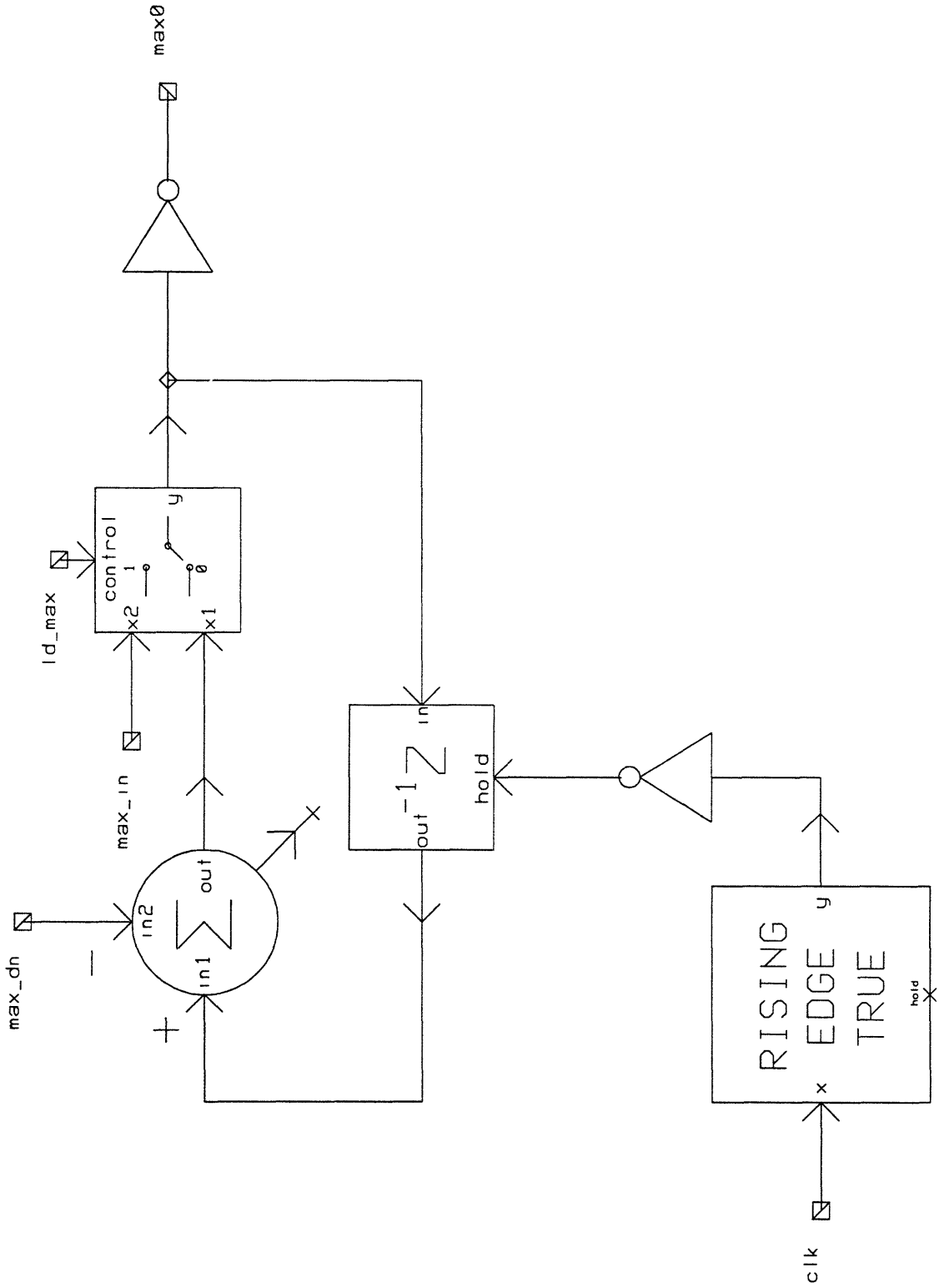

Figure B.18: max_count block diagram.

Block Name: **nodly**

Synopsis:

Input Signals:

in: The input signal.

Output Signals:

out: The output signal.

Parameters:

initial_value: The initial value of the output.

Functional Description:

This block does nothing. The only reason for its existence is to trick Comdisco into thinking that something is breaking a feedback path. The input is copied to the output each iteration, but the block has a feed type of NO_FEED_THROUGH, so Comdisco thinks it's a delay block.

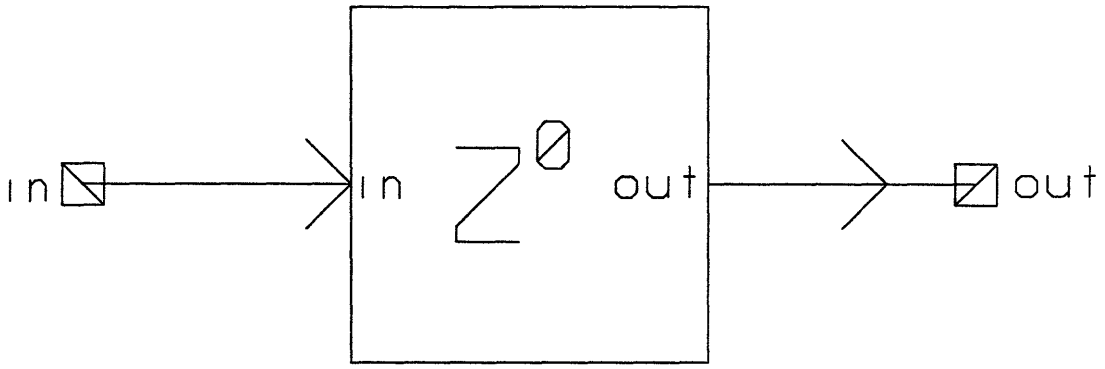

Figure B.19: nodly block symbol.

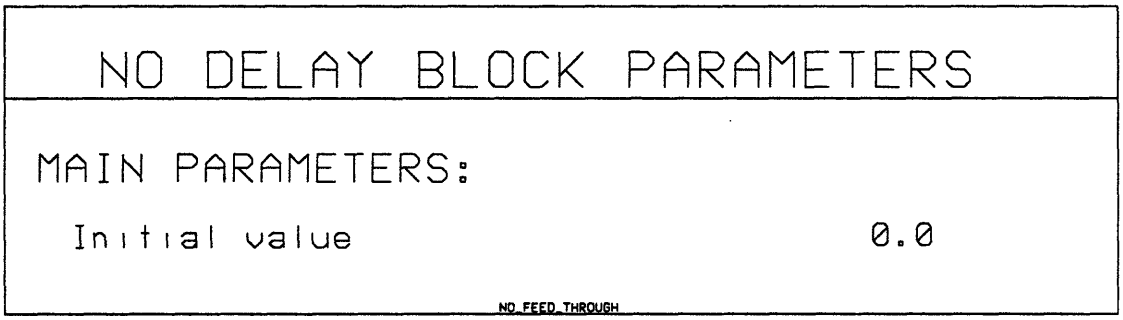

Figure B.20: nodly block parameters.

nodly block code

nodly.c

```
#include "spw_platform.h"
#ifdef UNIX
#include "caedata/regen_ranging/nodly/blockcode/nodlyu.c"
#else
#ifdef VAX_VMS
#include "[caedata.regen_ranging.nodly.blockcode]nodlyu.c"
#endif VAX_VMS
#endif UNIX
static char *REVISION = "2.50";

/*
 *
 *       Block Function: nodly
 *       Library: regen_ranging
 *       Date: Mon Sep 12 14:57:35 1994
 *
 */


/************************************************************************/
```

```
/*                                                                    */
/*    FEED_THROUGH_LIST INFORMATION:                                  */
/*                                                                    */
/*    --> FEED_THROUGH_TYPE IS NOT EDITABLE. The BDE parameter        */
/*        screen associated with the block must be edited to change   */
/*        the block's FEED_THROUGH_TYPE.                              */
/*                                                                    */
/*        FEED_THROUGH_TYPE = NO_FEED_THROUGH.                        */
/*                                                                    */
/**********************************************************************/

/**********************************************************************/
/*                                                                    */
/*    LINK_OPTIONS INFORMATION:                                       */
/*                                                                    */
/*    --> The LINK_OPTIONS list is editable.  It contains all the     */
/*        libraries which the code must be linked to.  Each item in   */
/*        the list must be surrounded by double quotes and            */
/*        separated by commas.  The math library is automatically     */
/*        linked, and does not need to be specified.  The paths       */
/*        may be specified as full paths or as paths relative to      */
/*        the host.                                                   */
/*        A link option can also be specified in the form "-lx"       */
/*        (where x is defined in the UNIX manual on "ld"              */
/*                                                                    */
/*        IMPORTANT:  The entire LINK_OPTIONS list must be deleted    */
/*        if it doesn't contain any elements.                         */
/*                                                                    */
/*        Sample LINK_OPTIONS list:                                   */
/*        (Actual list should be placed below this comment block)     */
/*                                                                    */
/*        LINK_OPTIONS = { "-lm",                                     */
/*                         "//host/code/lib/sample.a" };              */
/*                                                                    */
/*                                                                    */
/**********************************************************************/

/**********************************************************************/
/*                                                                    */
/*    INCLUDE_DIRS INFORMATION:                                       */
/*                                                                    */
/*    --> The INCLUDE_DIRS list is editable.  The list should         */
/*        contain all directory search paths needed to locate all     */
/*        the include files used by this block.  It has the same      */
/*        format as the LINK_OPTIONS list.                            */
/*                                                                    */
```

```
/*        IMPORTANT:  The entire INCLUDE_DIRS list must be deleted  */
/*        if it doesn't contain any elements.                       */
/*                                                                  */
/*        Sample INCLUDE_DIRS list:                                 */
/*        (Actual list should be placed below this comment block)   */
/*                                                                  */
/*        INCLUDE_DIRS = { "//host/u/code/include",                 */
/*                         "//host/lib/dir" };                      */
/*                                                                  */
/*                                                                  */
/********************************************************************/

/********************************************************************/
/*                                                                  */
/*       EDITABLE FUNCTIONS                                         */
/*                  --> In_nodly_regen_ranging ()
/*                  --> Ro_nodly_regen_ranging ()
/*                  --> Ri_nodly_regen_ranging ()
/*                  --> Te_nodly_regen_ranging ()
/*                                                                  */
/*       Structure use:                                             */
/*         Typical input value reference                            */
/*           local_var = *(spb_input->var_name);                    */
/* **OR**    local_var = I_var_name;    */
/*         Typical output value update                              */
/*           spb_output->var_name = local_var;                      */
/* **OR**    O_var_name = local_var;    */
/*         Typical parameter reference                              */
/*           local_var = spb_parm->var_name;                        */
/* **OR**    local_var = P_var_name;    */
/*                                                                  */
/*       (See reference manual for further information)             */
/*                                                                  */
/********************************************************************/

/*
 *             Initialize Function (must be present)
 *             --> If editing, modify only the lines within the
 *                 function's opening and closing brackets.
 *
 *             This function is used to initialize the state structure
 *                  and constant outputs of the block.  It is called once
 *                  for each block instance during simulation.
 *
 *             Function must always return either SYS_OK, SYS_TERM,
 *                  or SYS_FATAL by using the return() function.
```

```
 *                   User may modify the line containing
"return(SYS_OK);".
 */

In_nodly_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_nodly_regen_ranging *spb_parm;
STRUCT It_nodly_regen_ranging *spb_input;
STRUCT Ot_nodly_regen_ranging *spb_output;
STRUCT St_nodly_regen_ranging *spb_state;
{
  O_out=P_initial_value;

  return (SYS_OK);
}

/*
 *              Run Output Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to update the outputs and/or state
 *                   of the block.  It is called each iteration, for each
 *                   block instance during simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Ro_nodly_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_nodly_regen_ranging *spb_parm;
STRUCT It_nodly_regen_ranging *spb_input;
STRUCT Ot_nodly_regen_ranging *spb_output;
STRUCT St_nodly_regen_ranging *spb_state;
{
  O_out=I_in;

  return (SYS_OK);
}

/*
 *              Run Input Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
```

```
 *              This function is used to update the state of the block.
 *                   It is called each iteration, for each block instance
 *                   during the simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Ri_nodly_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_nodly_regen_ranging *spb_parm;
STRUCT It_nodly_regen_ranging *spb_input;
STRUCT Ot_nodly_regen_ranging *spb_output;
STRUCT St_nodly_regen_ranging *spb_state;
{

        return (SYS_OK);
}

/*
 *              Termination Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to dump the final state of the
 *                   block.  It is called once for each block instance
 *                   during the simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Te_nodly_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_nodly_regen_ranging *spb_parm;
STRUCT It_nodly_regen_ranging *spb_input;
STRUCT Ot_nodly_regen_ranging *spb_output;
STRUCT St_nodly_regen_ranging *spb_state;
{

        return (SYS_OK);
}
```

nodly.h

```
#include "FBCDEFS.h"

/*
 *
 *       Block Function: nodly
 *       Library: regen_ranging
 *       Date: Mon Sep 12 14:57:35 1994
 *
 */

/*****************************************************************/
/*                                                               */
/*       EDITABLE USER DEFINED STATE STRUCTURE                   */
/*                  --> STRUCT St_nodly_regen_ranging
/*                                                               */
/*****************************************************************/

/*
 *                State Structure (User Defined, editable)
 */
STRUCT St_nodly_regen_ranging {
        int instance;
};

/*****************************************************************/
/*                                                               */
/*       UNEDITABLE SIMULATOR DEFINED STRUCTURES                 */
/*                  --> STRUCT Pt_nodly_regen_ranging
/*                  --> STRUCT It_nodly_regen_ranging
/*                  --> STRUCT Ot_nodly_regen_ranging
/*                                                               */
/*****************************************************************/
/*
 *                Parameter Structure, Simulator Defined, uneditable
 */
STRUCT Pt_nodly_regen_ranging {
        double initial_value;
};

/*
 *                Input Structure, Simulator Defined, uneditable
 */
STRUCT It_nodly_regen_ranging {
        double *in;
```

```
};

/*
 *                Output Structure, Simulator Defined, uneditable
 */
STRUCT Ot_nodly_regen_ranging {
        double out;
};

/*****************************************************************/
/*                                                               */
/*       The following #defines may be used to shorten           */
/*       references to members of the above structures.          */
/*                                                               */
/*****************************************************************/
#define P_initial_value (spb_parm->initial_value)
#define I_in (*spb_input->in)
#define O_out (spb_output->out)
```

Block Name: phase_error

Synopsis:

Input Signals:

A: The delayed code input.
B: The advanced (at least not delayed) code input.

Output Signals:

phase_error: A measure of the phase error between the original signal and the current not delayed (or halfway delayed) approximation to that signal.

Parameters:

s_freq: The sampling frequency of the simulation.
win_length: The length of the time average window in the integrate and dump block.

Functional Description:

This block attempts to provide a measure of the phase error between the two signals used to generate its inputs. Its two inputs A and B are the output of mixers that mix the original PN sequence and the regenerated clock either delayed by a quarter cycle, or advanced by a quarter cycle. It outputs the integral of the following function:

$$\frac{B-A}{2} \tag{B.3}$$

The length of the integration is set by the parameter **win_length**. In addition, due to the process of performing the integration, the output of this block runs at 1/**win_length** of the normal sampling rate.

This is a multirate block.

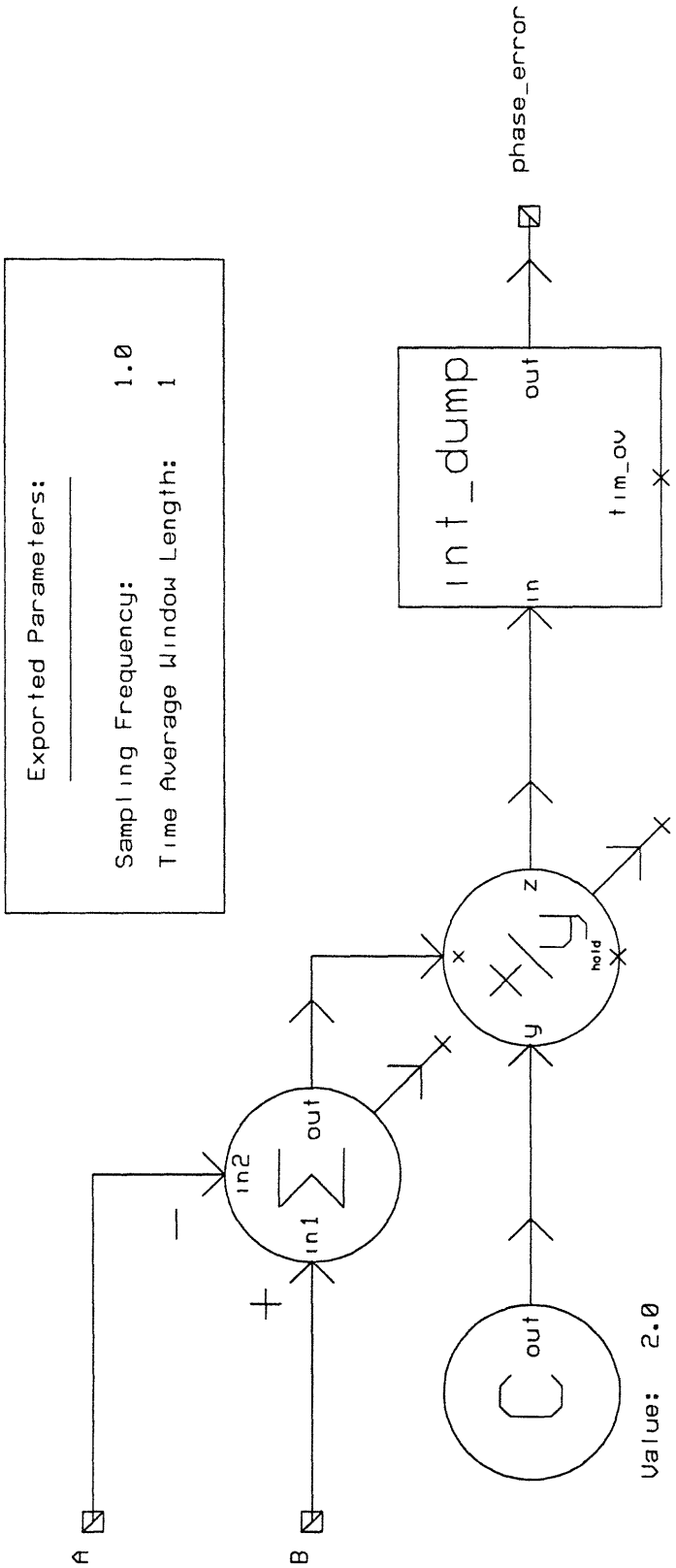

Figure B.21: phase_error block diagram.

Block Name: **pn_2comp**

Synopsis:

Input Signals:

clk_phase: The phase of the clock signal.
delays: A vector containing the delays for each component of the PN code.
hold: A control signal that allows the output to be held, regardless of the other inputs.
reset: A control signal that allows the source to be reset to its initial state.

Output Signals:

clock: The clock component of the PN code.
PN_sequence: The combined, converted PN code sequence. High is 1, Low is -1.

Parameters:

clk_freq: The frequency of the PN clock component.
noise_variance: The variance of the Gaussian noise added to the PN sequence.
s_freq: The sampling frequency for the simulation.

Functional Description:

This block generates, combines, and converts a noisy PN sequence with two components, plus clock. It allows the phase of the clock component and the delays for each individual PN component other than the clock to be input. It outputs a PN sequence that is AC coupled, with a high represented by a 1, and a low represented by a -1. White Gaussian noise with the specified variance is added to the output sequence.

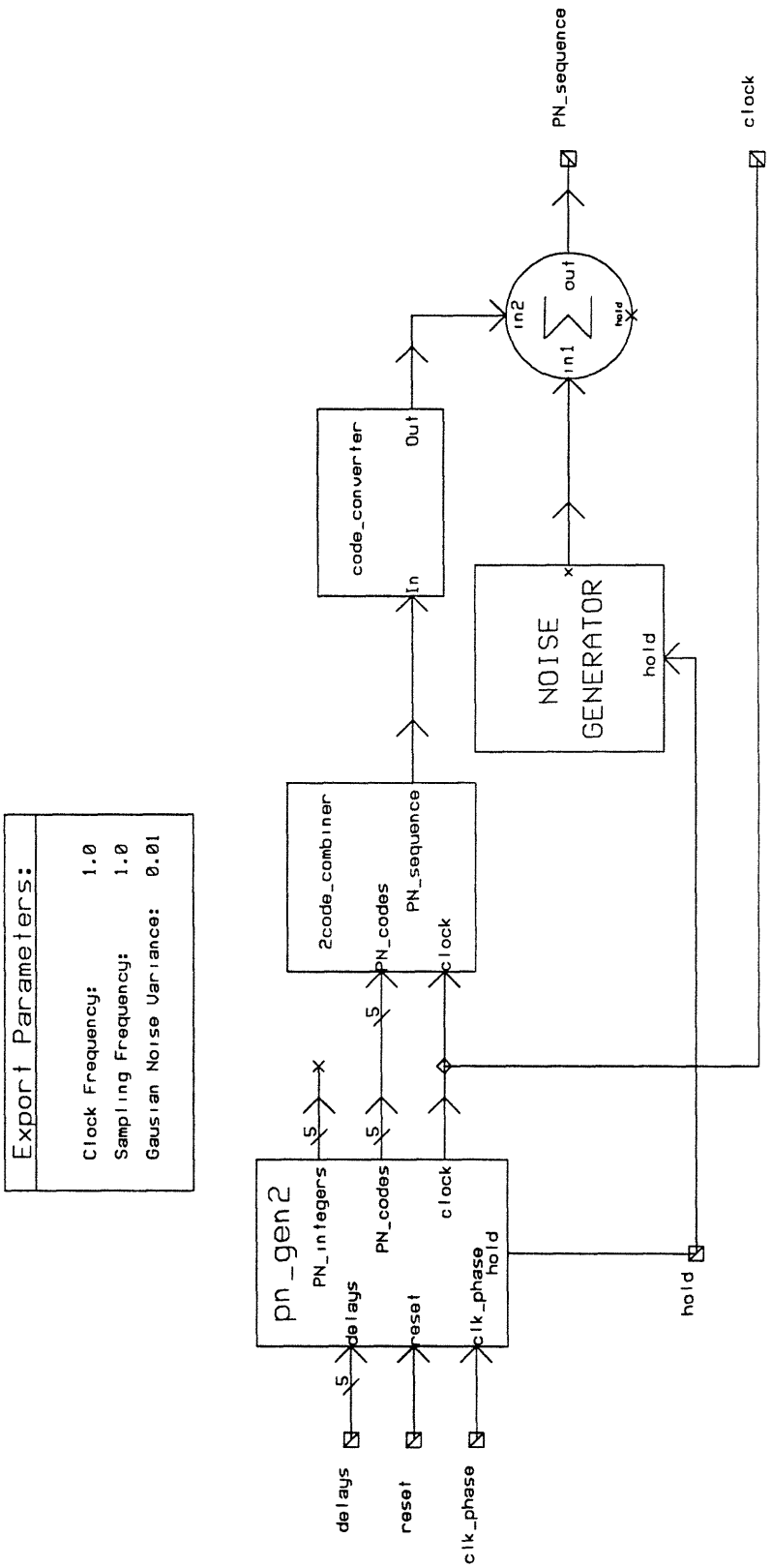

Figure B.22: pn_2comp block diagram.

Block Name: **pn_comp_count**

Synopsis:

Input Signals:

clock: The PN clock component.
count_ctls: The control bits for the counters.
hold: A control signal that allows the counters to be held.
load: A control signal that allows the counters to be reset.

Output Signals:

counter_bits: The contents of the counters.

Parameters:

num_bits: The length of the associated PN component (in bits). Also the number of counters.

Functional Description:

This block consists of **num_bits** counters, one for each bit of the associated PN component. The counters are set up so that the counters are incremented whenever their respective control line is high, and decremented whenever it is low. The counter values only change at edges of the **clock** input. In addition, the counters can be loaded with the current input values, essentially resetting them. This also can only take place at an edge of the **clock** input.

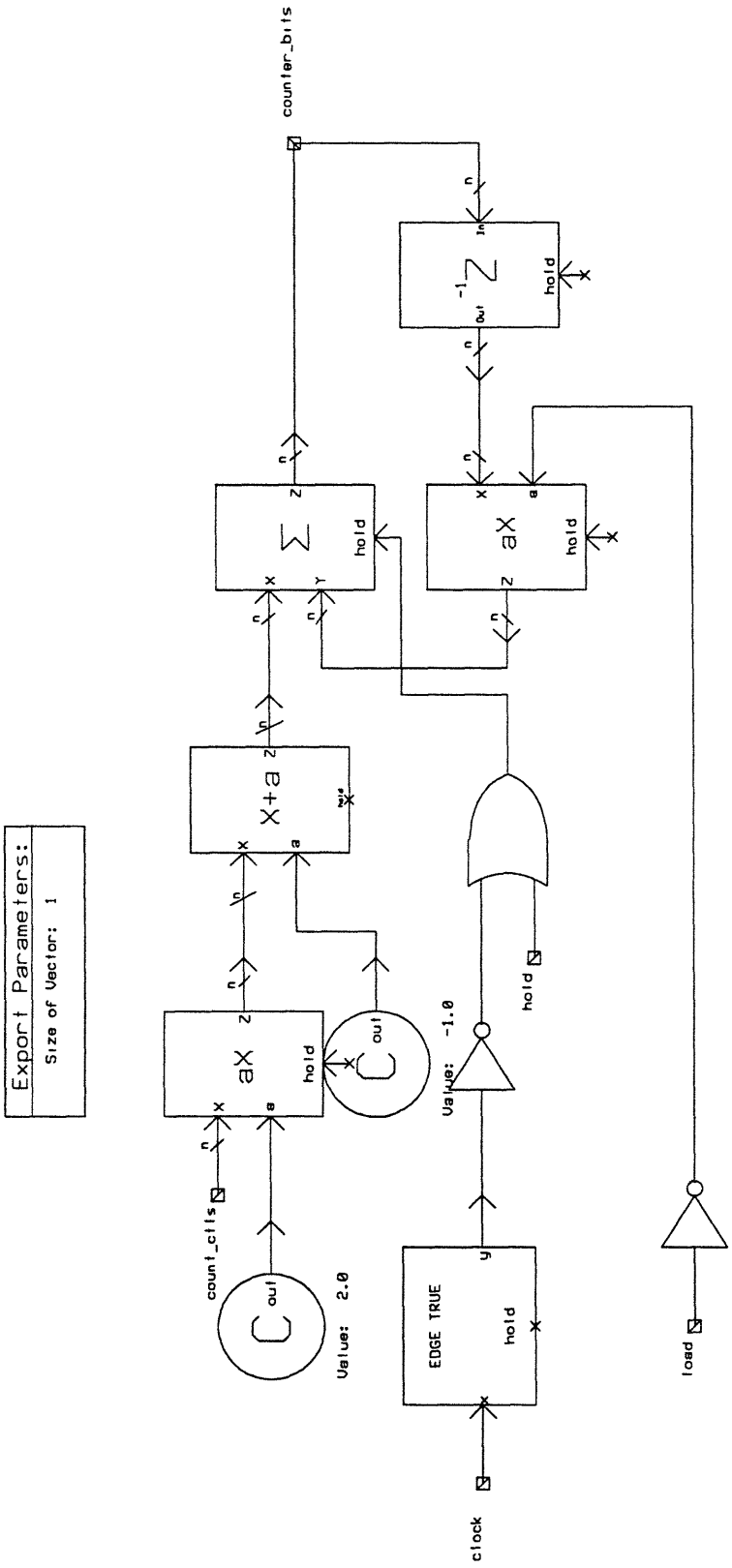

Figure B.23: pn_comp_count block diagram.

Block Name: **pn_delays**

Synopsis:

Input Signals:

none

Output Signals:

delays: A vector signal consisting of the 5 component delays.

Parameters:

c1_dly: The delay on PN component 1.
c2_dly: The delay on PN component 2.
c3_dly: The delay on PN component 3.
c4_dly: The delay on PN component 4.
c5_dly: The delay on PN component 5.

Functional Description:

This block combines the parameters, etc. necessary to implement setting the delays for each individual PN component into a nice clean block, so it'll look nice on the diagram.

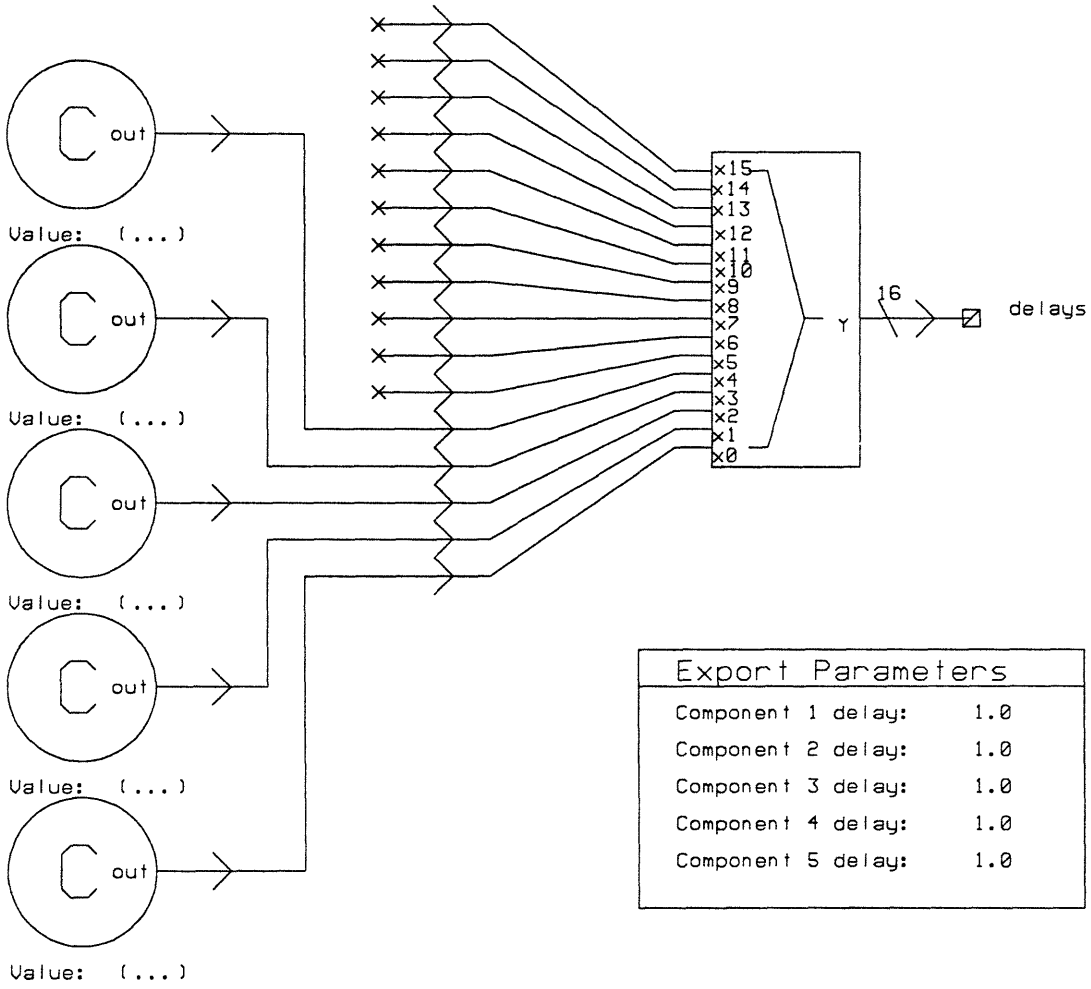

Figure B.24: pn_delays block diagram.

Block Name: **pn_gen2**

Synopsis:

Input Signals:

clk_phase: The phase of the clock signal.
delays: A vector containing the delays for each component of the PN code.
hold: A control signal that allows the output to be held, regardless of the other inputs.
reset: A control signal that allows the generator to be reset to its initial state.

Output Signals:

clock: The clock component of the PN code.
PN_codes: A vector containing the remaining PN code components.
PN_integers: A vector containing the integer representations of the phase of each PN component.

Parameters:

clk_freq: The frequency of the PN clock component.
s_freq: The sampling frequency of the simulation.
7_code: An integer representation of the 7-bit PN code component.
11_code: An integer representation of the 11-bit PN code component.
15_code: An integer representation of the 15-bit PN code component.
19_code: An integer representation of the 19-bit PN code component.
23_code: An integer representation of the 23-bit PN code component.

Functional Description:

This block generates all the PN components, including the clock. The frequency and phase of the clock are adjustable, and the code outputs are all synchronized to the clock. In addition, each code component can be individually delayed, to allow for changes in the phase of each component. In addition to the PN code components, the block also outputs a set of integers that represent the phase of each code component (excluding the clock component).

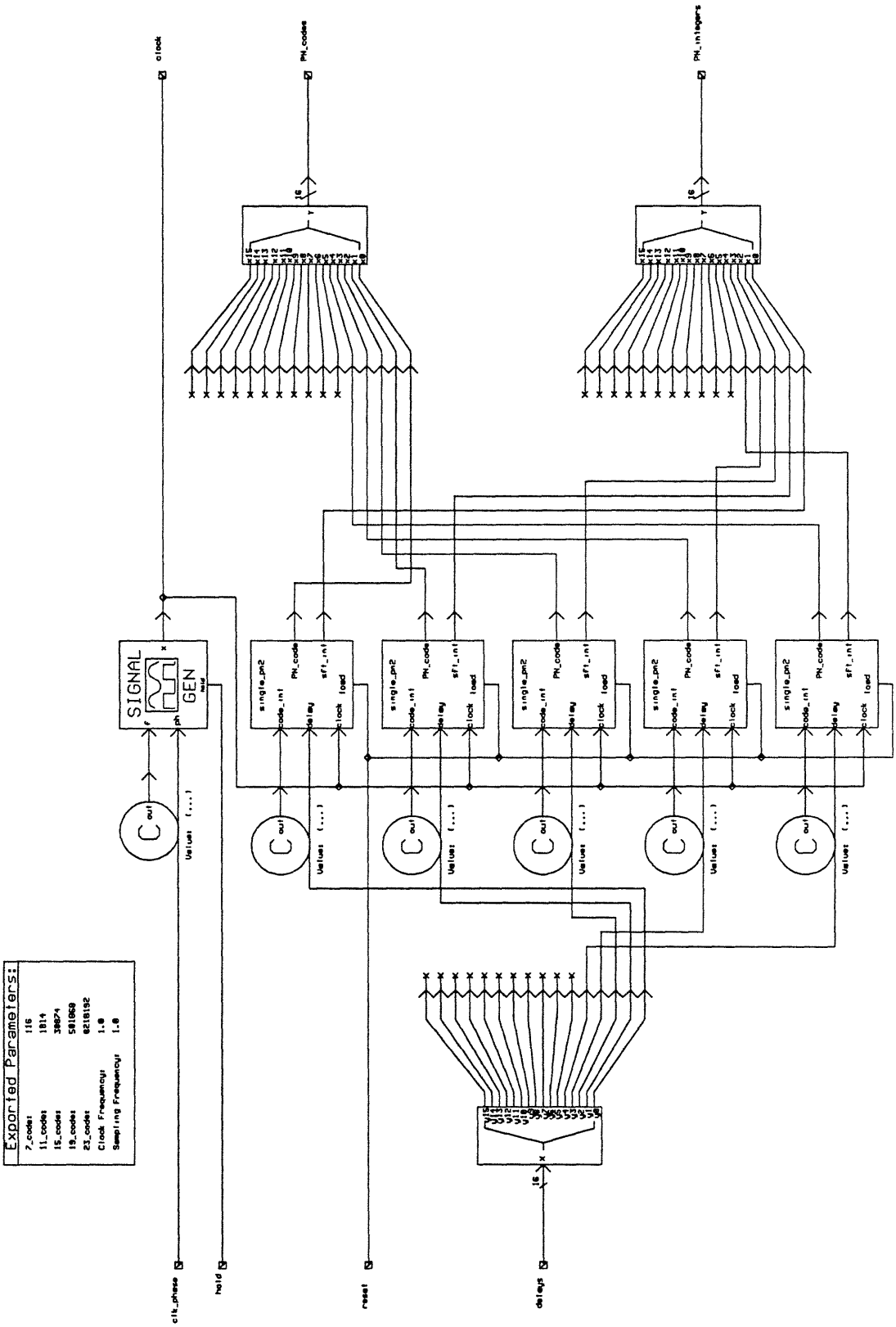

Figure B.25: pn_gen2 block diagram.

Block Name: **pn_generator**

Synopsis:

Input Signals:

delays: A vector containing the delays for each component of the PN code.
clk_reset: A control signal that allows the clock component to be reset.

Output Signals:

clock: The clock component of the PN code.
PN_codes: A vector containing the remaining PN code components.
PN_integers: A vector containing the integer representations of the phase of each PN component.

Parameters:

7_code: An integer representation of the 7-bit PN code component.
11_code: An integer representation of the 11-bit PN code component.
15_code: An integer representation of the 15-bit PN code component.
19_code: An integer representation of the 19-bit PN code component.
23_code: An integer representation of the 23-bit PN code component.

Functional Description:

This block generates all the PN code components, including the clock. The clock is not adjustable, and runs with a period of 2 simulation iterations. Each code component can be individually delayed, to allow for changes in the phase of each component. In addition to the PN code components, the block also outputs a set of integers that represent the phase of each code component (excluding the clock component).

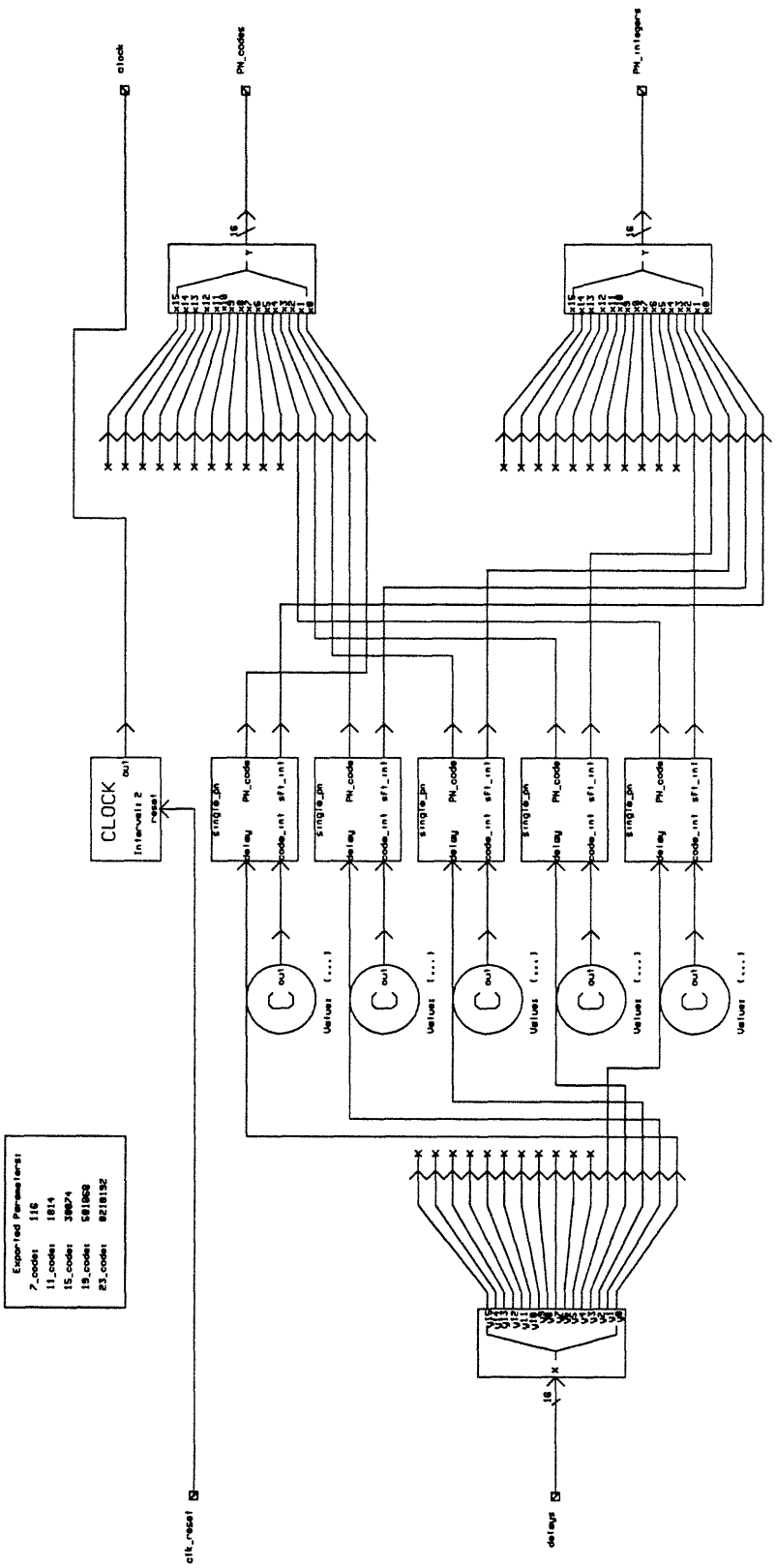

Figure B.26: pn_generator block diagram.

Block Name: **pn_regen**

Synopsis:

Input Signals:

comp*X*_phase: A vector representation of PN component *X*.
hold: A control signal that allows the PN generators to be held.
ld_c*X*: A control signal that indicates when to load the phase of component *X*.
recovered_clock: The recovered clock component of the received PN sequence.

Output Signals:

recovered_PN: The regenerated PN sequence.

Parameters:

c1_len: The length of PN component 1.
c2_len: The length of PN component 2.
c3_len: The length of PN component 3.
c4_len: The length of PN component 4.
c5_len: The length of PN component 5.
shift_len: The amount of rotation required on each component phase vector.

Functional Description:

This block takes the phases of all the PN components from the **comp_rec_wctl** block, generates the PN code, and combines it into the final regenerated PN sequence.

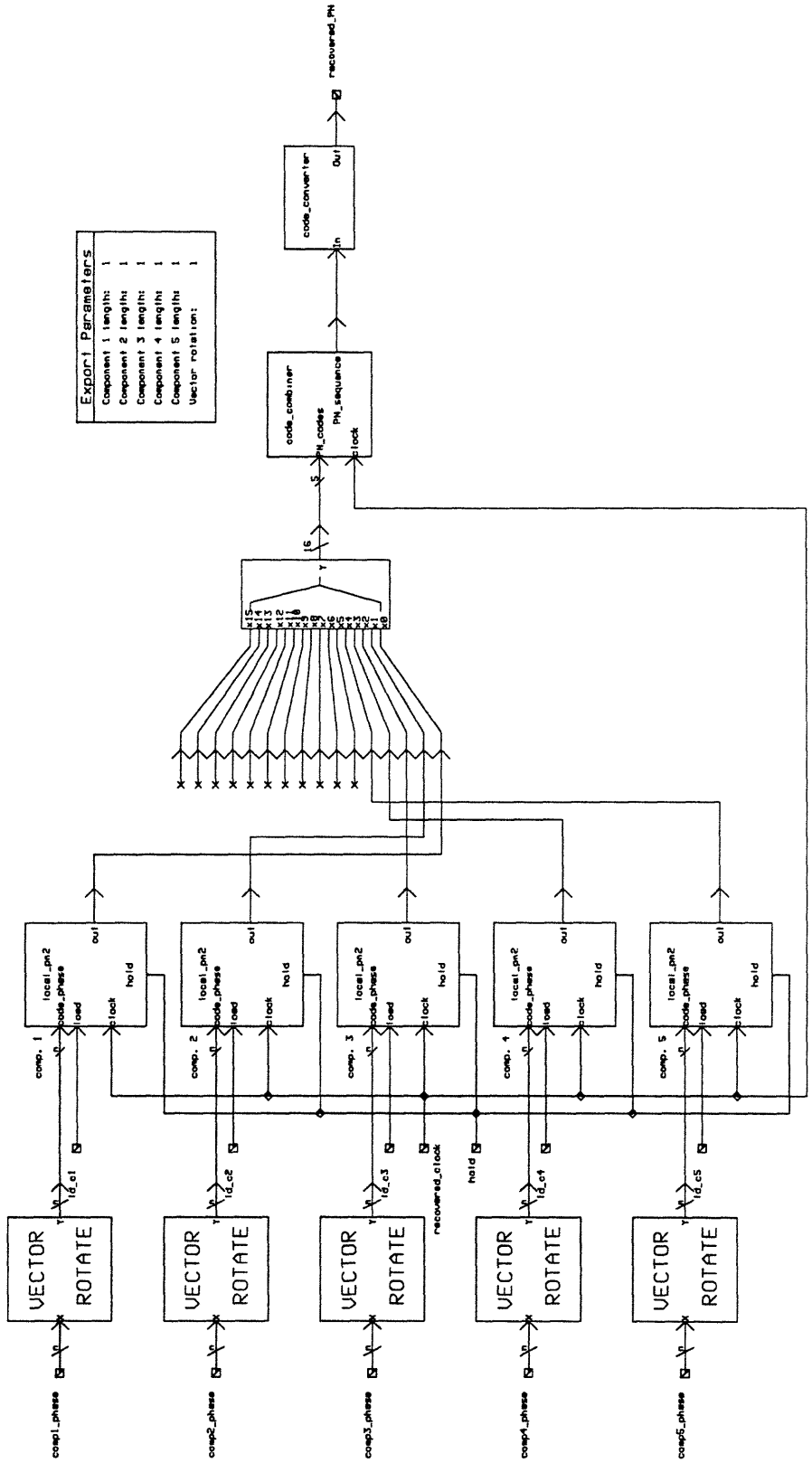

Figure B.27: pn_regen block diagram.

Block Name: pn_source

Synopsis:

Input Signals:

clk_phase: The phase of the clock signal.
delays: A vector containing the delays for each component of the PN code.
hold: A control signal that allows the output to be held, regardless of the other inputs.
reset: A control signal that allows the source to be reset to its initial state.

Output Signals:

PN_sequence: The combined, converted PN code sequence. High is 1, Low is -1.

Parameters:

clk_freq: The frequency of the PN clock component.
s_freq: The sampling frequency for the simulation.

Functional Description:

This block generates, combines, and converts a PN sequence with five components, plus clock. It allows the phase of the clock component and the delays for each individual PN component other than the clock to be input. It outputs a PN sequence that is AC coupled, with a high represented by a 1, and a low represented by a -1.

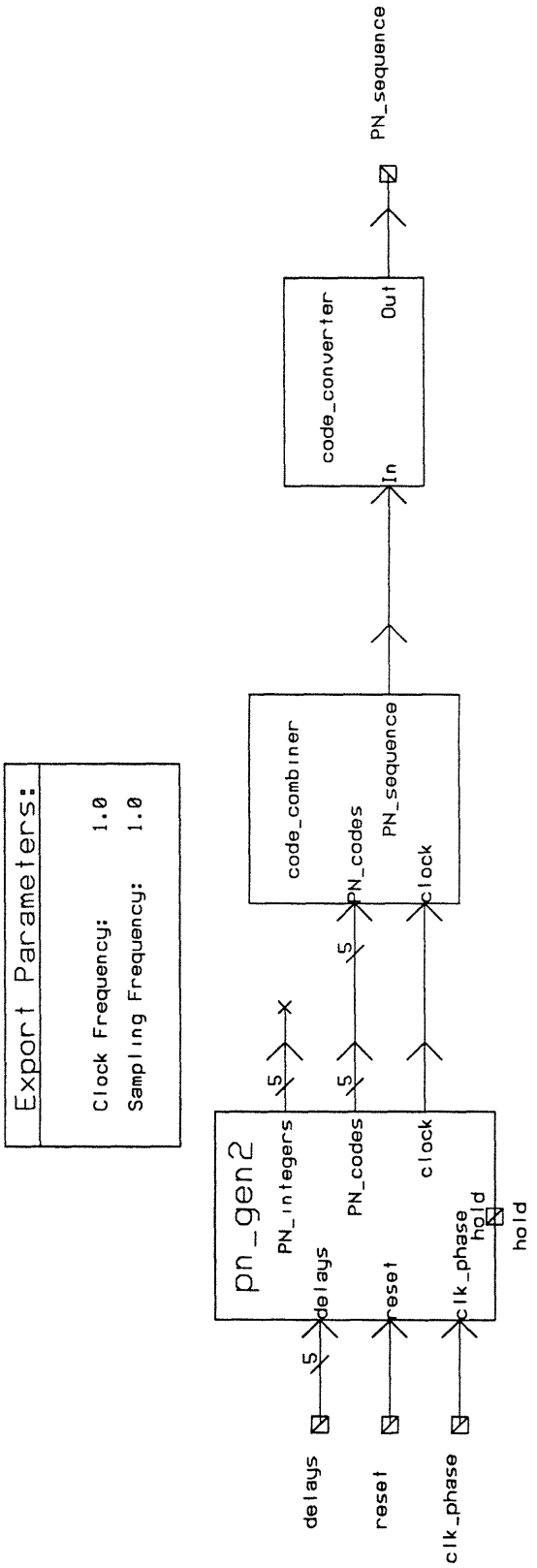

Figure B.28: pn_source block diagram.

Block Name: **pn_source2**

Synopsis:

Input Signals:

clk_phase: The phase of the clock signal.
delays: A vector containing the delays for each component of the PN code.
hold: A control signal that allows the output to be held, regardless of the other inputs.
reset: A control signal that allows the source to be reset to its initial state.

Output Signals:

clock: The clock component of the PN code.
PN_sequence: The combined, converted PN code sequence. High is 1, Low is -1.

Parameters:

clk_freq: The frequency of the PN clock component.
noise_variance: The variance of the Gaussian noise added to the PN sequence.
s_freq: The sampling frequency for the simulation.

Functional Description:

This block generates, combines, and converts a noisy PN sequence with five components, plus clock. It allows the phase of the clock component and the delays for each individual PN component other than the clock to be input. It outputs a PN sequence that is AC coupled, with a high represented by a 1, and a low represented by a -1. White Gaussian noise with the specified variance is added to the output sequence.

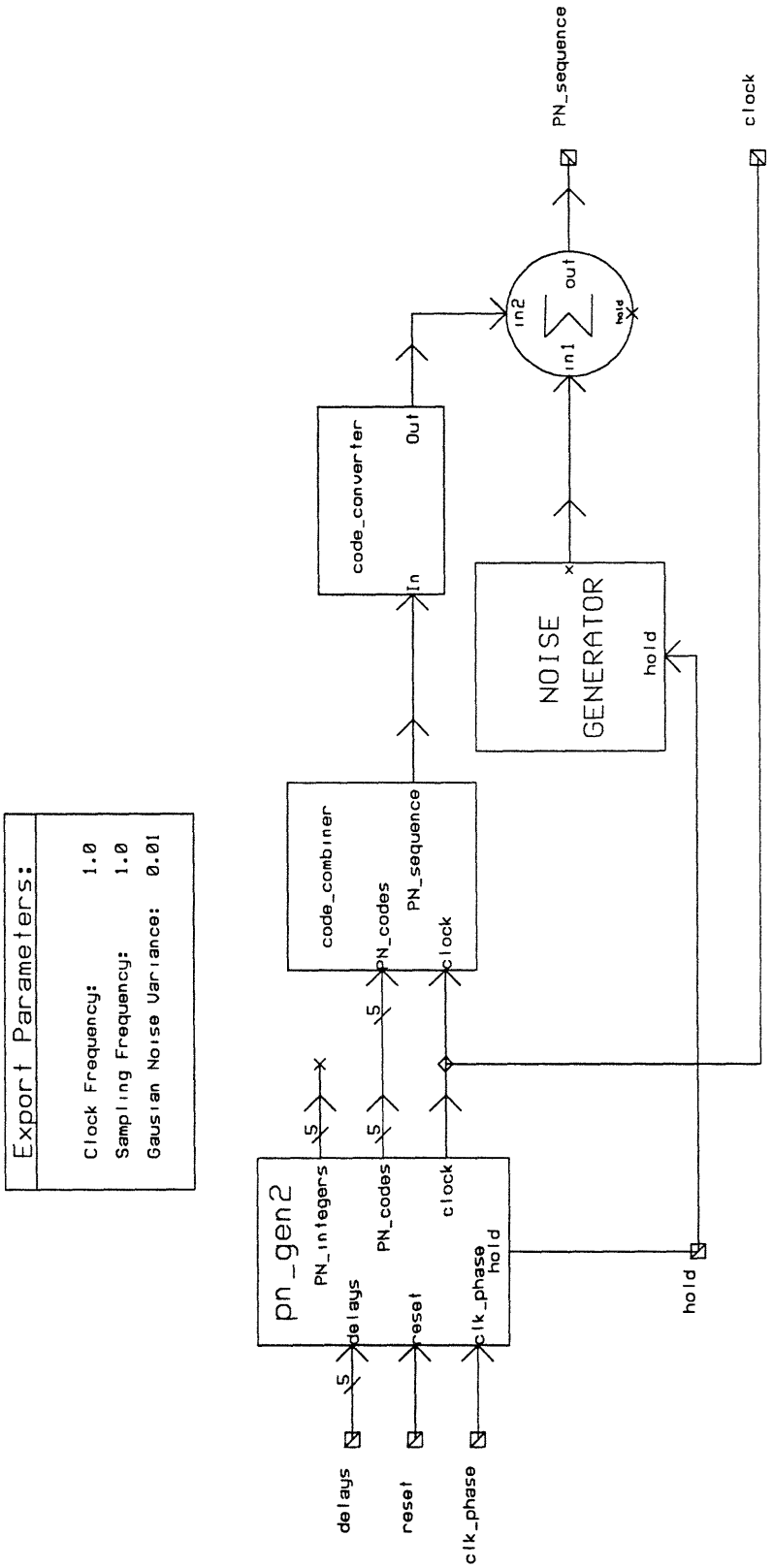

Figure B.29: pn_source2 block diagram.

Block Name: **regen_ranging**

Synopsis:

Input Signals:

PN_sequence: The received PN sequence.
reset: A control signal allowing the system to be reset.

Output Signals:

recovered_PN: The regenerated PN sequence.

Parameters:

accum_per: The number of periods over which the correlation is performed.
c_freq: The receiver carrier frequency.
clk_freq: The PN clock component frequency.
c*X*_int: An integer representation of the binary code of component X.
c*X*_len: The length of component X (in bits).
n, m, div_ratio: The ratio between the clock frequency and the carrier frequency.
s_freq: The sampling frequency.

Functional Description:

This is the high-level block for the entire regenerative ranging system. This block takes the incoming PN sequence and recovers the clock component from it. It then passes this, along with the original received sequence to the component recovery section, where the PN code components are acquired and regenerated. The output is the regenerated PN sequence.

This block contains multirate blocks.

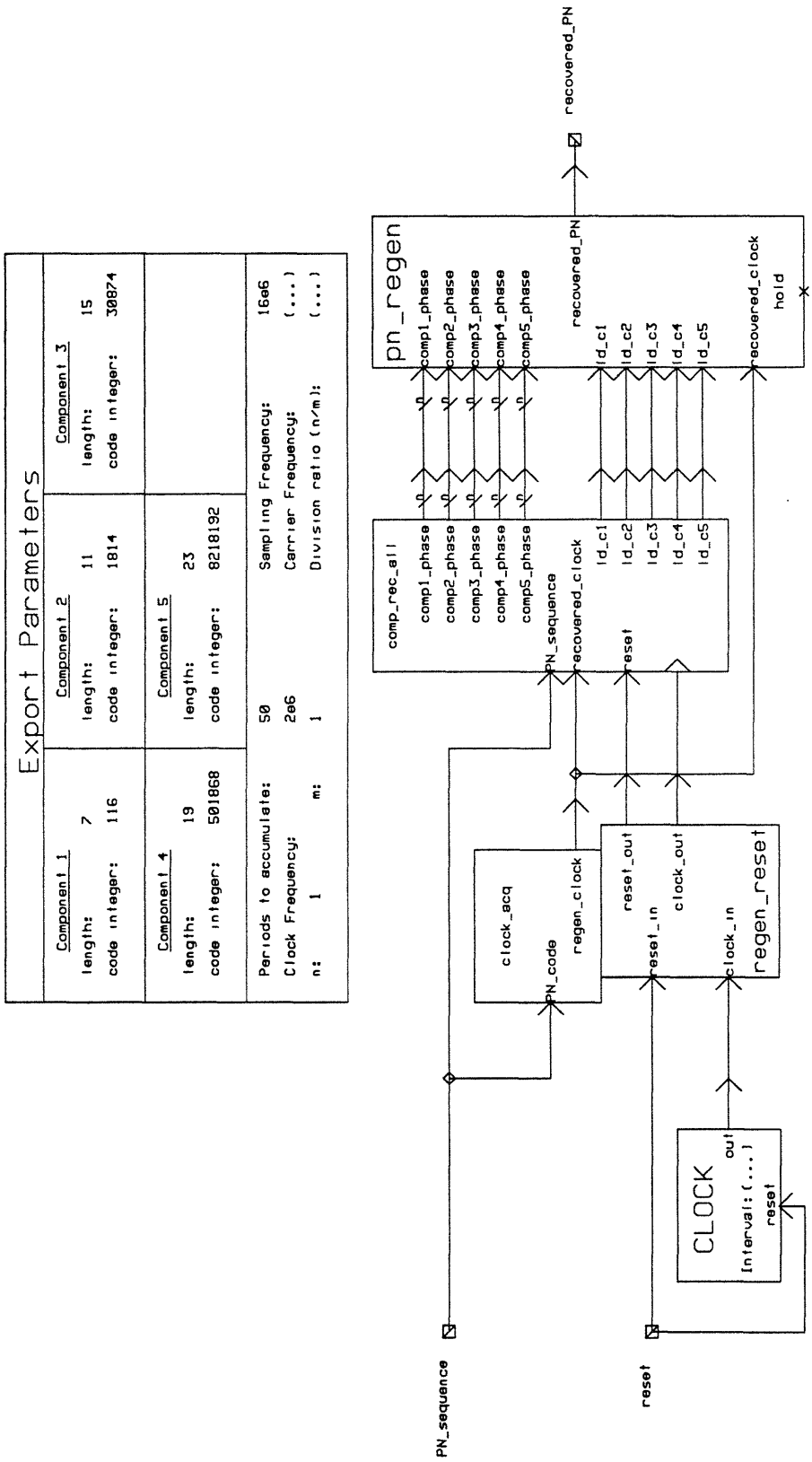

Figure B.30: regen_ranging block diagram.

Block Name: **regen_reset**

Synopsis:

Input Signals:

clock_in: The clock, as input.
reset_in: The system reset signal.

Output Signals:

clock_out: The clock, with an added rising edge to force recognition of the reset.
reset_out: The reset signal to the rest of the system.

Parameters:

none

Functional Description:

This block processes the system reset signal to force the system to recognize it. It holds the reset signal high for a minimum of three iterations, while inserting a rising edge on the clock signal. This will force the remainder of the system to notice the reset signal.

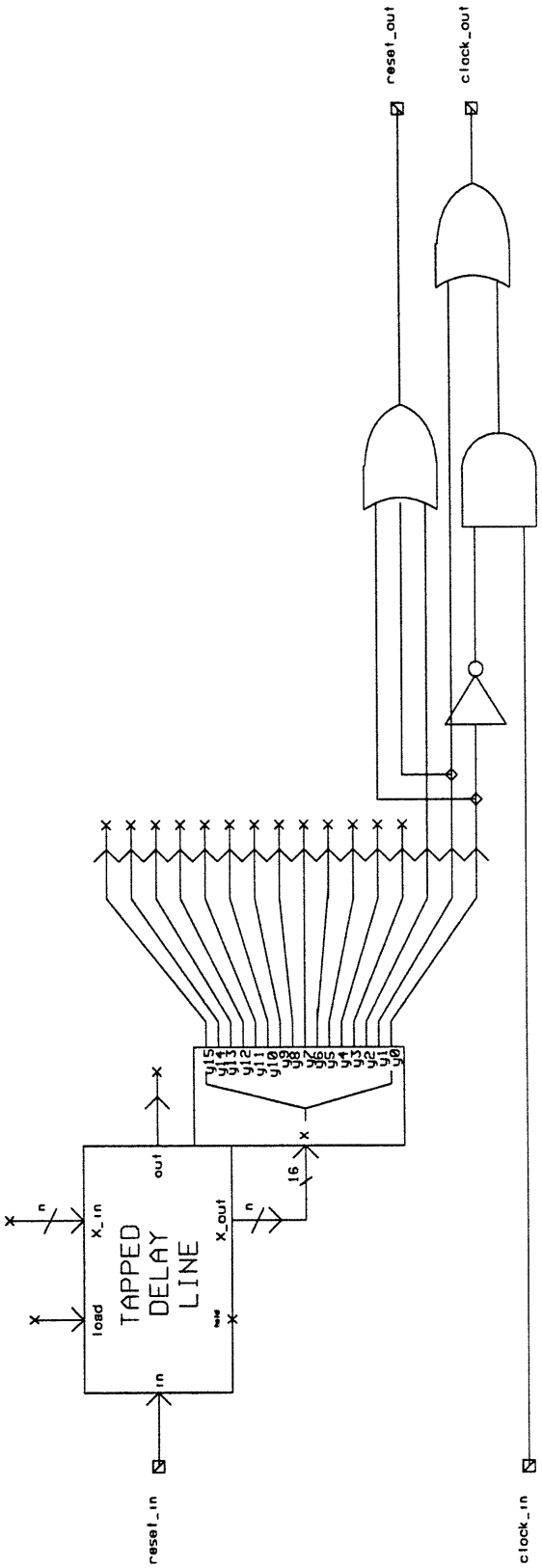

Figure B.31: regen_reset block diagram.

Block Name: **rsum**

Synopsis:

Input Signals:

x: Input to accumulator.
reset: Reset the accumulator memory register.
hold: Hold the memory register output.

Output Signals:

y: Output accumulator value.

Parameters:

none

Functional Description:

This block accumulates the input values. When the **reset** signal goes high the accumulator is reset with the current input value. The output value at reset is the value in the accumulator memory register.

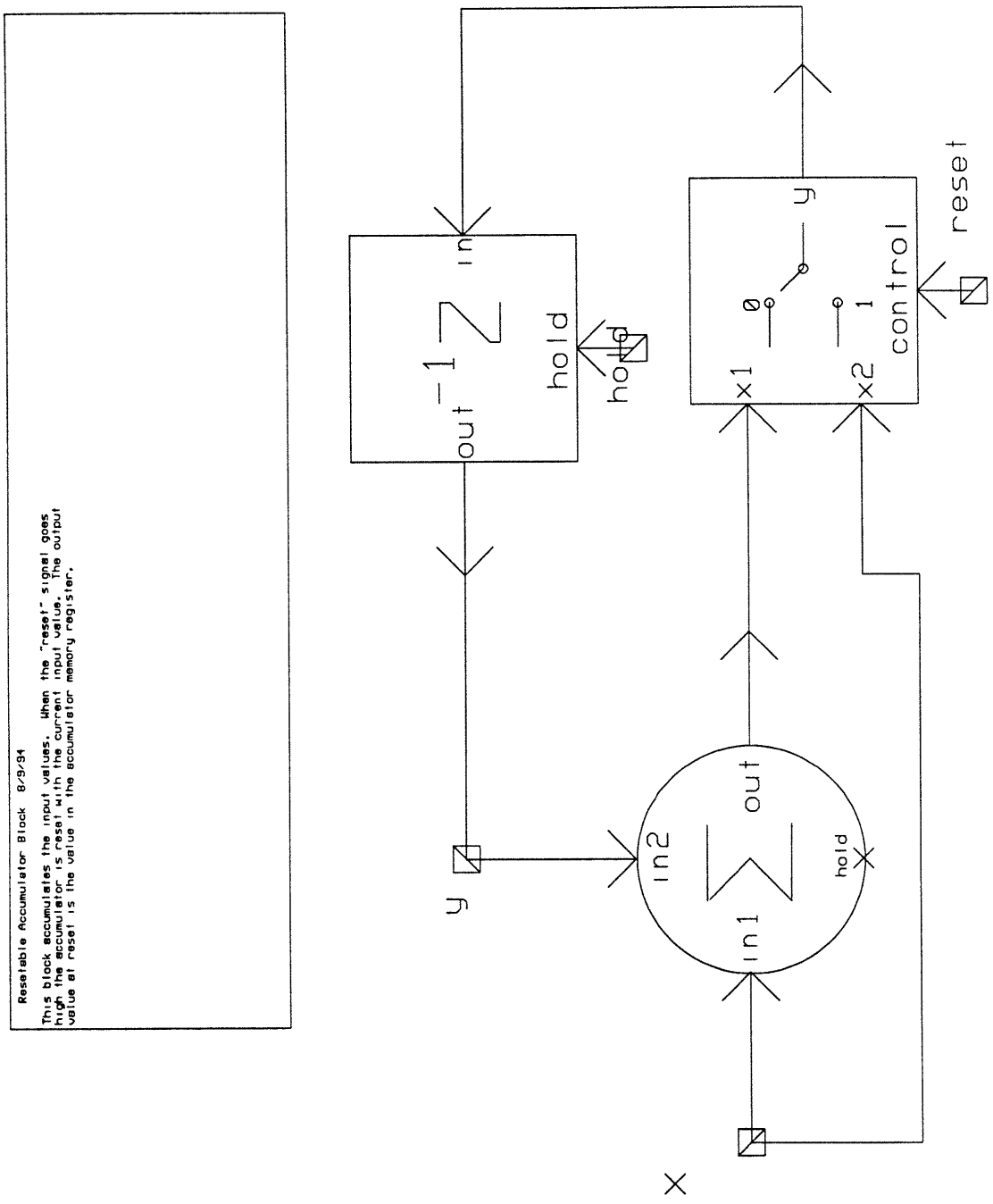

Figure B.32: rsum block diagram.

Block Name: **single_pn**

Synopsis:

Input Signals:

code_int: An integer representation of the PN code component.
delay: The delay for the PN code component.

Output Signals:

PN_code: The PN code component.
sft_int: An integer representation of the phase of the PN code component.

Parameters:

num_bits: The length of the PN code component.

Functional Description:

This block generates a single PN component. It contains a delay that can be varied, allowing the output phase of the component to be adjusted.

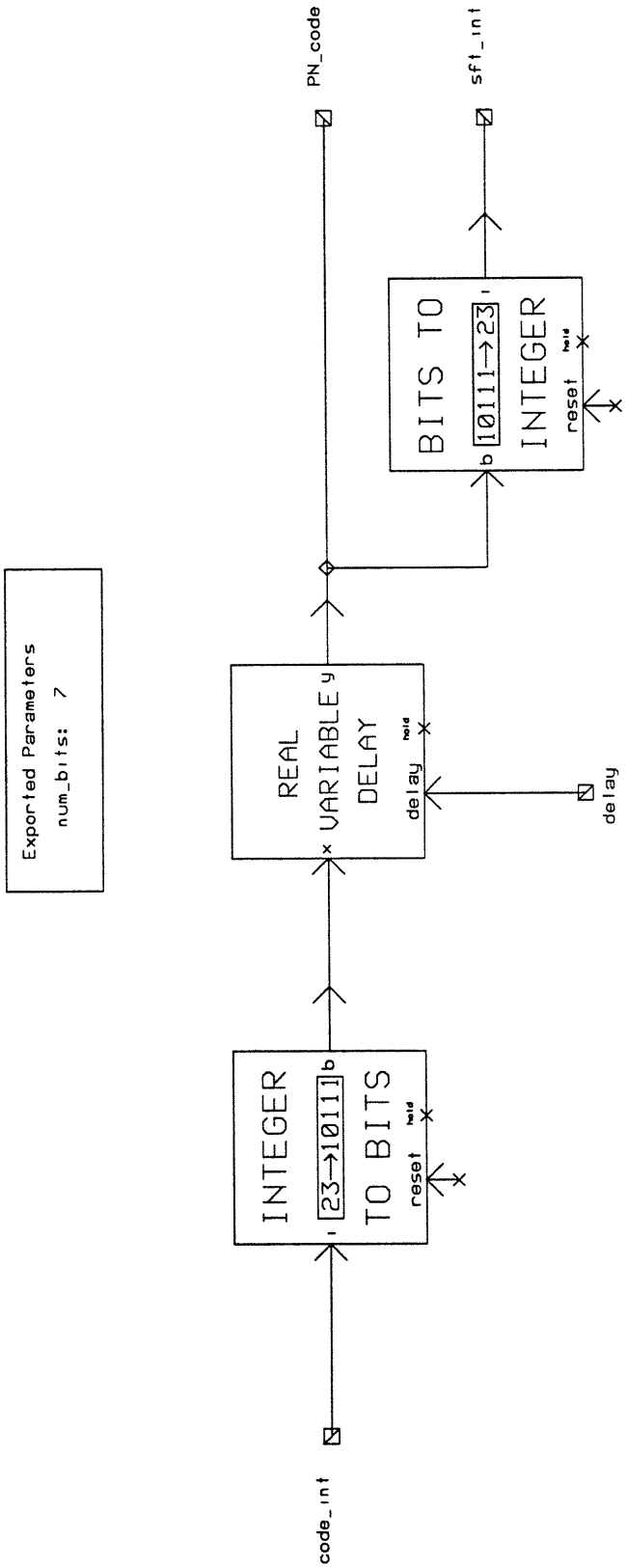

Figure B.33: single_pn block diagram.

Block Name: **single_pn2**

Synopsis:

Input Signals:

code_int: An integer representation of the PN code component.
clock: The PN clock component, to allow synchronization.
delay: The delay for the PN code component.
load: A control signal that allows the shift register to be loaded with the **code_int** input.

Output Signals:

PN_code: The PN code component.
sft_int: An integer representation of the phase of the PN code component.

Parameters:

num_bits: The length of the PN code component.

Functional Description:

This block generates a single PN component, synchronized to the clock. It allows an integer to be loaded into its internal shift register, then shifted out to the output. In addition, the output can be delayed, to permit the phase of the component to be set. The block also outputs an integer that represents the internal state of the shift register.

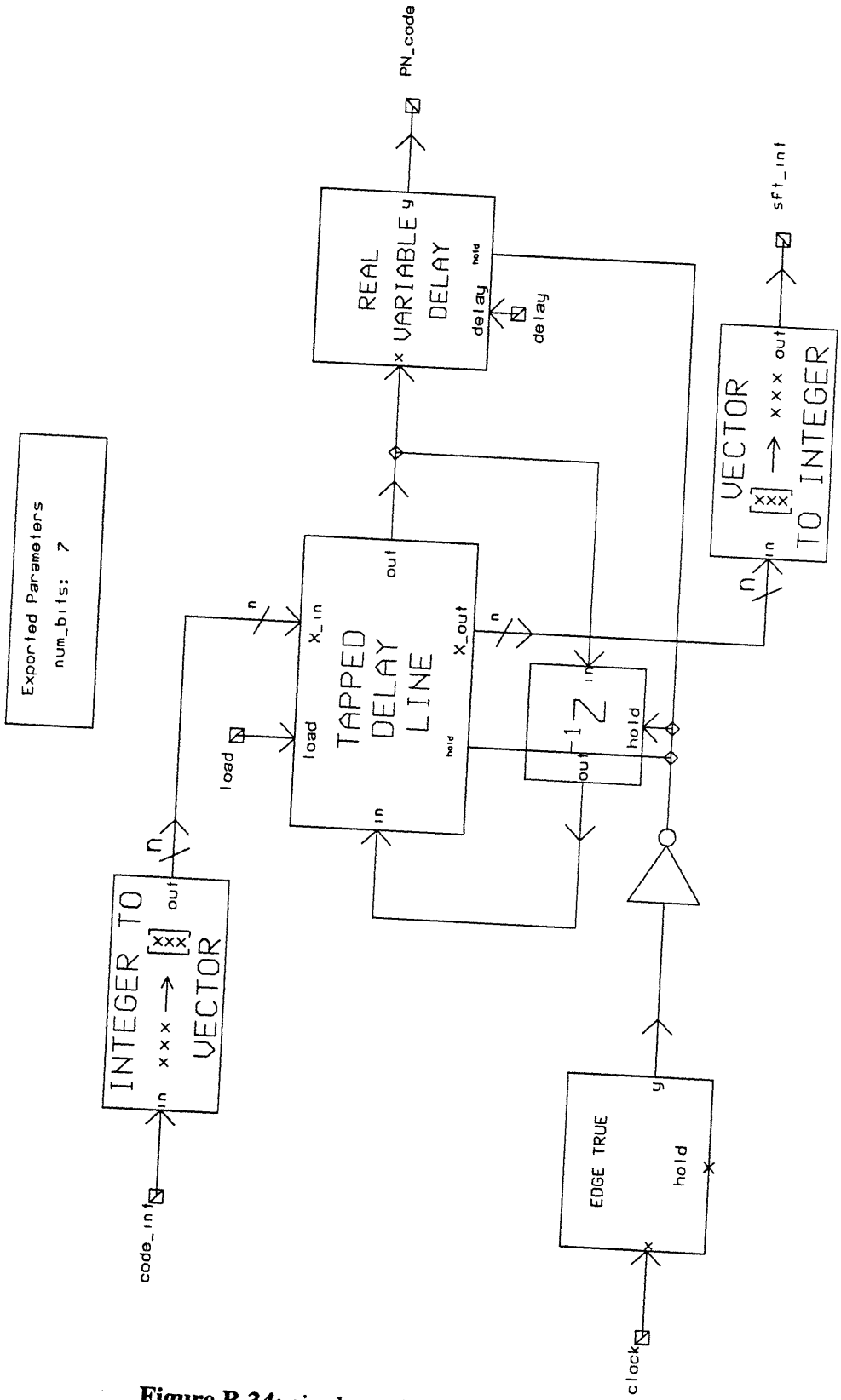

Figure B.34: single_pn2 block diagram.

Block Name: vec2int

Synopsis:

Input Signals:

in: A vector containing a binary representation to be converted.

Output Signals:

out: A decimal representation of the input vector signal.

Parameters:

hold_in_val: The initial value of the output.
high_low: The order in which the binary representation fills the vector (MSB or LSB to the low component).
in_IOVEC_LEN: The number of bits used in the binary representation.

Functional Description:

This block takes a vector containing a binary representation of an integer as input, and outputs the integer on the same simulation iteration. It expects the low component of the vector to contain either the LSB or the MSB of the binary representation, according to the **high_low** parameter.

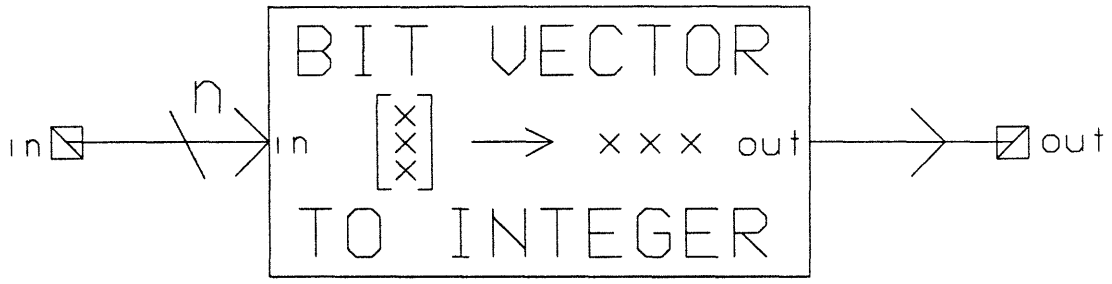

Figure B.35: vec2int block symbol.

VECTOR TO INTEGER BLOCK PARAMETERS

MAIN PARAMETERS:

Number of bits per integer (size of vector)	16
Low component of vector is (MSB/LSB):	'MSB'

MISCELLANEOUS PARAMETERS:

Initial value	0.0

ALL_FEED_THROUGH in

Figure B.36: vec2int block parameters.

vec2int block code

vec2int.c

```
#include "spw_platform.h"
#ifdef UNIX
#include "caedata/regen_ranging/vec2int/blockcode/vec2intu.c"
#else
#ifdef VAX_VMS
#include "[caedata.regen_ranging.vec2int.blockcode]vec2intu.c"
#endif VAX_VMS
#endif UNIX
static char *REVISION = "2.50";

/*
 *
 *       Block Function: vec2int
 *       Library: regen_ranging
 *       Date: Fri Aug 19 13:40:53 1994
 *
 */
```

```
/*********************************************************************/
/*                                                                   */
/*    FEED_THROUGH_LIST INFORMATION:                                 */
/*                                                                   */
/*    --> FEED_THROUGH_TYPE IS NOT EDITABLE. The BDE parameter       */
/*        screen associated with the block must be edited to change  */
/*        the block's FEED_THROUGH_TYPE.                             */
/*                                                                   */
/*        FEED_THROUGH_TYPE = ALL_FEED_THROUGH.                      */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/*    LINK_OPTIONS INFORMATION:                                      */
/*                                                                   */
/*    --> The LINK_OPTIONS list is editable.  It contains all the    */
/*        libraries which the code must be linked to.  Each item in  */
/*        the list must be surrounded by double quotes and           */
/*        separated by commas.  The math library is automatically    */
/*        linked, and does not need to be specified.  The paths      */
/*        may be specified as full paths or as paths relative to     */
/*        the host.                                                  */
/*        A link option can also be specified in the form "-lx"      */
/*        (where x is defined in the UNIX manual on "ld"             */
/*                                                                   */
/*        IMPORTANT:  The entire LINK_OPTIONS list must be deleted   */
/*        if it doesn't contain any elements.                        */
/*                                                                   */
/*        Sample LINK_OPTIONS list:                                  */
/*        (Actual list should be placed below this comment block)    */
/*                                                                   */
/*        LINK_OPTIONS = { "-lm",                                    */
/*                         "//host/code/lib/sample.a" };             */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/*    INCLUDE_DIRS INFORMATION:                                      */
/*                                                                   */
/*    --> The INCLUDE_DIRS list is editable.  The list should        */
/*        contain all directory search paths needed to locate all    */
/*        the include files used by this block.  It has the same     */
```

```
/*        format as the LINK_OPTIONS list.                                */
/*                                                                        */
/*        IMPORTANT:  The entire INCLUDE_DIRS list must be deleted        */
/*        if it doesn't contain any elements.                             */
/*                                                                        */
/*        Sample INCLUDE_DIRS list:                                       */
/*        (Actual list should be placed below this comment block)         */
/*                                                                        */
/*        INCLUDE_DIRS = { "//host/u/code/include",                       */
/*                         "//host/lib/dir" };                            */
/*                                                                        */
/*                                                                        */
/**************************************************************************/

/**************************************************************************/
/*                                                                        */
/*       EDITABLE FUNCTIONS                                               */
/*                  --> In_vec2int_regen_ranging ()
/*                  --> Ro_vec2int_regen_ranging ()
/*                  --> Te_vec2int_regen_ranging ()
/*                                                                        */
/*       Structure use:                                                   */
/*         Typical input value reference                                  */
/*           local_var = *(spb_input->var_name);                          */
/* **OR**    local_var = I_var_name;    */
/*         Typical output value update                                    */
/*           spb_output->var_name = local_var;                            */
/* **OR**    O_var_name = local_var;    */
/*         Typical parameter reference                                    */
/*           local_var = spb_parm->var_name;                              */
/* **OR**    local_var = P_var_name;    */
/*                                                                        */
/*       (See reference manual for further information)                   */
/*                                                                        */
/**************************************************************************/

/*
 *             Initialize Function (must be present)
 *             --> If editing, modify only the lines within the
 *                 function's opening and closing brackets.
 *
 *             This function is used to initialize the state structure
 *                  and constant outputs of the block.  It is called once
 *                   for each block instance during simulation.
 *
 *             Function must always return either SYS_OK, SYS_TERM,
```

```
 *                  or SYS_FATAL by using the return() function.
 *                  User may modify the line containing
"return(SYS_OK);".
 */

In_vec2int_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec2int_regen_ranging *spb_parm;
STRUCT It_vec2int_regen_ranging *spb_input;
STRUCT Ot_vec2int_regen_ranging *spb_output;
STRUCT St_vec2int_regen_ranging *spb_state;
{
  O_out = P_hold_in_val;

  return (SYS_OK);
}

/*
 *             Run Output Function (must be present)
 *             --> If editing, modify only the lines within the
 *                 function's opening and closing brackets.
 *
 *             This function is used to update the outputs and/or state
 *                  of the block.  It is called each iteration, for each
 *                  block instance during simulation.
 *
 *             Function must always return either SYS_OK, SYS_TERM,
 *                  or SYS_FATAL by using the return() function.
 *                  User may modify the line containing
"return(SYS_OK);".
 */

Ro_vec2int_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec2int_regen_ranging *spb_parm;
STRUCT It_vec2int_regen_ranging *spb_input;
STRUCT Ot_vec2int_regen_ranging *spb_output;
STRUCT St_vec2int_regen_ranging *spb_state;
{
  int i;
  int temp_out = 0;

  if (*P_high_low == 'L')
    for (i = (I_in_iovec_len -1) ; i >= 0 ; i--)
      {
         temp_out = temp_out << 1;
         temp_out += VEC_GET(I_in,i);
      }
```

```
  else
    for (i = 0 ; i < I_in_iovec_len ; i++)
      {
         temp_out = temp_out << 1;
         temp_out += VEC_GET(I_in,i);
      }
  O_out = (double) temp_out;

  return (SYS_OK);
}

/*
 *              Termination Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to dump the final state of the
 *                   block.  It is called once for each block instance
 *                   during the simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Te_vec2int_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec2int_regen_ranging *spb_parm;
STRUCT It_vec2int_regen_ranging *spb_input;
STRUCT Ot_vec2int_regen_ranging *spb_output;
STRUCT St_vec2int_regen_ranging *spb_state;
{

        return (SYS_OK);
}

/*****************************************************************/
/*                                                               */
/*      Add any additional functions you need here.              */
/*                                                               */
/*****************************************************************/
```

vec2int.h

```
#include "FBCDEFS.h"

/*
 *
 *       Block Function: vec2int
 *       Library: regen_ranging
 *       Date: Fri Aug 19 13:40:53 1994
 *
 */

/*****************************************************************/
/*                                                               */
/*       EDITABLE USER DEFINED STATE STRUCTURE                   */
/*                  --> STRUCT St_vec2int_regen_ranging
/*                                                               */
/*****************************************************************/

/*
 *                State Structure (User Defined, editable)
 */
STRUCT St_vec2int_regen_ranging {
        int instance;
};

/*****************************************************************/
/*                                                               */
/*       UNEDITABLE SIMULATOR DEFINED STRUCTURES                 */
/*                  --> STRUCT Pt_vec2int_regen_ranging
/*                  --> STRUCT It_vec2int_regen_ranging
/*                  --> STRUCT Ot_vec2int_regen_ranging
/*                                                               */
/*****************************************************************/
/*
 *                Parameter Structure, Simulator Defined, uneditable
 */
STRUCT Pt_vec2int_regen_ranging {
        char * high_low;
        double hold_in_val;
};

/*
 *                Input Structure, Simulator Defined, uneditable
 */
STRUCT It_vec2int_regen_ranging {
```

```
        long in_iovec_len;
        double *in;
};

/*
 *                Output Structure, Simulator Defined, uneditable
 */
STRUCT Ot_vec2int_regen_ranging {
        double out;
};

/*****************************************************************/
/*                                                               */
/*       The following #defines may be used to shorten           */
/*       references to members of the above structures.          */
/*                                                               */
/*****************************************************************/
#define P_high_low (spb_parm->high_low)
#define P_hold_in_val (spb_parm->hold_in_val)
#define I_in_iovec_len (spb_input->in_iovec_len)
#define I_in (spb_input->in)
#define O_out (spb_output->out)
```

Block Name: vec_logic

Synopsis:

Input Signals:

in1: A vector of logic values.
in2: A vector of logic values.

Output Signals:

out: A vector of results.

Parameters:

DEFAULT_VECLEN: The length of the vectors.
hold_in_val: The initial value of each component of the output.
logic_fcn: Which logic function to perform (OR, AND, NOT, XOR, NOR, NAND, or XNOR).

Functional Description:

This block performs component-wise logic functions between two vectors. It places the results into **out**. If the logic function only requires one input (i.e. NOT), the block uses **in1**.

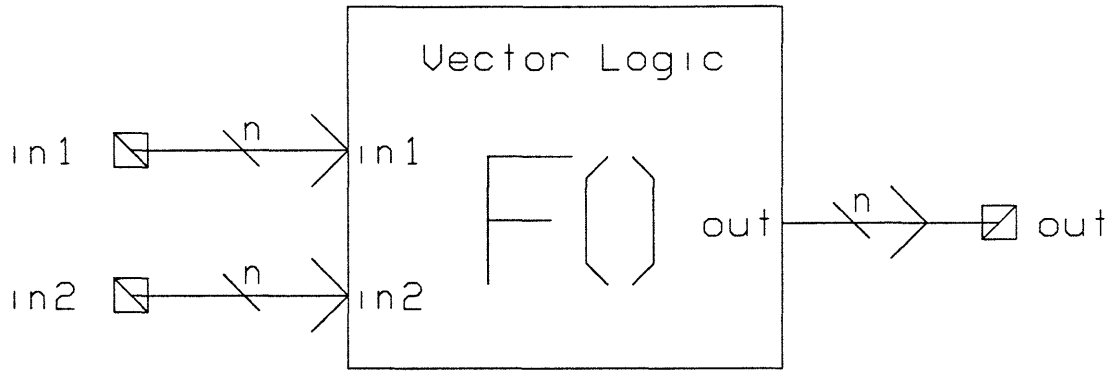

Figure B.37: vec_logic block symbol.

VECTOR LOGIC BLOCK PARAMETERS	
MAIN PARAMETERS:	
Length of Vectors:	16
Logic Function to Perform:	'OR'
MISCELLANEOUS PARAMETERS:	
Initial value	0.0
ALL_FEED_THROUGH	in1, in2, out

Figure B.38: vec_logic block parameters.

vec_logic block code

vec_logic.c

```
#include "spw_platform.h"
#ifdef UNIX
#include "caedata/regen_ranging/vec_logic/blockcode/vec_logicu.c"
#else
#ifdef VAX_VMS
#include "[caedata.regen_ranging.vec_logic.blockcode]vec_logicu.c"
#endif VAX_VMS
#endif UNIX
static char *REVISION = "2.50";

/*
 *
 *       Block Function: vec_logic
 *       Library: regen_ranging
```

```
 *       Date: Mon Aug 29 11:14:25 1994
 *
 */


/*****************************************************************/
/*                                                               */
/*    FEED_THROUGH_LIST INFORMATION:                             */
/*                                                               */
/*    --> FEED_THROUGH_TYPE IS NOT EDITABLE. The BDE parameter   */
/*        screen associated with the block must be edited to change */
/*        the block's FEED_THROUGH_TYPE.                         */
/*                                                               */
/*        FEED_THROUGH_TYPE = ALL_FEED_THROUGH.                  */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*    LINK_OPTIONS INFORMATION:                                  */
/*                                                               */
/*    --> The LINK_OPTIONS list is editable.  It contains all the */
/*        libraries which the code must be linked to.  Each item in */
/*        the list must be surrounded by double quotes and       */
/*        separated by commas.  The math library is automatically */
/*        linked, and does not need to be specified.  The paths  */
/*        may be specified as full paths or as paths relative to */
/*        the host.                                              */
/*        A link option can also be specified in the form "-lx"  */
/*        (where x is defined in the UNIX manual on "ld"         */
/*                                                               */
/*        IMPORTANT:  The entire LINK_OPTIONS list must be deleted */
/*        if it doesn't contain any elements.                    */
/*                                                               */
/*        Sample LINK_OPTIONS list:                              */
/*        (Actual list should be placed below this comment block) */
/*                                                               */
/*        LINK_OPTIONS = { "-lm",                                */
/*                         "//host/code/lib/sample.a" };         */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*    INCLUDE_DIRS INFORMATION:                                  */
```

```
/*                                                                    */
/*    --> The INCLUDE_DIRS list is editable.  The list should         */
/*        contain all directory search paths needed to locate all     */
/*        the include files used by this block.  It has the same      */
/*        format as the LINK_OPTIONS list.                            */
/*                                                                    */
/*        IMPORTANT:  The entire INCLUDE_DIRS list must be deleted    */
/*        if it doesn't contain any elements.                         */
/*                                                                    */
/*        Sample INCLUDE_DIRS list:                                   */
/*        (Actual list should be placed below this comment block)     */
/*                                                                    */
/*        INCLUDE_DIRS = { "//host/u/code/include",                   */
/*                         "//host/lib/dir" };                        */
/*                                                                    */
/*                                                                    */
/**********************************************************************/

/**********************************************************************/
/*                                                                    */
/*       EDITABLE FUNCTIONS                                           */
/*                  --> In_vec_logic_regen_ranging ()
/*                  --> Ro_vec_logic_regen_ranging ()
/*                  --> Te_vec_logic_regen_ranging ()
/*                                                                    */
/*       Structure use:                                               */
/*         Typical input value reference                              */
/*           local_var = *(spb_input->var_name);                      */
/* **OR**    local_var = I_var_name;    */
/*         Typical output value update                                */
/*           spb_output->var_name = local_var;                        */
/* **OR**    O_var_name = local_var;    */
/*         Typical parameter reference                                */
/*           local_var = spb_parm->var_name;                          */
/* **OR**    local_var = P_var_name;    */
/*                                                                    */
/*       (See reference manual for further information)               */
/*                                                                    */
/**********************************************************************/

/*
 *            Initialize Function (must be present)
 *            --> If editing, modify only the lines within the
 *                function's opening and closing brackets.
 *
 *            This function is used to initialize the state structure
```

```
 *                  and constant outputs of the block.  It is called once
 *                   for each block instance during simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

In_vec_logic_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec_logic_regen_ranging *spb_parm;
STRUCT It_vec_logic_regen_ranging *spb_input;
STRUCT Ot_vec_logic_regen_ranging *spb_output;
STRUCT St_vec_logic_regen_ranging *spb_state;
{
  int i;

  for (i=0 ; i < O_out_iovec_len ; i++)
    VEC_SET(O_out,i,P_hold_in_val);

  return (SYS_OK);
}

/*
 *              Run Output Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to update the outputs and/or state
 *                   of the block.  It is called each iteration, for each
 *                   block instance during simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Ro_vec_logic_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec_logic_regen_ranging *spb_parm;
STRUCT It_vec_logic_regen_ranging *spb_input;
STRUCT Ot_vec_logic_regen_ranging *spb_output;
STRUCT St_vec_logic_regen_ranging *spb_state;
{
  int i, fcn;
  int in1, in2;
```

```
  double temp_out;

  fcn = vec_logic_parse(P_logic_fcn);

  for (i=0; i < O_out_iovec_len ; i++)
    {
      in1 = (int) VEC_GET(I_in1,i);
      in2 = (int) VEC_GET(I_in2,i);
      if (in1 <= 0) in1=0;
      if (in2 <= 0) in2=0;
      switch(fcn)
         {
         case -1: /* Doesn't match a programmed function */
           strcpy(wmsgbuf, "Error: bad logic_fcn parameter specified to
Vector Logic Block!");
           wmsgErrors(wmsgbuf);
           return(SYS_TERM);
         case 0:  /* OR */
           temp_out = (double) (in1 || in2);
           break;
         case 1:  /* AND */
           temp_out = (double) (in1 && in2);
           break;
         case 2:  /* NOT */
           temp_out = (double) (! in1);
           break;
         case 3:  /* XOR */
           temp_out = (double) vec_logic_xor(in1,in2);
           break;
         case 4:  /* NOR */
           temp_out = (double) (! (in1 || in2));
           break;
         case 5:  /* NAND */
           temp_out = (double) (! (in1 && in2));
           break;
         case 6:  /* XNOR */
           temp_out = (double) (! vec_logic_xor(in1,in2));
           break;
         }
      VEC_SET(O_out,i,temp_out);
    }

  return (SYS_OK);
}

/*
```

```
 *              Termination Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to dump the final state of the
 *                   block.  It is called once for each block instance
 *                   during the simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Te_vec_logic_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec_logic_regen_ranging *spb_parm;
STRUCT It_vec_logic_regen_ranging *spb_input;
STRUCT Ot_vec_logic_regen_ranging *spb_output;
STRUCT St_vec_logic_regen_ranging *spb_state;
{
         return (SYS_OK);
}
/*************************************************************************/
/*                                                                       */
/*      Add any additional functions you need here.                      */
/*                                                                       */
/*************************************************************************/

vec_logic_parse(str)
char * str;
{
  int i;

  switch(*str)
    {
    case 'O':     /* OR */
      i = 0;
      break;
    case 'A':     /* AND */
      i = 1;
      break;
    case 'N':
      switch(*(str+1))
         {
         case 'O':
           switch(*(str+2))
```

```
            {
            case 'T':/* NOT */
              i=2;
              break;
            case 'R':/* NOR */
              i=4;
              break;
            default:
              i=-1;
            }
          break;
        case 'A':/* NAND */
          i = 5;
          break;
        default:
          i=-1;
        }
      break;
    case 'X':
      switch(*(str+1))
        {
        case 'O':/* XOR */
          i = 3;
          break;
        case 'N':/* XNOR */
          i = 6;
          break;
        default:
          i = -1;
        }
      break;
    default:
      i=-1;
    }

  return i;
}

vec_logic_xor(a, b)
int a,b;
{
  int out;

  out = ((a && (!b)) || ((!a) && b));
  return(out);
}
```

vec_logic.h

```
#include "FBCDEFS.h"

/*
 *
 *       Block Function: vec_logic
 *       Library: regen_ranging
 *       Date: Mon Aug 29 11:14:25 1994
 *
 */

/*****************************************************************/
/*                                                               */
/*       EDITABLE USER DEFINED STATE STRUCTURE                   */
/*                  --> STRUCT St_vec_logic_regen_ranging
/*                                                               */
/*****************************************************************/

/*
 *                State Structure (User Defined, editable)
 */
STRUCT St_vec_logic_regen_ranging {
        int instance;
};

/*****************************************************************/
/*                                                               */
/*       UNEDITABLE SIMULATOR DEFINED STRUCTURES                 */
/*                  --> STRUCT Pt_vec_logic_regen_ranging
/*                  --> STRUCT It_vec_logic_regen_ranging
/*                  --> STRUCT Ot_vec_logic_regen_ranging
/*                                                               */
/*****************************************************************/
/*
 *                Parameter Structure, Simulator Defined, uneditable
 */
STRUCT Pt_vec_logic_regen_ranging {
        double hold_in_val;
        char * logic_fcn;
};

/*
 *                Input Structure, Simulator Defined, uneditable
 */
```

```
STRUCT It_vec_logic_regen_ranging {
        long in1_iovec_len;
        double *in1;
        long in2_iovec_len;
        double *in2;
};

/*
 *                Output Structure, Simulator Defined, uneditable
 */
STRUCT Ot_vec_logic_regen_ranging {
        long out_iovec_len;
        double *out;
};

/*****************************************************************/
/*                                                               */
/*       The following #defines may be used to shorten           */
/*       references to members of the above structures.          */
/*                                                               */
/*****************************************************************/
#define P_hold_in_val (spb_parm->hold_in_val)
#define P_logic_fcn (spb_parm->logic_fcn)
#define I_in1_iovec_len (spb_input->in1_iovec_len)
#define I_in1 (spb_input->in1)
#define I_in2_iovec_len (spb_input->in2_iovec_len)
#define I_in2 (spb_input->in2)
#define O_out_iovec_len (spb_output->out_iovec_len)
#define O_out (spb_output->out)
```

Block Name: vec_rotate

Synopsis:

Input Signals:

X: An input vector.

Output Signals:

Y: The output vector.

Parameters:

DEFAULT_VECLEN: The size of the vectors.
shift_len: The amount to rotate by.

Functional Description:

This block performs a circular rotation on a vector. For example, rotating the vector [1 2 3 4 5] by 2 yields [4 5 1 2 3]. Rotating by -1 yields [2 3 4 5 1]. This rotation is performed on the input vector each simulation iteration. In other words, there is no delay between the vector being input and the rotated vector being output.

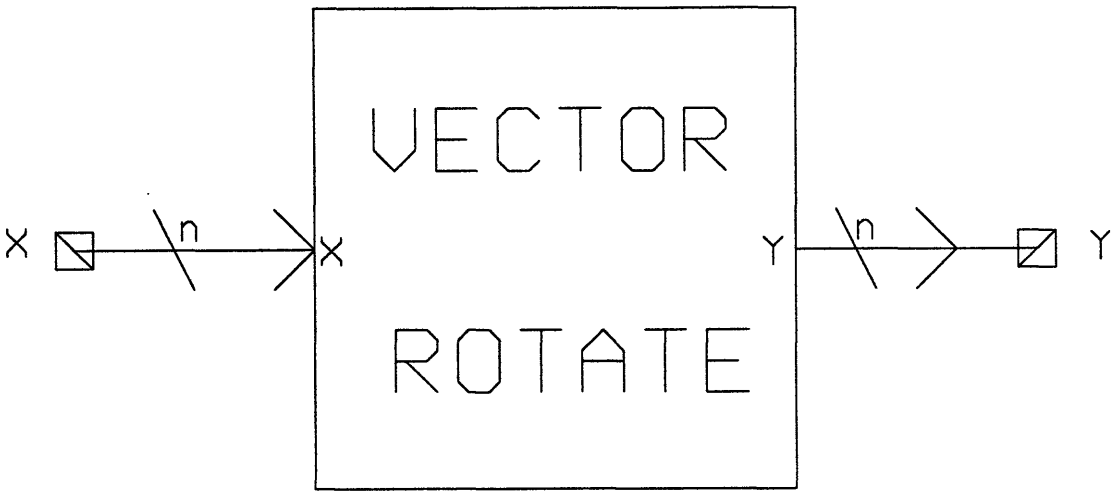

Figure B.39: vec_rotate block symbol.

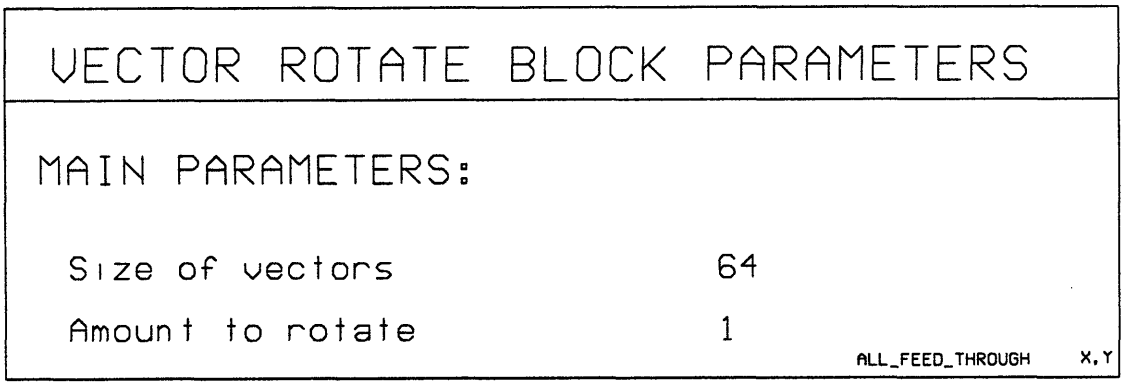

Figure B.40: vec_rotate block parameters.

vec_rotate block code

vec_rotate.c

```
#include "spw_platform.h"
#ifdef UNIX
#include "caedata/regen_ranging/vec_rotate/blockcode/vec_rotateu.c"
#else
#ifdef VAX_VMS
#include "[caedata.regen_ranging.vec_rotate.blockcode]vec_rotateu.c"
#endif VAX_VMS
#endif UNIX
static char *REVISION = "2.50";

/*
 *
 *       Block Function: vec_rotate
 *       Library: regen_ranging
 *       Date: Wed Sep 14 14:32:00 1994
```

```
 *
 */


/*****************************************************************/
/*                                                               */
/*    FEED_THROUGH_LIST INFORMATION:                             */
/*                                                               */
/*    --> FEED_THROUGH_TYPE IS NOT EDITABLE. The BDE parameter   */
/*        screen associated with the block must be edited to change */
/*        the block's FEED_THROUGH_TYPE.                         */
/*                                                               */
/*        FEED_THROUGH_TYPE = ALL_FEED_THROUGH.                  */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*    LINK_OPTIONS INFORMATION:                                  */
/*                                                               */
/*    --> The LINK_OPTIONS list is editable.  It contains all the */
/*        libraries which the code must be linked to.  Each item in */
/*        the list must be surrounded by double quotes and       */
/*        separated by commas.  The math library is automatically */
/*        linked, and does not need to be specified.  The paths  */
/*        may be specified as full paths or as paths relative to */
/*        the host.                                              */
/*        A link option can also be specified in the form "-lx"  */
/*        (where x is defined in the UNIX manual on "ld"         */
/*                                                               */
/*        IMPORTANT:  The entire LINK_OPTIONS list must be deleted */
/*        if it doesn't contain any elements.                    */
/*                                                               */
/*        Sample LINK_OPTIONS list:                              */
/*        (Actual list should be placed below this comment block) */
/*                                                               */
/*        LINK_OPTIONS = { "-lm",                                */
/*                         "//host/code/lib/sample.a" };         */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/*    INCLUDE_DIRS INFORMATION:                                  */
/*                                                               */
```

```
/*    --> The INCLUDE_DIRS list is editable.  The list should       */
/*        contain all directory search paths needed to locate all   */
/*        the include files used by this block.  It has the same    */
/*        format as the LINK_OPTIONS list.                          */
/*                                                                  */
/*        IMPORTANT:  The entire INCLUDE_DIRS list must be deleted  */
/*        if it doesn't contain any elements.                       */
/*                                                                  */
/*        Sample INCLUDE_DIRS list:                                 */
/*        (Actual list should be placed below this comment block)   */
/*                                                                  */
/*        INCLUDE_DIRS = { "//host/u/code/include",                 */
/*                         "//host/lib/dir" };                      */
/*                                                                  */
/*                                                                  */
/********************************************************************/

/********************************************************************/
/*                                                                  */
/*       EDITABLE FUNCTIONS                                         */
/*                  --> In_vec_rotate_regen_ranging ()              */
/*                  --> Ro_vec_rotate_regen_ranging ()              */
/*                  --> Te_vec_rotate_regen_ranging ()              */
/*                                                                  */
/*       Structure use:                                             */
/*         Typical input value reference                            */
/*           local_var = *(spb_input->var_name);                    */
/* **OR**    local_var = I_var_name;    */
/*         Typical output value update                              */
/*           spb_output->var_name = local_var;                      */
/* **OR**    O_var_name = local_var;    */
/*         Typical parameter reference                              */
/*           local_var = spb_parm->var_name;                        */
/* **OR**    local_var = P_var_name;    */
/*                                                                  */
/*       (See reference manual for further information)             */
/*                                                                  */
/********************************************************************/

/*
 *            Initialize Function (must be present)
 *            --> If editing, modify only the lines within the
 *                function's opening and closing brackets.
 *
 *            This function is used to initialize the state structure
 *                and constant outputs of the block.  It is called once
```

```
 *                  for each block instance during simulation.
 *
 *             Function must always return either SYS_OK, SYS_TERM,
 *                  or SYS_FATAL by using the return() function.
 *                  User may modify the line containing
"return(SYS_OK);".
 */

In_vec_rotate_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec_rotate_regen_ranging *spb_parm;
STRUCT It_vec_rotate_regen_ranging *spb_input;
STRUCT Ot_vec_rotate_regen_ranging *spb_output;
STRUCT St_vec_rotate_regen_ranging *spb_state;
{

  return (SYS_OK);
}

/*
 *             Run Output Function (must be present)
 *             --> If editing, modify only the lines within the
 *                 function's opening and closing brackets.
 *
 *             This function is used to update the outputs and/or state
 *                  of the block.  It is called each iteration, for each
 *                  block instance during simulation.
 *
 *             Function must always return either SYS_OK, SYS_TERM,
 *                  or SYS_FATAL by using the return() function.
 *                  User may modify the line containing
"return(SYS_OK);".
 */

Ro_vec_rotate_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec_rotate_regen_ranging *spb_parm;
STRUCT It_vec_rotate_regen_ranging *spb_input;
STRUCT Ot_vec_rotate_regen_ranging *spb_output;
STRUCT St_vec_rotate_regen_ranging *spb_state;
{
  int i,j;

  for (i=0 ; i < I_X_iovec_len ; i++)
    {
      j = vec_rotate_mod((i+P_shift_len),I_X_iovec_len);
      VEC_SET(O_Y,j,VEC_GET(I_X,i));
    }
```

```
  return (SYS_OK);
}

/*
 *              Termination Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to dump the final state of the
 *                   block.  It is called once for each block instance
 *                   during the simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Te_vec_rotate_regen_ranging (spb_parm, spb_input, spb_output, spb_state)
STRUCT Pt_vec_rotate_regen_ranging *spb_parm;
STRUCT It_vec_rotate_regen_ranging *spb_input;
STRUCT Ot_vec_rotate_regen_ranging *spb_output;
STRUCT St_vec_rotate_regen_ranging *spb_state;
{
  return (SYS_OK);
}
/*****************************************************************/
/*                                                               */
/*      Add any additional functions you need here.              */
/*                                                               */
/*****************************************************************/

vec_rotate_mod(a,b)
long int a,b;
{
  long out;

  out = a;
  while(out < 0)
    out += b;
  while(out >= b)
    out -= b;

  return out;
}
```

vec_rotate.h

```
#include "FBCDEFS.h"

/*
 *
 *       Block Function: vec_rotate
 *       Library: regen_ranging
 *       Date: Wed Sep 14 14:32:00 1994
 *
 */

/*****************************************************************/
/*                                                               */
/*       EDITABLE USER DEFINED STATE STRUCTURE                   */
/*                  --> STRUCT St_vec_rotate_regen_ranging       */
/*                                                               */
/*****************************************************************/

/*
 *                State Structure (User Defined, editable)
 */
STRUCT St_vec_rotate_regen_ranging {
        int instance;
};

/*****************************************************************/
/*                                                               */
/*       UNEDITABLE SIMULATOR DEFINED STRUCTURES                 */
/*                  --> STRUCT Pt_vec_rotate_regen_ranging       */
/*                  --> STRUCT It_vec_rotate_regen_ranging       */
/*                  --> STRUCT Ot_vec_rotate_regen_ranging       */
/*                                                               */
/*****************************************************************/
/*
 *                Parameter Structure, Simulator Defined, uneditable
 */
STRUCT Pt_vec_rotate_regen_ranging {
        long shift_len;
};

/*
 *                Input Structure, Simulator Defined, uneditable
 */
STRUCT It_vec_rotate_regen_ranging {
        long X_iovec_len;
```

```
        double *X;
};

/*
 *                Output Structure, Simulator Defined, uneditable
 */
STRUCT Ot_vec_rotate_regen_ranging {
        long Y_iovec_len;
        double *Y;
};

/*****************************************************************/
/*                                                               */
/*       The following #defines may be used to shorten           */
/*       references to members of the above structures.          */
/*                                                               */
/*****************************************************************/
#define P_shift_len (spb_parm->shift_len)
#define I_X_iovec_len (spb_input->X_iovec_len)
#define I_X (spb_input->X)
#define O_Y_iovec_len (spb_output->Y_iovec_len)
#define O_Y (spb_output->Y)
```

Block Name: vec_sca_logic

Synopsis:

Input Signals:

in1: A vector of logic values.
in2: A logic value.

Output Signals:

out: A vector of results.

Parameters:

DEFAULT_VECLEN: The length of the vectors.
hold_in_val: The initial value of each component of the output.
logic_fcn: Which logic function to perform (OR, AND, NOT, XOR, NOR, NAND, or XNOR).

Functional Description:

This block performs logic functions between a scalar (**in2**) and each component of a vector (**in1**). It places the results into **out**. If the logic function only requires one input (i.e. NOT), the block uses **in1**.

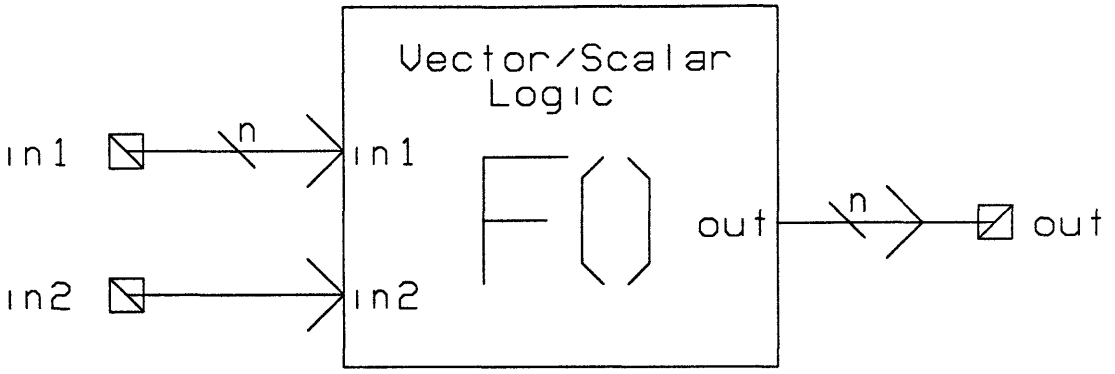

Figure B.41: vec_sca_logic block symbol.

VECTOR/SCALAR LOGIC BLOCK PARAMETERS	
MAIN PARAMETERS:	
Length of Vectors:	16
Logic Function to Perform:	'OR'
MISCELLANEOUS PARAMETERS:	
Initial value	0.0
ALL_FEED_THROUGH	in1,out

Figure B.42: vec_sca_logic block parameters.

vec_sca_logic block code

vec_sca_logic.c

```
#include "spw_platform.h"
#ifdef UNIX
#include "caedata/regen_ranging/vec_sca_logic/blockcode/
vec_sca_logicu.c"
#else
#ifdef VAX_VMS
#include "[caedata.regen_ranging.vec_sca_logic.block-
code]vec_sca_logicu.c"
#endif VAX_VMS
#endif UNIX
static char *REVISION = "2.50";

/*
 *
 *       Block Function: vec_sca_logic
 *       Library: regen_ranging
```

```
 *       Date: Mon Aug 29 16:10:38 1994
 *
 */


/************************************************************************/
/*                                                                      */
/*    FEED_THROUGH_LIST INFORMATION:                                    */
/*                                                                      */
/*    --> FEED_THROUGH_TYPE IS NOT EDITABLE. The BDE parameter          */
/*        screen associated with the block must be edited to change     */
/*        the block's FEED_THROUGH_TYPE.                                */
/*                                                                      */
/*        FEED_THROUGH_TYPE = ALL_FEED_THROUGH.                         */
/*                                                                      */
/************************************************************************/

/************************************************************************/
/*                                                                      */
/*    LINK_OPTIONS INFORMATION:                                         */
/*                                                                      */
/*    --> The LINK_OPTIONS list is editable.  It contains all the       */
/*        libraries which the code must be linked to.  Each item in     */
/*        the list must be surrounded by double quotes and              */
/*        separated by commas.  The math library is automatically       */
/*        linked, and does not need to be specified.  The paths         */
/*        may be specified as full paths or as paths relative to        */
/*        the host.                                                     */
/*        A link option can also be specified in the form "-lx"         */
/*        (where x is defined in the UNIX manual on "ld"                */
/*                                                                      */
/*        IMPORTANT:  The entire LINK_OPTIONS list must be deleted      */
/*        if it doesn't contain any elements.                           */
/*                                                                      */
/*        Sample LINK_OPTIONS list:                                     */
/*        (Actual list should be placed below this comment block)       */
/*                                                                      */
/*        LINK_OPTIONS = { "-lm",                                       */
/*                         "//host/code/lib/sample.a" };                */
/*                                                                      */
/*                                                                      */
/************************************************************************/

/************************************************************************/
/*                                                                      */
/*    INCLUDE_DIRS INFORMATION:                                         */
```

```
/*                                                                    */
/*    --> The INCLUDE_DIRS list is editable.  The list should         */
/*        contain all directory search paths needed to locate all     */
/*        the include files used by this block.  It has the same      */
/*        format as the LINK_OPTIONS list.                            */
/*                                                                    */
/*        IMPORTANT:  The entire INCLUDE_DIRS list must be deleted    */
/*        if it doesn't contain any elements.                         */
/*                                                                    */
/*        Sample INCLUDE_DIRS list:                                   */
/*        (Actual list should be placed below this comment block)     */
/*                                                                    */
/*        INCLUDE_DIRS = { "//host/u/code/include",                   */
/*                         "//host/lib/dir" };                        */
/*                                                                    */
/*                                                                    */
/**********************************************************************/

/**********************************************************************/
/*                                                                    */
/*       EDITABLE FUNCTIONS                                           */
/*                  --> In_vec_sca_logic_regen_ranging ()             */
/*                  --> Ro_vec_sca_logic_regen_ranging ()             */
/*                  --> Te_vec_sca_logic_regen_ranging ()             */
/*                                                                    */
/*       Structure use:                                               */
/*         Typical input value reference                              */
/*           local_var = *(spb_input->var_name);                      */
/* **OR**    local_var = I_var_name;     */
/*         Typical output value update                                */
/*           spb_output->var_name = local_var;                        */
/* **OR**    O_var_name = local_var;     */
/*         Typical parameter reference                                */
/*           local_var = spb_parm->var_name;                          */
/* **OR**    local_var = P_var_name;     */
/*                                                                    */
/*       (See reference manual for further information)               */
/*                                                                    */
/**********************************************************************/

/*
 *             Initialize Function (must be present)
 *             --> If editing, modify only the lines within the
 *                 function's opening and closing brackets.
 *
 *             This function is used to initialize the state structure
```

```
 *                  and constant outputs of the block.  It is called once
 *                  for each block instance during simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

In_vec_sca_logic_regen_ranging (spb_parm, spb_input, spb_output,
spb_state)
STRUCT Pt_vec_sca_logic_regen_ranging *spb_parm;
STRUCT It_vec_sca_logic_regen_ranging *spb_input;
STRUCT Ot_vec_sca_logic_regen_ranging *spb_output;
STRUCT St_vec_sca_logic_regen_ranging *spb_state;
{
  int i;

  for (i=0 ; i < O_out_iovec_len ; i++)
    VEC_SET(O_out,i,P_hold_in_val);

  return (SYS_OK);
}

/*
 *              Run Output Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to update the outputs and/or state
 *                   of the block.  It is called each iteration, for each
 *                   block instance during simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */

Ro_vec_sca_logic_regen_ranging (spb_parm, spb_input, spb_output,
spb_state)
STRUCT Pt_vec_sca_logic_regen_ranging *spb_parm;
STRUCT It_vec_sca_logic_regen_ranging *spb_input;
STRUCT Ot_vec_sca_logic_regen_ranging *spb_output;
STRUCT St_vec_sca_logic_regen_ranging *spb_state;
{
  int i, fcn;
```

```
  int in1, in2;
  double temp_out;

  fcn = vec_sca_logic_parse(P_logic_fcn);
  in2 = (int) I_in2;
  if (in2 <= 0) in2=0;

  for (i=0; i < O_out_iovec_len ; i++)
    {
      in1 = (int) VEC_GET(I_in1,i);
      if (in1 <= 0) in1=0;
      switch(fcn)
         {
         case -1: /* Doesn't match a programmed function */
           strcpy(wmsgbuf, "Error: bad logic_fcn parameter specified to
Vector Logic Block!");
           wmsgErrors(wmsgbuf);
           return(SYS_TERM);
         case 0:  /* OR */
           temp_out = (double) (in1 || in2);
           break;
         case 1:  /* AND */
           temp_out = (double) (in1 && in2);
           break;
         case 2:  /* NOT */
           temp_out = (double) (! in1);
           break;
         case 3:  /* XOR */
           temp_out = (double) vec_sca_logic_xor(in1,in2);
           break;
         case 4:  /* NOR */
           temp_out = (double) (! (in1 || in2));
           break;
         case 5:  /* NAND */
           temp_out = (double) (! (in1 && in2));
           break;
         case 6:  /* XNOR */
           temp_out = (double) (! vec_sca_logic_xor(in1,in2));
           break;
         }
      VEC_SET(O_out,i,temp_out);
    }
  return (SYS_OK);
}

/*
```

```
 *              Termination Function (must be present)
 *              --> If editing, modify only the lines within the
 *                  function's opening and closing brackets.
 *
 *              This function is used to dump the final state of the
 *                   block.  It is called once for each block instance
 *                   during the simulation.
 *
 *              Function must always return either SYS_OK, SYS_TERM,
 *                   or SYS_FATAL by using the return() function.
 *                   User may modify the line containing
"return(SYS_OK);".
 */
Te_vec_sca_logic_regen_ranging (spb_parm, spb_input, spb_output,
spb_state)
STRUCT Pt_vec_sca_logic_regen_ranging *spb_parm;
STRUCT It_vec_sca_logic_regen_ranging *spb_input;
STRUCT Ot_vec_sca_logic_regen_ranging *spb_output;
STRUCT St_vec_sca_logic_regen_ranging *spb_state;
{
         return (SYS_OK);
}
/*********************************************************************/
/*                                                                   */
/*      Add any additional functions you need here.                  */
/*                                                                   */
/*********************************************************************/

vec_sca_logic_parse(str)
char * str;
{
  int i;

  switch(*str)
    {
    case 'O':      /* OR */
      i = 0;
      break;
    case 'A':      /* AND */
      i = 1;
      break;
    case 'N':
      switch(*(str+1))
         {
         case 'O':
           switch(*(str+2))
```

```
            {
            case 'T':/* NOT */
              i=2;
              break;
            case 'R':/* NOR */
              i=4;
              break;
            default:
              i=-1;
            }
          break;
        case 'A':/* NAND */
          i = 5;
          break;
        default:
          i=-1;
        }
      break;
    case 'X':
      switch(*(str+1))
        {
        case 'O':/* XOR */
          i = 3;
          break;
        case 'N':/* XNOR */
          i = 6;
          break;
        default:
          i = -1;
        }
      break;
    default:
      i=-1;
    }

  return i;
}

vec_sca_logic_xor(a, b)
int a,b;
{
  int out;

  out = ((a && (!b)) || ((!a) && b));
  return(out);
}
```

vec_sca_logic.h

```
#include "FBCDEFS.h"

/*
 *
 *       Block Function: vec_sca_logic
 *       Library: regen_ranging
 *       Date: Mon Aug 29 16:10:38 1994
 *
 */

/*******************************************************************/
/*                                                                 */
/*       EDITABLE USER DEFINED STATE STRUCTURE                     */
/*                  --> STRUCT St_vec_sca_logic_regen_ranging
/*                                                                 */
/*******************************************************************/

/*
 *               State Structure (User Defined, editable)
 */
STRUCT St_vec_sca_logic_regen_ranging {
        int instance;
};

/*******************************************************************/
/*                                                                 */
/*       UNEDITABLE SIMULATOR DEFINED STRUCTURES                   */
/*                  --> STRUCT Pt_vec_sca_logic_regen_ranging
/*                  --> STRUCT It_vec_sca_logic_regen_ranging
/*                  --> STRUCT Ot_vec_sca_logic_regen_ranging
/*                                                                 */
/*******************************************************************/
/*
 *               Parameter Structure, Simulator Defined, uneditable
 */
STRUCT Pt_vec_sca_logic_regen_ranging {
        double hold_in_val;
        char * logic_fcn;
};

/*
 *               Input Structure, Simulator Defined, uneditable
 */
```

```
STRUCT It_vec_sca_logic_regen_ranging {
        long in1_iovec_len;
        double *in1;
        double *in2;
};

/*
 *               Output Structure, Simulator Defined, uneditable
 */
STRUCT Ot_vec_sca_logic_regen_ranging {
        long out_iovec_len;
        double *out;
};

/*****************************************************************/
/*                                                               */
/*       The following #defines may be used to shorten           */
/*       references to members of the above structures.          */
/*                                                               */
/*****************************************************************/
#define P_hold_in_val (spb_parm->hold_in_val)
#define P_logic_fcn (spb_parm->logic_fcn)
#define I_in1_iovec_len (spb_input->in1_iovec_len)
#define I_in1 (spb_input->in1)
#define I_in2 (*spb_input->in2)
#define O_out_iovec_len (spb_output->out_iovec_len)
#define O_out (spb_output->out)
```

Block Name: xnor

Synopsis:

Input Signals:

hold: A control signal allowing the output to be held.
in1: A scalar signal.
in2: A scalar signal.

Output Signals:

out: A scalar result.

Parameters:

none

Functional Description:

This block performs the boolean function **in1** XNOR **in2**. It places the result into **out**.

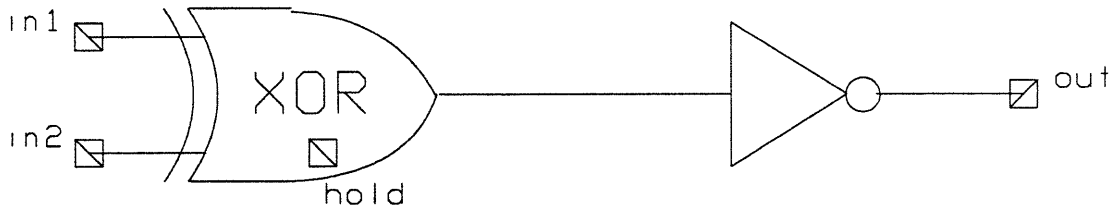

Figure B.43: xnor block diagram.

Appendix C

Systems used to Test Comdisco Designs

This appendix contains descriptions of the systems used to test Comdisco blocks. The descriptions are formatted in the style of data sheets, with a listing of the inputs, outputs, and method of use for each system. Each description also contains a figure showing the block diagram of each system.

Block Name: **2comp_rec_test**

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

accum_per: Number of periods over which to perform the correlation.
clk_freq: The PN clock component frequency.
code_int: An integer representation of the binary PN component code.
noise_var: The variance of the white Gaussian noise added to the received PN sequence.
num_bits: The length of the PN component (in bits).
s_freq: The sampling frequency.

Functional Description:

This block allows the entire component recovery algorithm to be tested on a PN sequence consisting of only two components, plus clock.

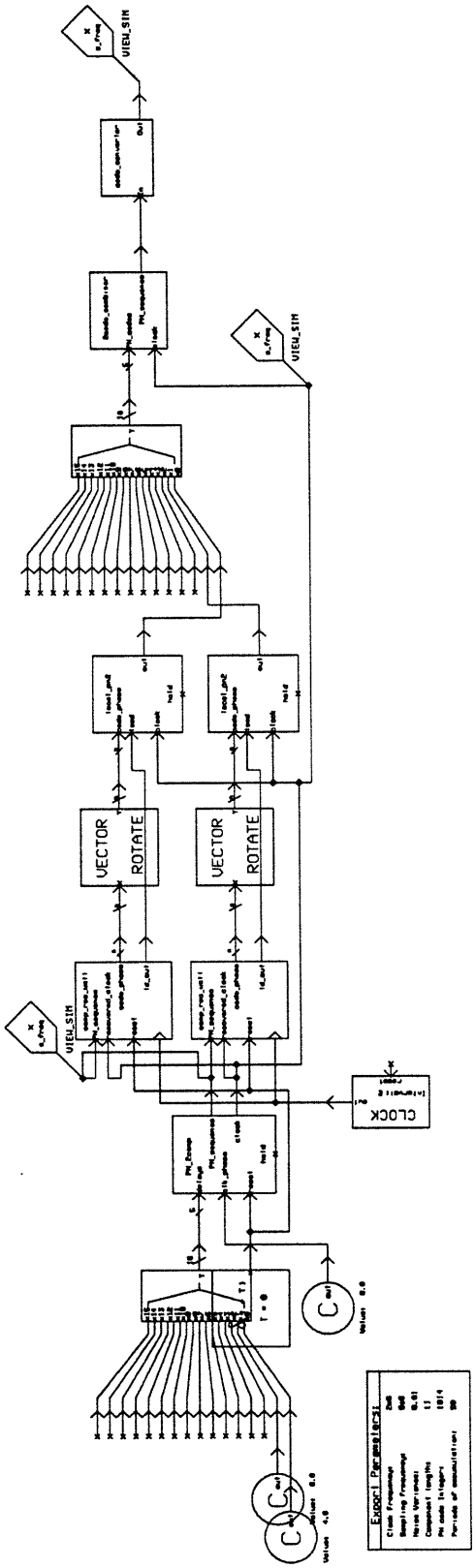

Figure C.1: 2comp_rec_test block diagram.

Block Name: acq_test

Synopsis:

Input Signals:

PN_code: PN sequence from receiver.

Output Signals:

regen_clock: Regenerated clock component of PN code.

Parameters:

c_freq: The receiver carrier frequency.
clk_freq: The PN clock component frequency.
delta_t: The number of simulation iterations in 1/8 period of the PN clock component.
div_ratio, n, m: The ratio between the clock frequency and the carrier frequency.
noise_variance: The variance of the white Gaussian noise added to the received PN sequence.
win_length: The number of simulation iterations in 1/4 period of the PN clock.

Functional Description:

These blocks (.detail and .system) were used to test the **clock_acq** block and its algorithms.

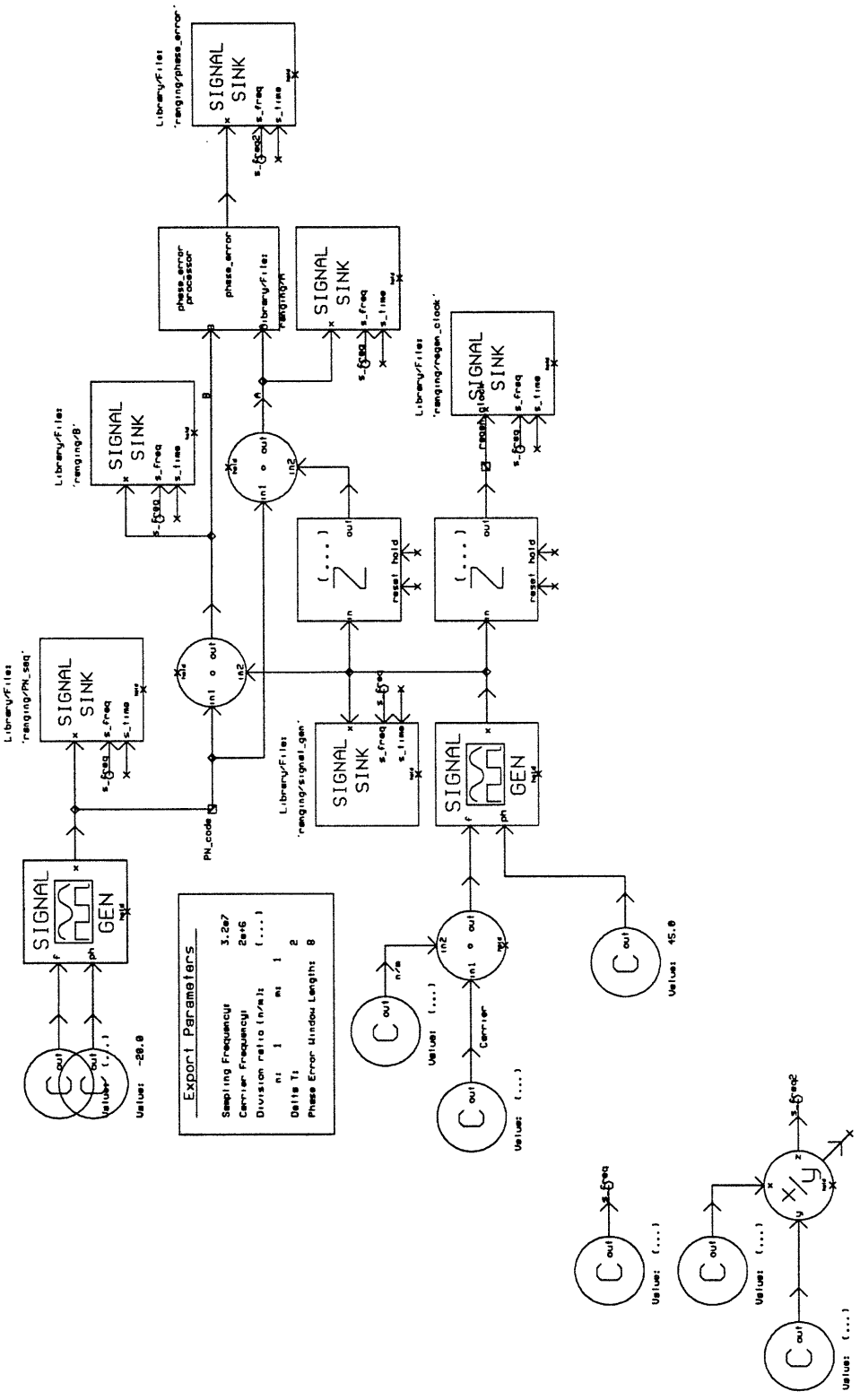

Figure C.2: acq_test.detail block diagram.

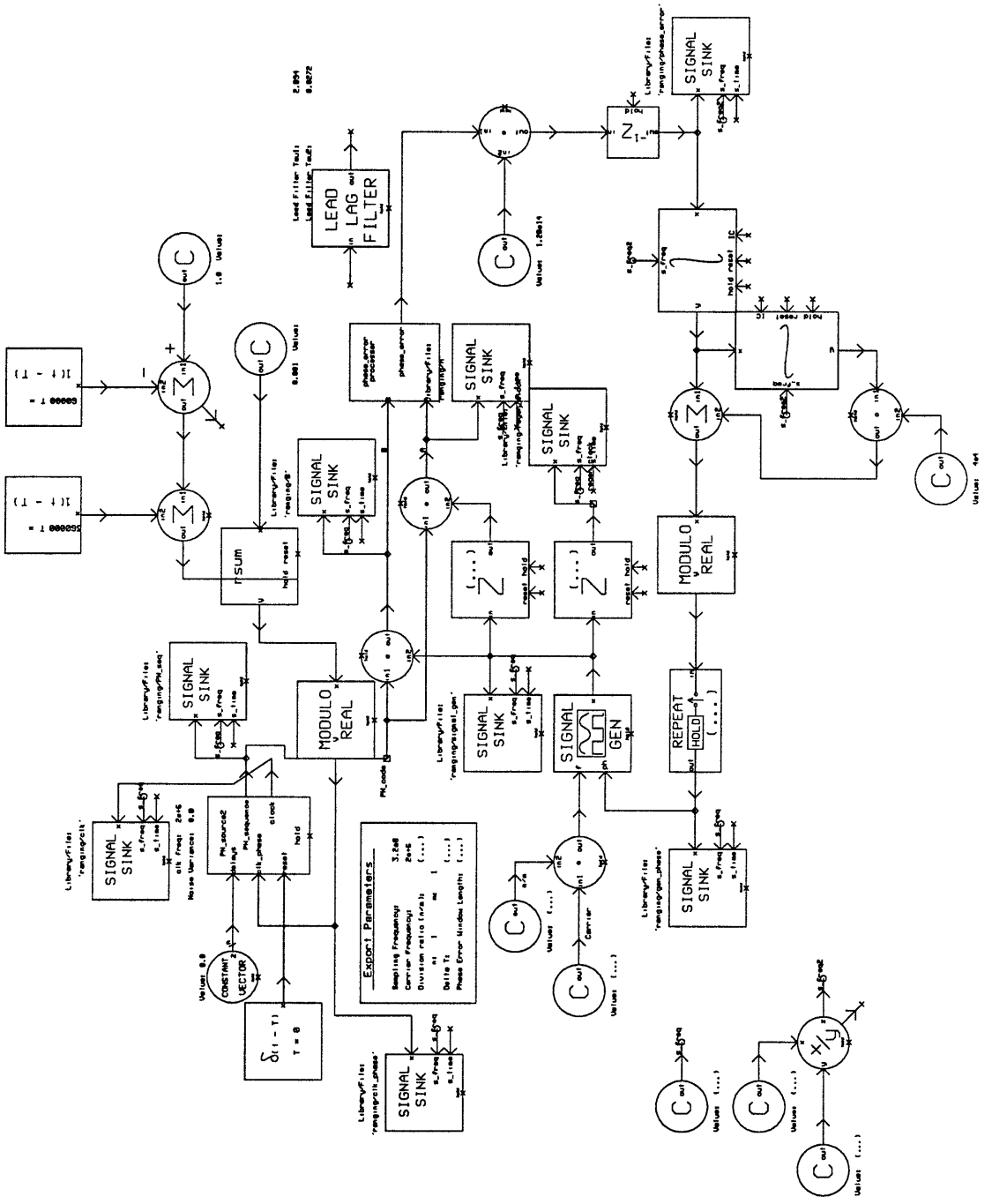

Figure C.3: acq_test.system block diagram.

Block Name: cmp_cnt_test

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

none

Functional Description:

This is a test block that allows the **PN_comp_count** block to be verified.

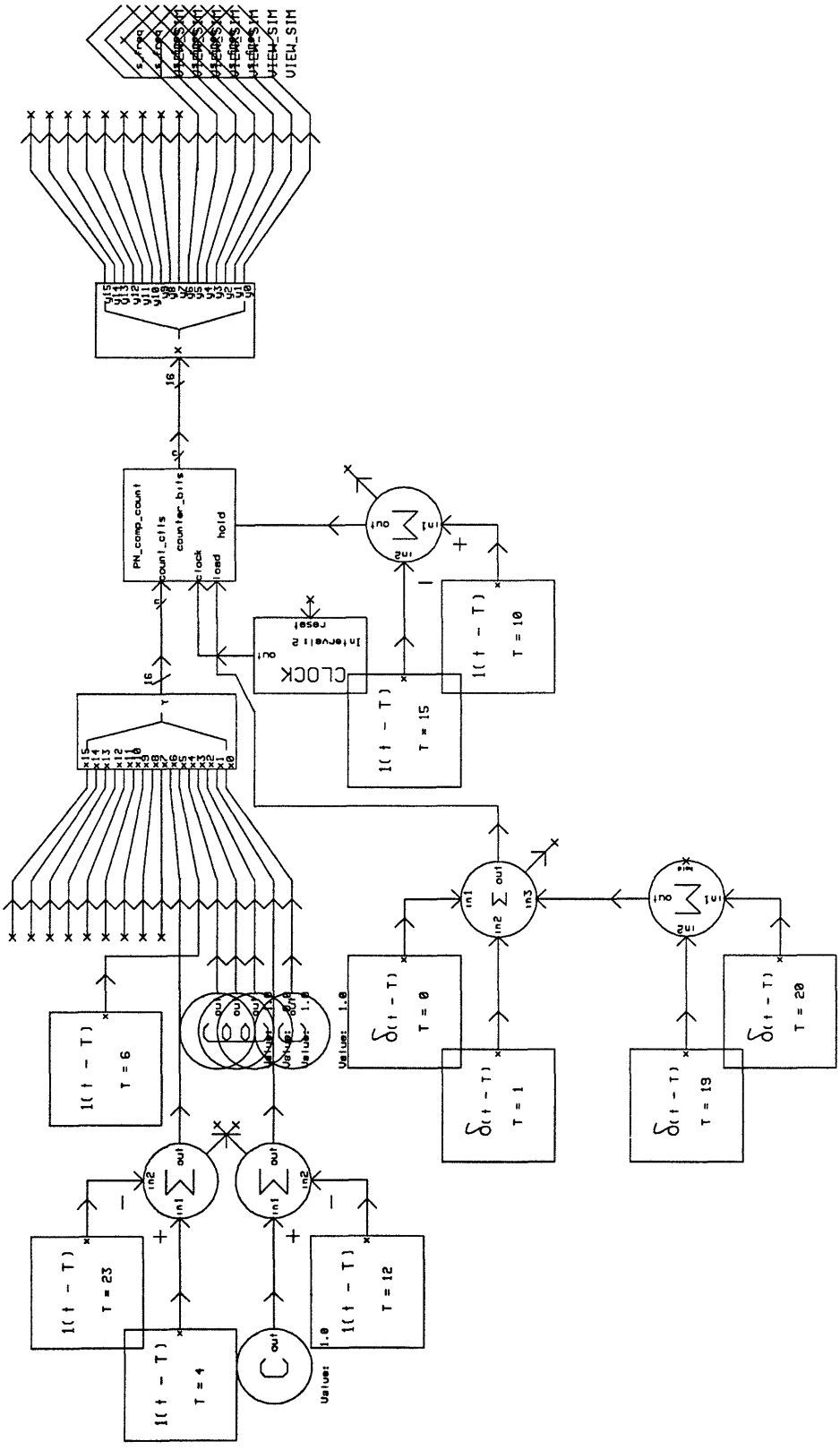

Figure C.4: cmp_cnt_test block diagram.

Block Name: **combiner_test**

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

none

Functional Description:

This is a test block that allows the **code_combiner** block to be verified.

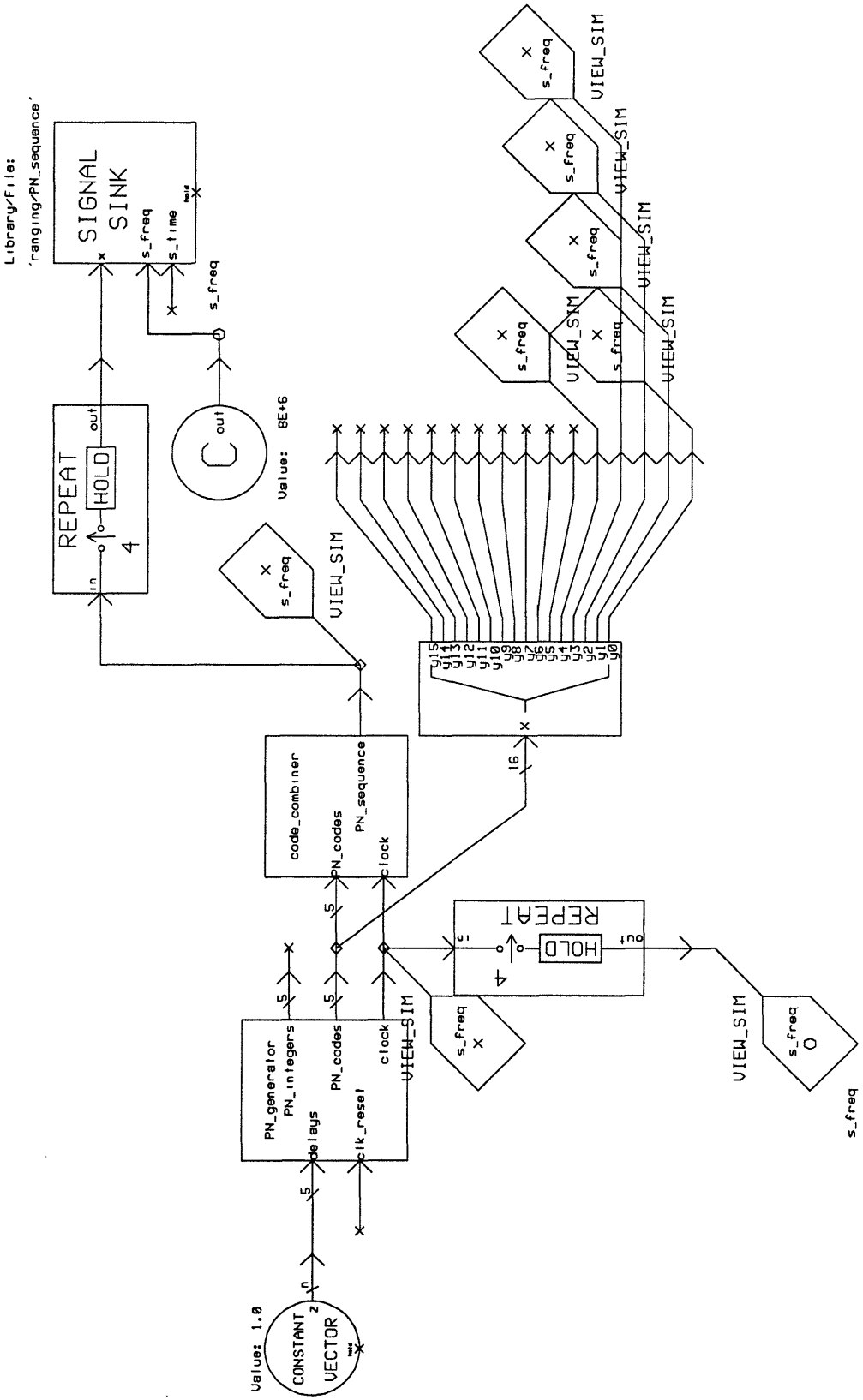

Figure C.5: combiner_test block diagram.

Block Name: comp_rec_test

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

clk_freq: The PN clock component frequency.
code_int: An integer representation of the binary code for the PN component being acquired.
noise_var: The variance of the white Gaussian noise added to the received PN sequence.
num_bits: The length (in bits) of the PN component being acquired.
s_freq: The sampling frequency.

Functional Description:

This is a test block that allows the **comp_recover** block to be verified. It attempts to acquire a PN code component.

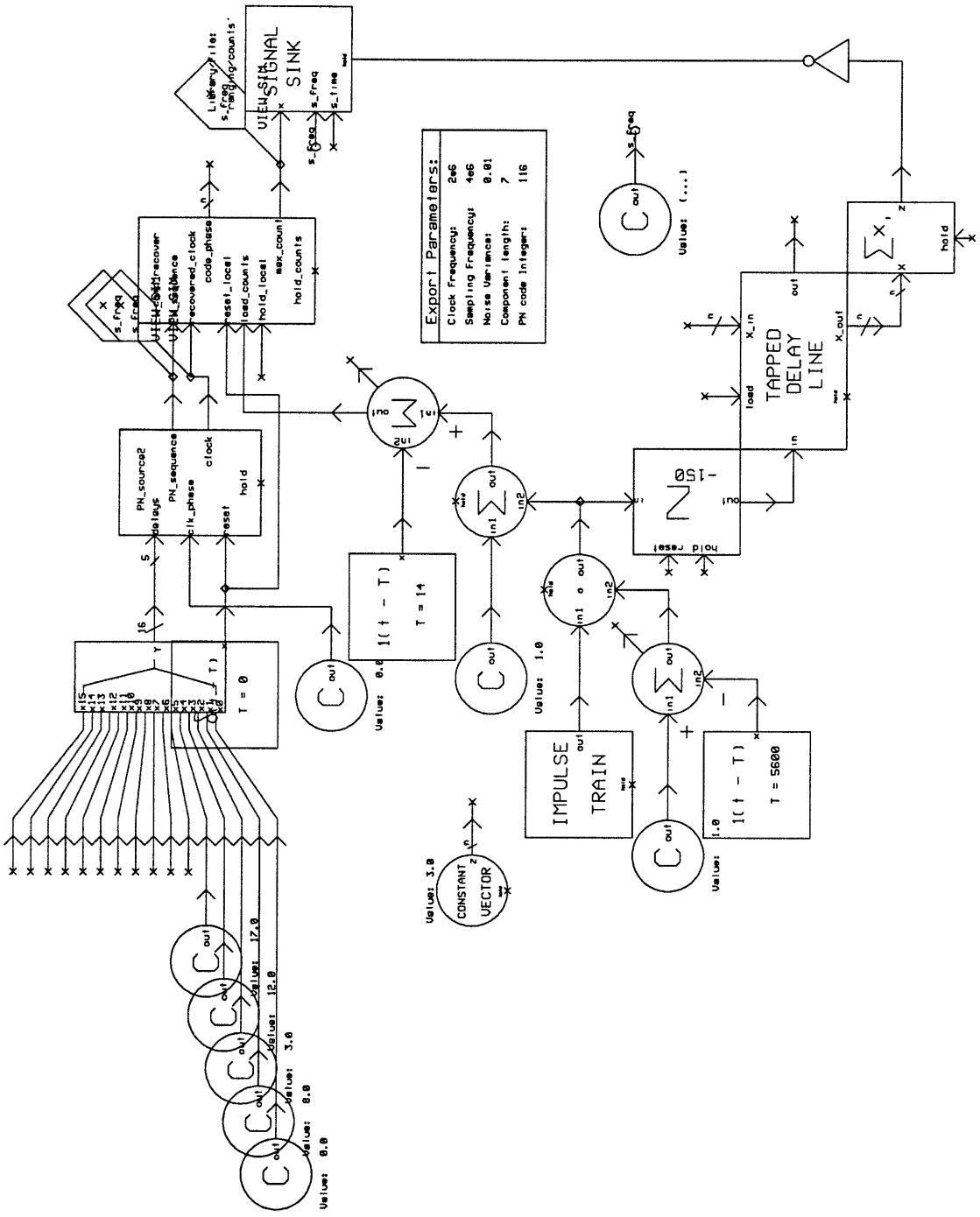

Figure C.6: comp_rec_test block diagram.

Block Name: comp_rec_test2

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

accum_per: Number of periods over which to perform the correlation.
clk_freq: The PN clock component frequency.
code_int: An integer representation of the binary PN component code.
noise_var: The variance of the white Gaussian noise added to the received PN sequence.
num_bits: The length of the PN component (in bits).
s_freq: The sampling frequency.

Functional Description:

This is a test block that allows the **comp_rec_wctl** block to be verified.

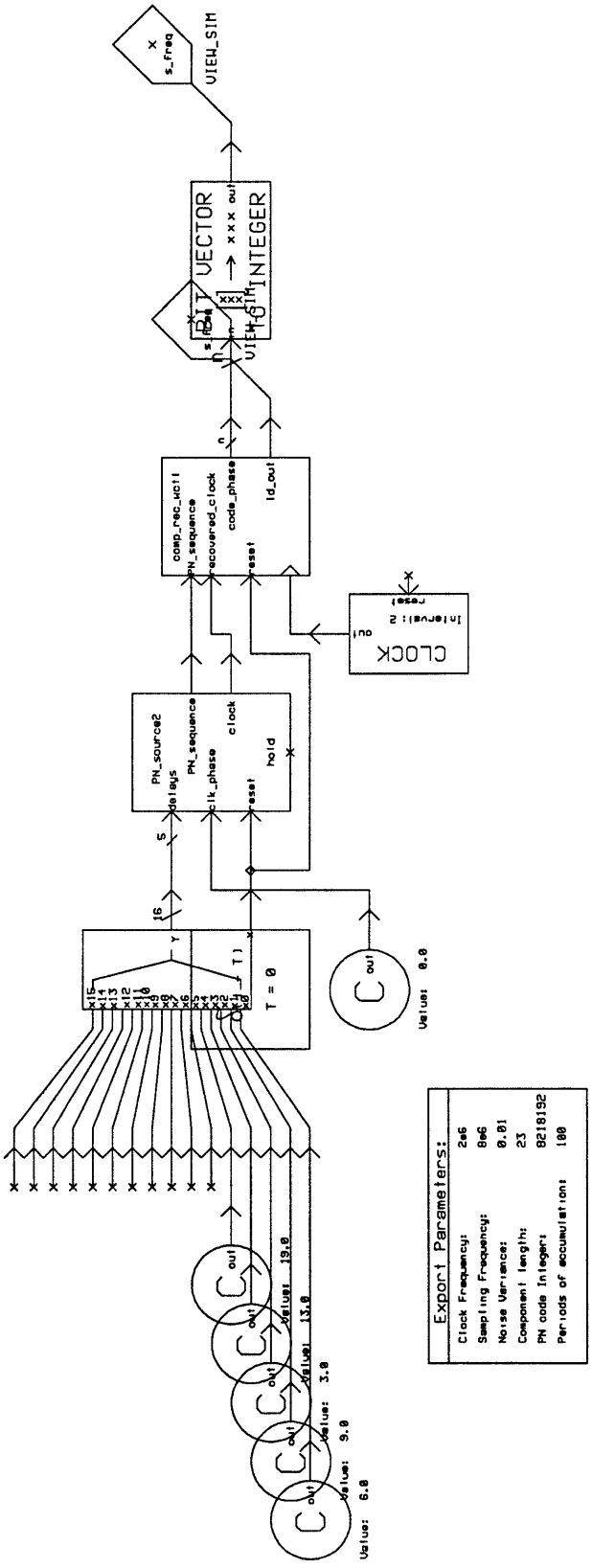

Figure C.7: comp_rec_test2 block diagram.

Block Name: **comp_rec_test3**

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

accum_per: The number of periods over which to perform the correlation.
clk_freq: The frequency of the PN clock component.
c*X*_int: An integer representation of the binary code for PN component X.
c*X*_len: The length (in bits) of PN component X.
noise_var: The variance of the white Gaussian noise added to the received PN sequence.
s_freq: The sampling frequency.
shift_len: The number of samples to rotate the comp_phase vectors.

Functional Description:

This is a test block that allows the **comp_rec_all** and **pn_regen** blocks to be verified.

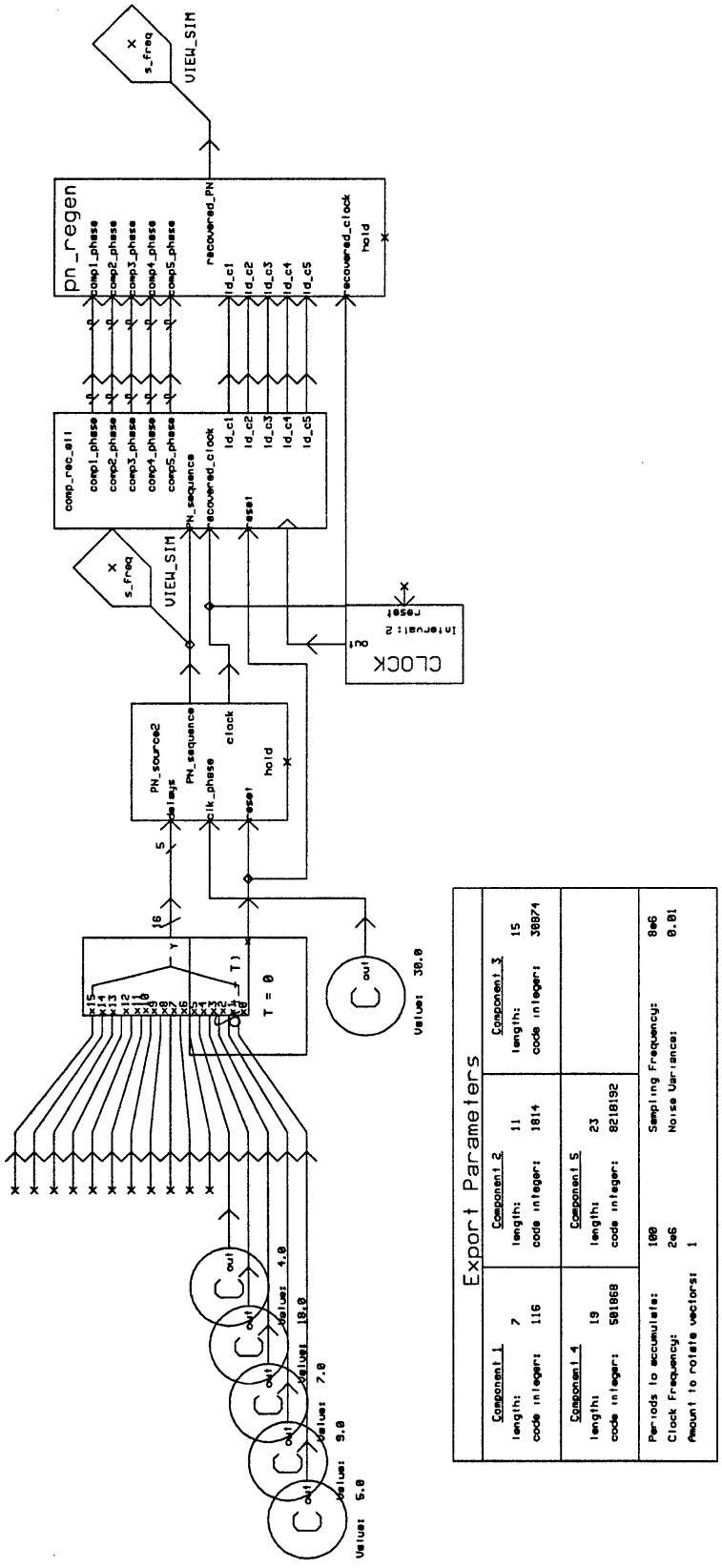

Figure C.8: comp_rec_test3 block diagram.

Block Name: **count_test**

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

none

Functional Description:

This is a test block that allows the **control_cnt** block to be verified.

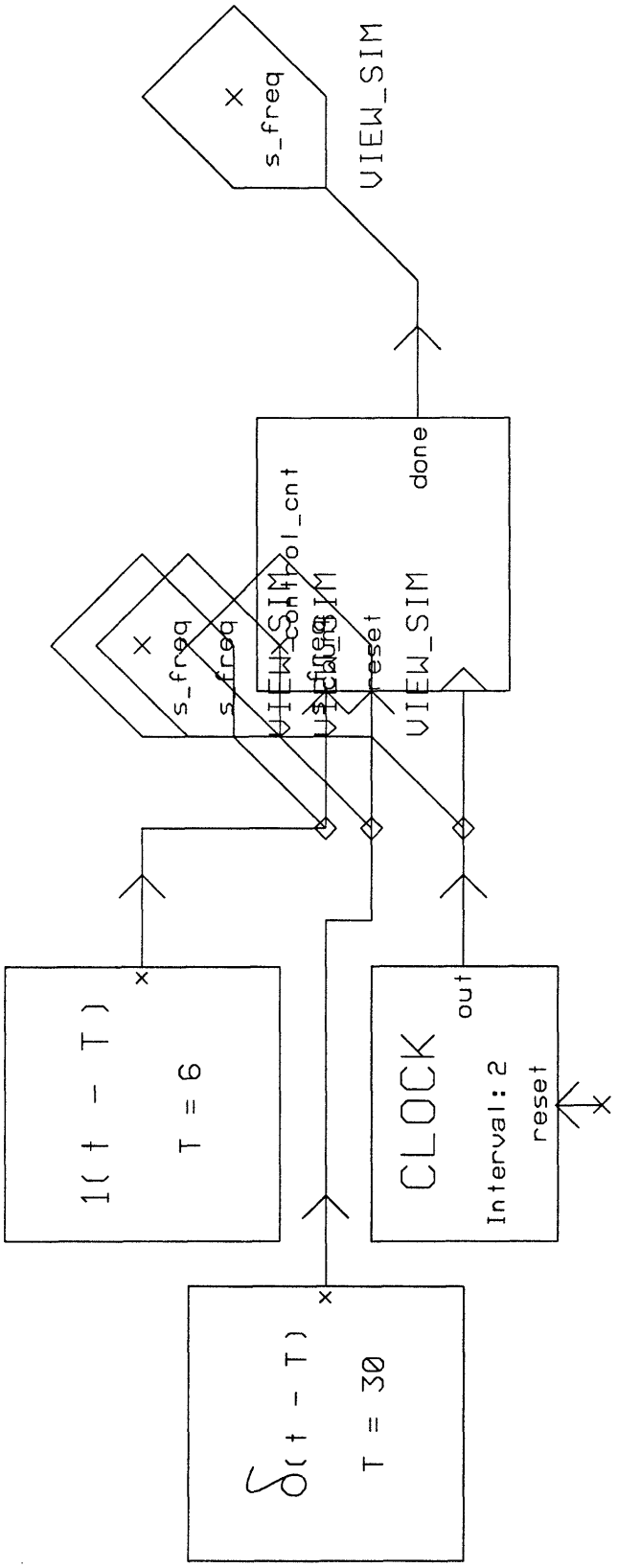

Figure C.9: count_test block diagram.

Block Name: fsm_test

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

none

Functional Description:

This is a test block that allows the **control_fsm** block to be verified.

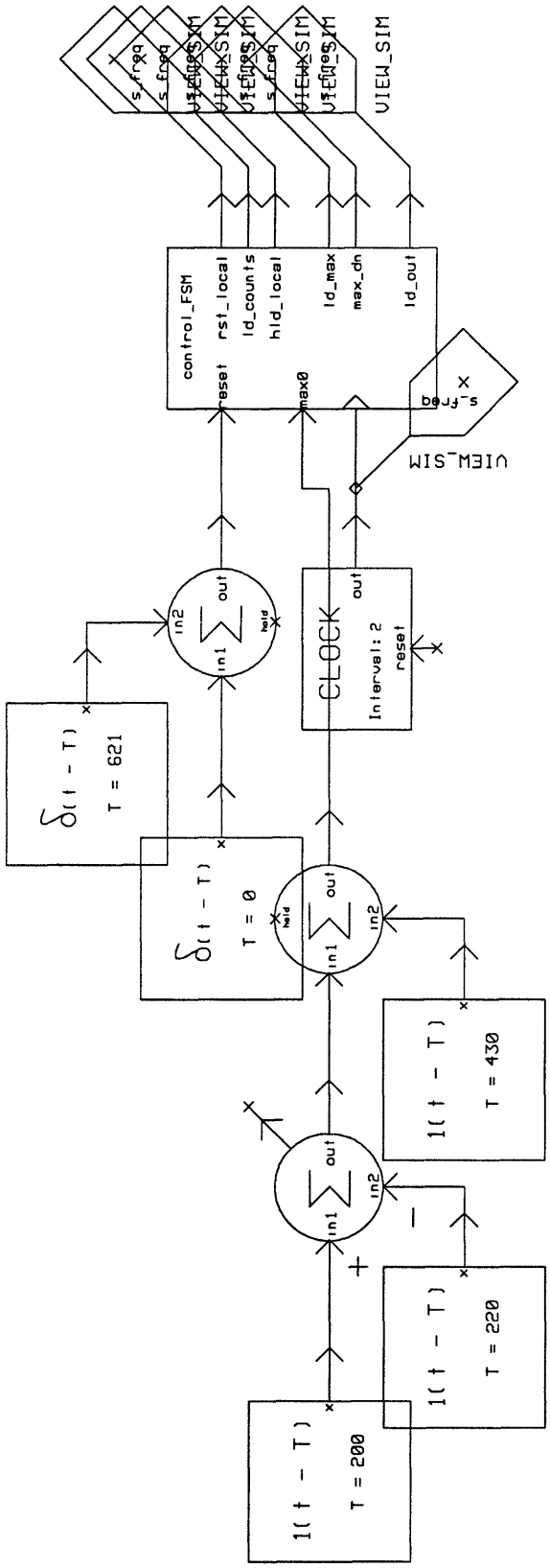

Figure C.10: fsm_test block diagram.

Block Name: int2vec_test

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

high_low: The order in which the binary representation fills the bit vector (MSB or LSB to the low component).

num_bits: The number of bits used in the binary representation.

Functional Description:

This is a test block that allows the **int2vec** and the **vec2int** blocks to be verified.

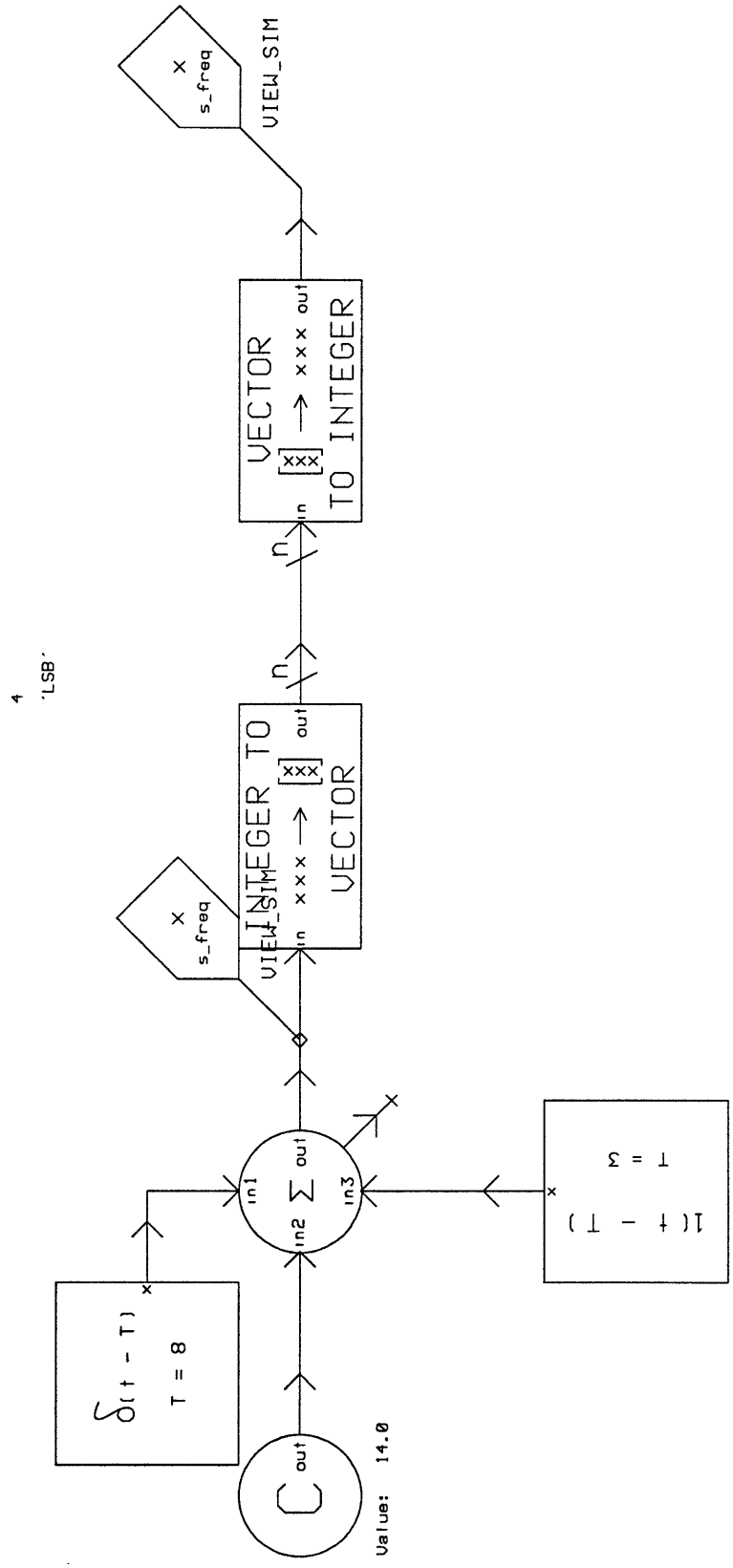

Figure C.11: int2vec_test block diagram.

Block Name: **local_pn_test**

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

none

Functional Description:

This is a test block that allows the **local_pn** block to be verified.

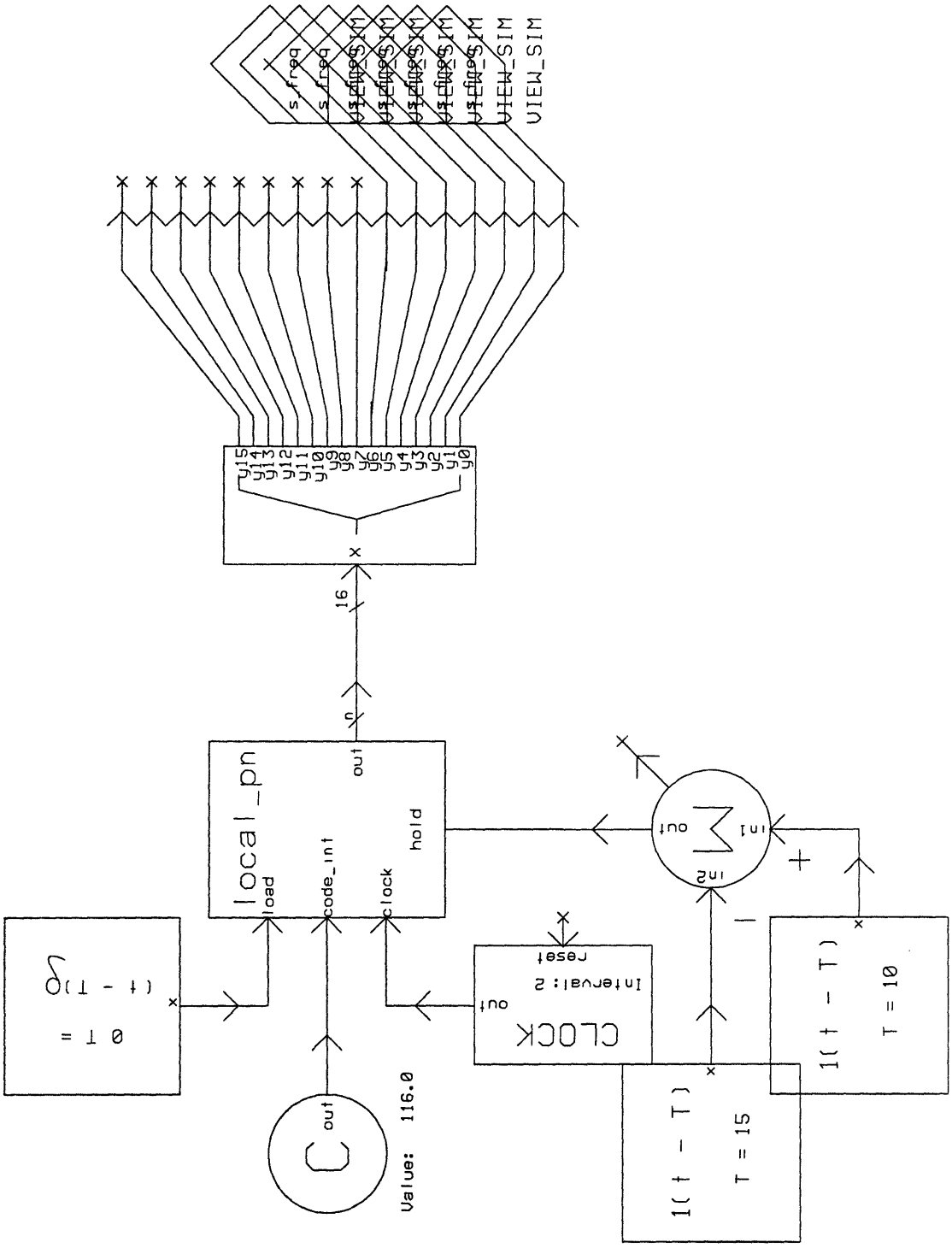

Figure C.12: local_pn_test block diagram.

Block Name: **pn_gen_test**

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

none

Functional Description:

This is a test block that allows the **pn_generator** and **pn_gen2** blocks to be verified.

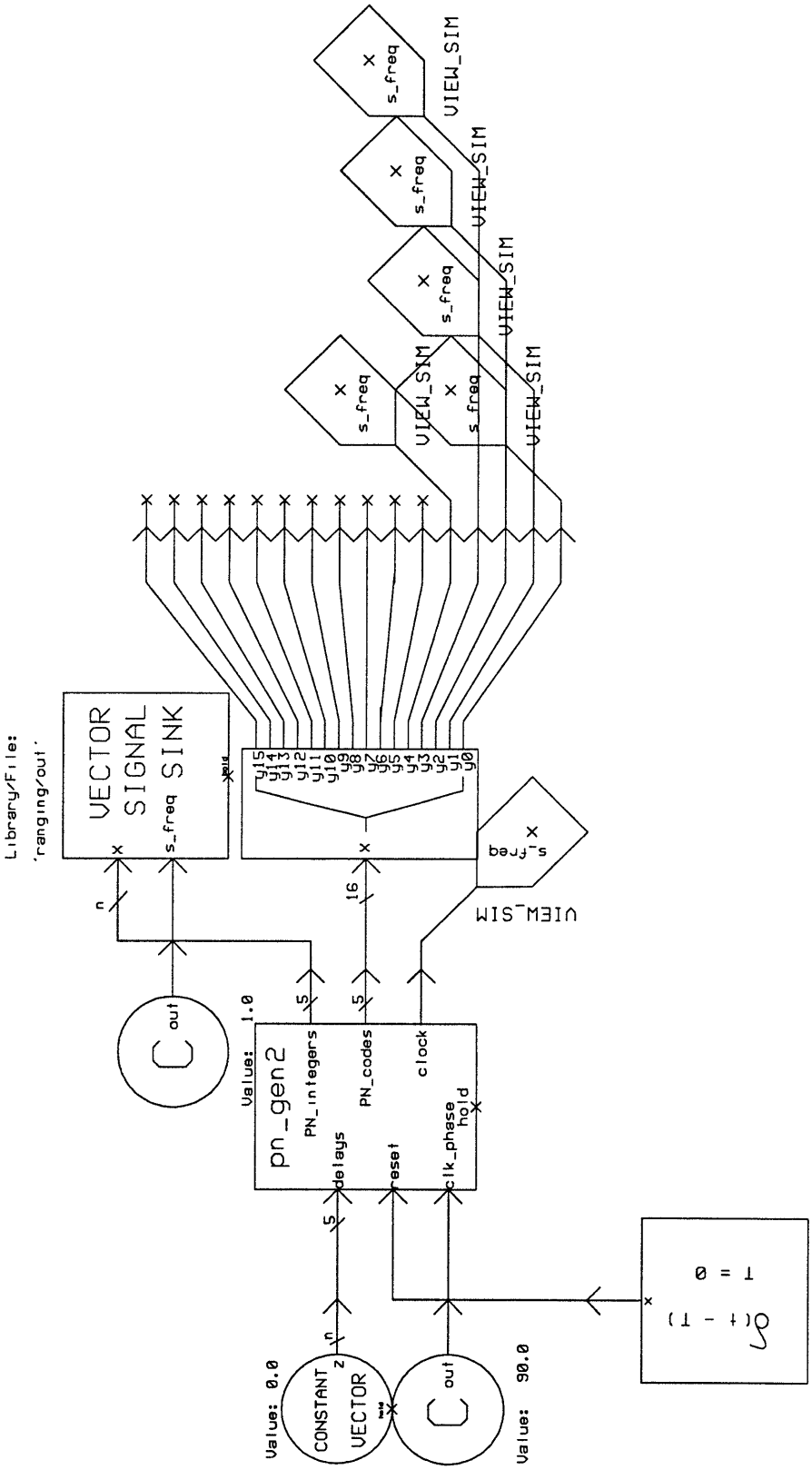

Figure C.13: pn_gen_test block diagram.

Block Name: **ranging_test**

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

accum_per: The number of periods over which to perform the correlation.
clk_freq: The frequency of the PN clock component.
cX_dly: The delay applied to component X of the received PN code.
cX_int: An integer representation of the binary code of PN component X.
cX_len: The length of PN component X (in bits).
div_ratio, n, m: The ratio of the clock frequency to the receiver carrier frequency.
noise_var: The variance of the noise added to the received PN sequence.
s_freq: The sampling frequency.

Functional Description:

This is a test block that allows the **regen_ranging** block to be verified.

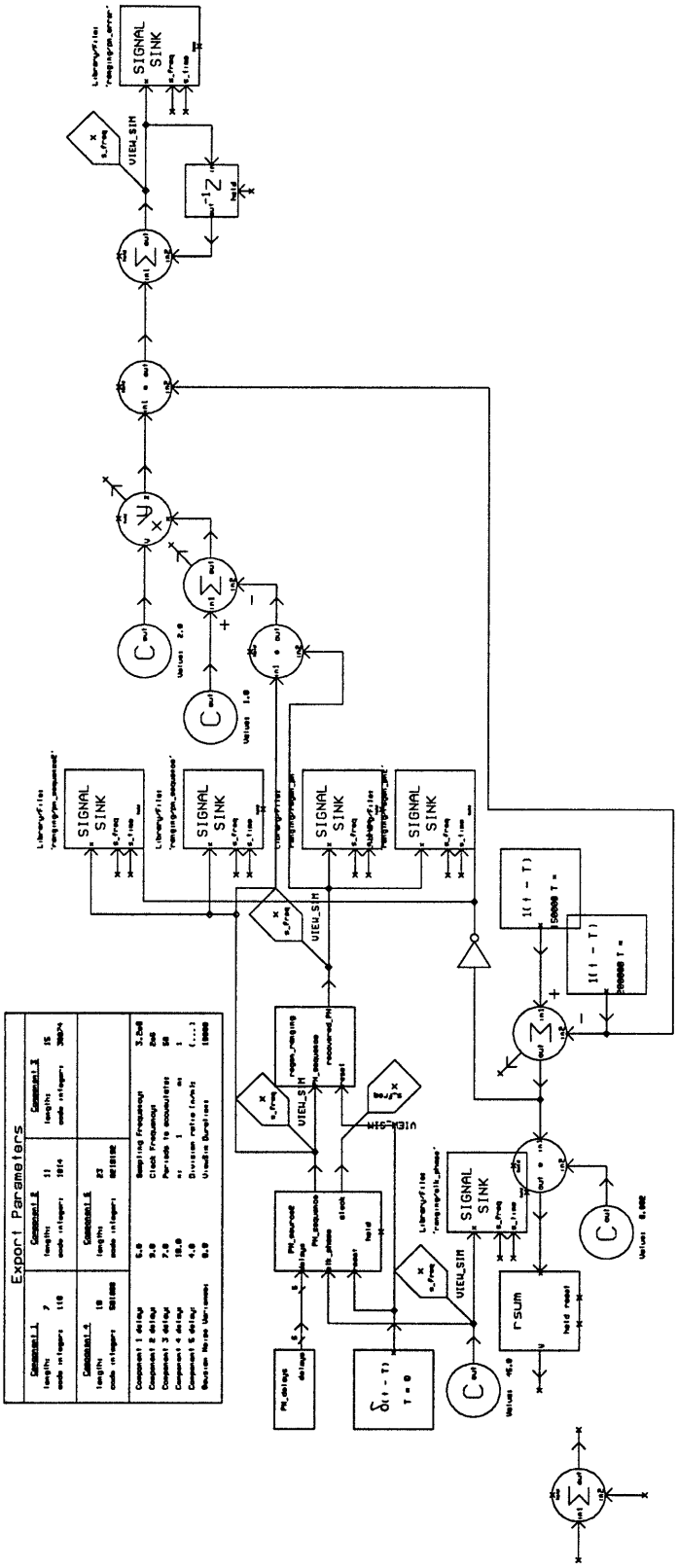

Figure C.14: ranging_test block diagram.

Block Name: single_test

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

none

Functional Description:

This is a test block that allows the **single_pn** and **single_pn2** blocks to be verified.

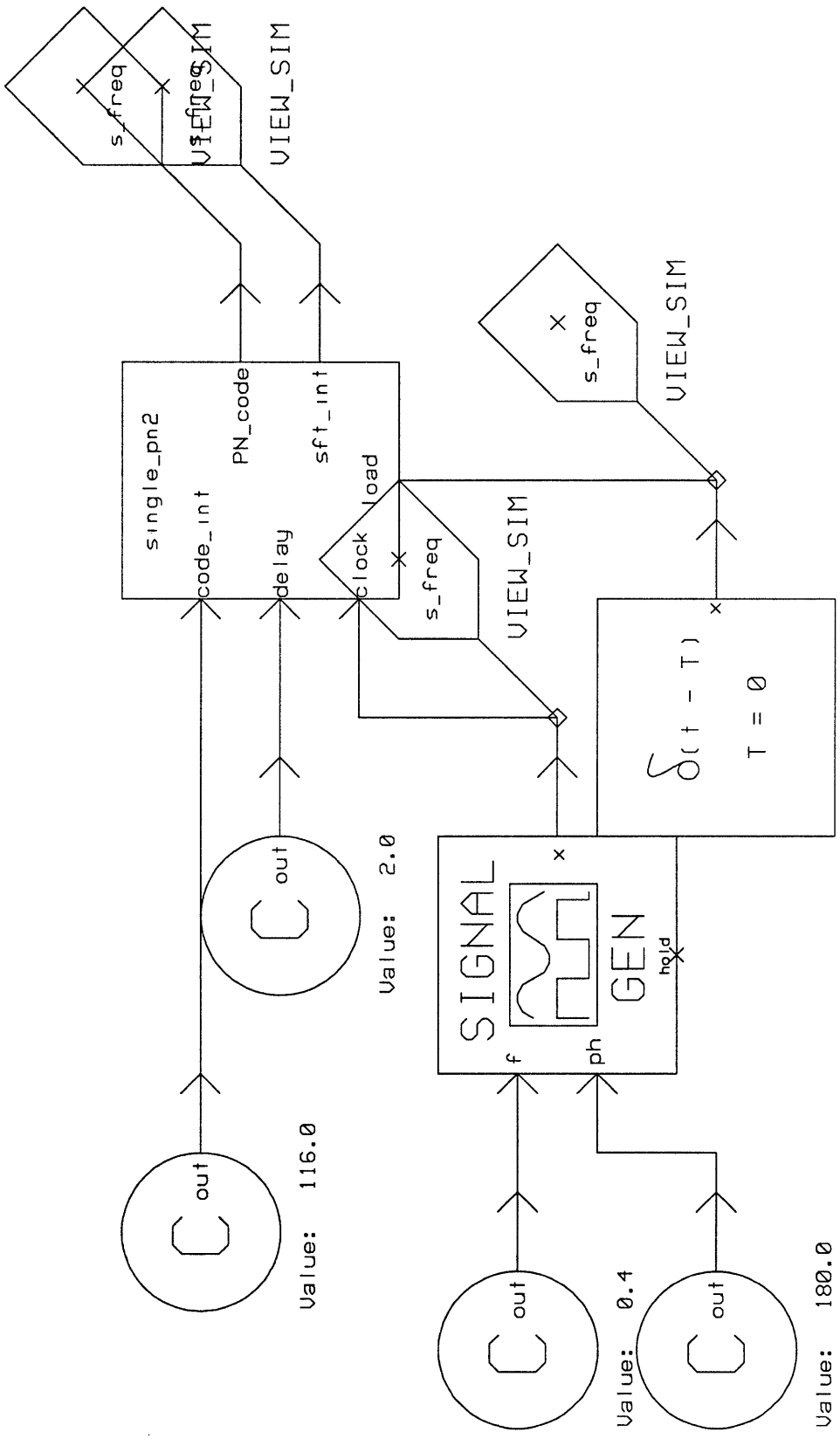

Figure C.15: single_test block diagram.

Block Name: vec_logic_test

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

logic_fcn: The logic function to be performed (OR, AND, NOT, XOR, NOR, NAND, or XNOR).

Functional Description:

This is a test block that allows the **vec_sca_logic** and **vec_logic** blocks to be verified.

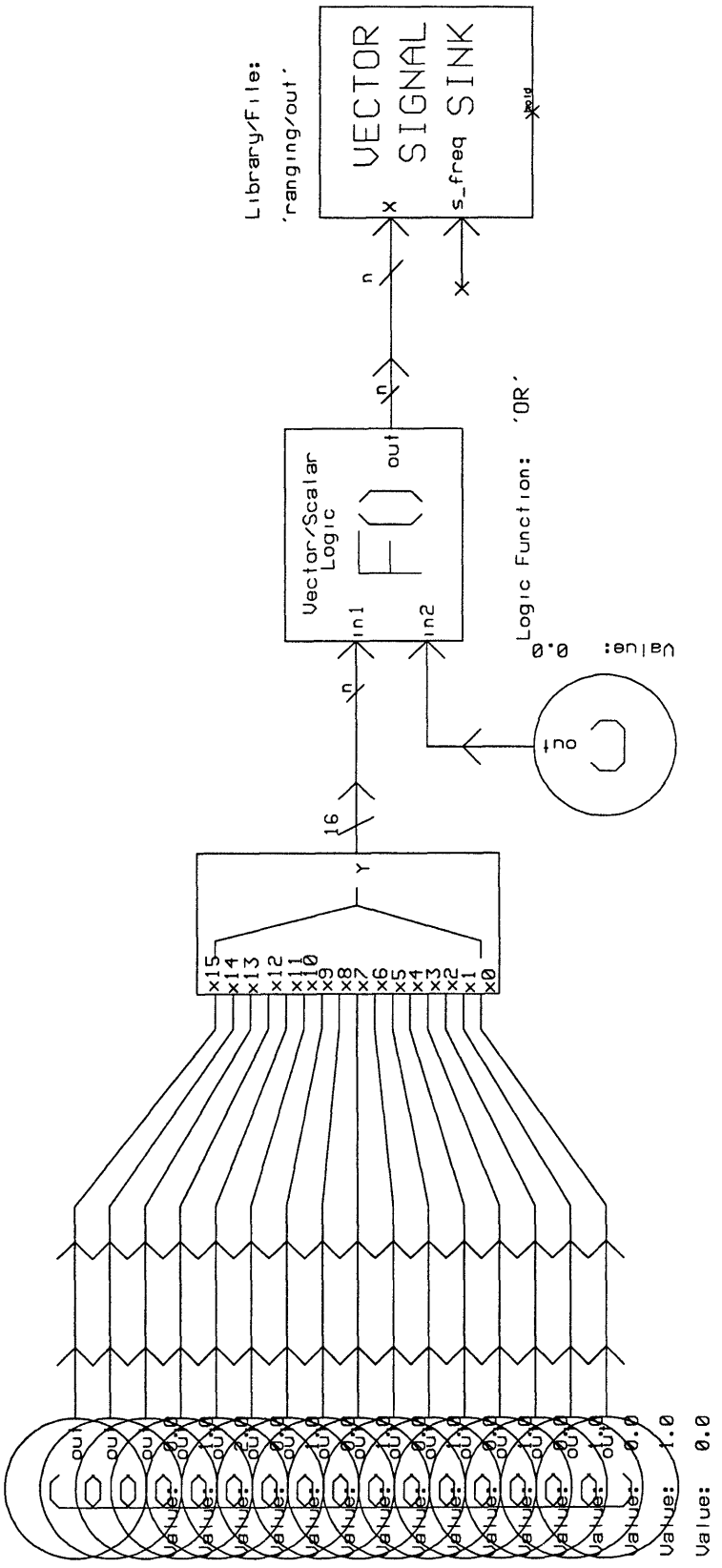

Figure C.16: vec_logic_test block diagram.

Block Name: vec_rot_test

Synopsis:

Input Signals:

none

Output Signals:

none

Parameters:

none

Functional Description:

This is a test block that allows the **vec_rotate** block to be verified.

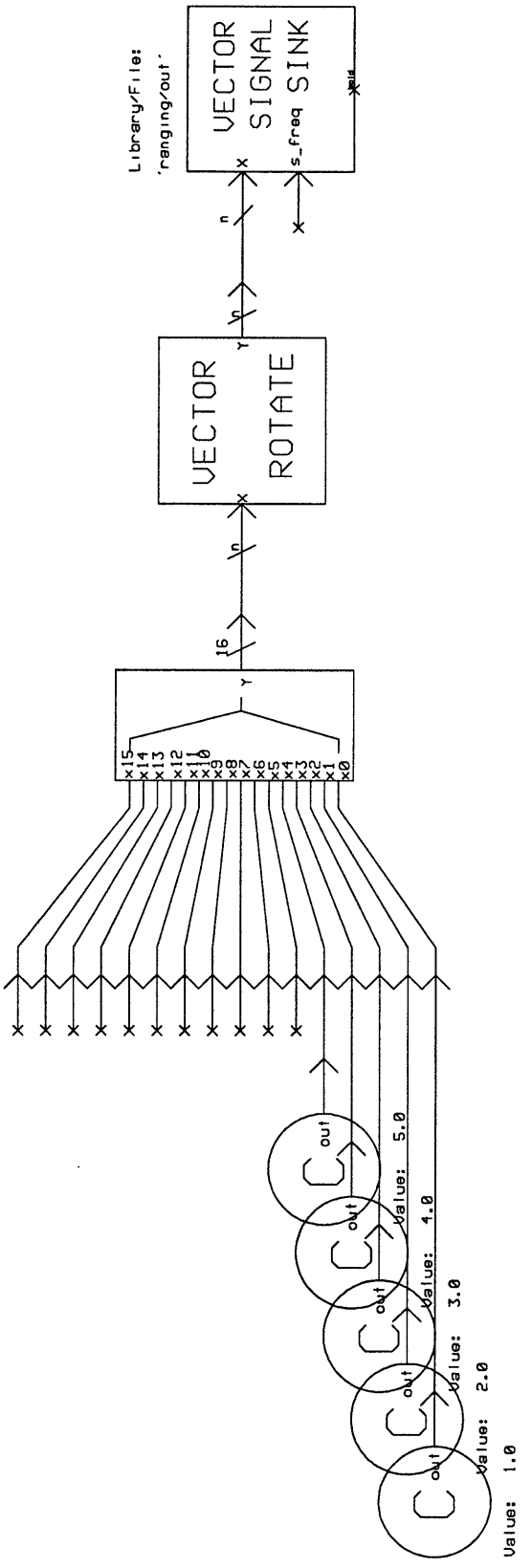

Figure C.17: vec_rot_test block diagram.

Appendix D

FSM description files

D.1 FSM State Diagram

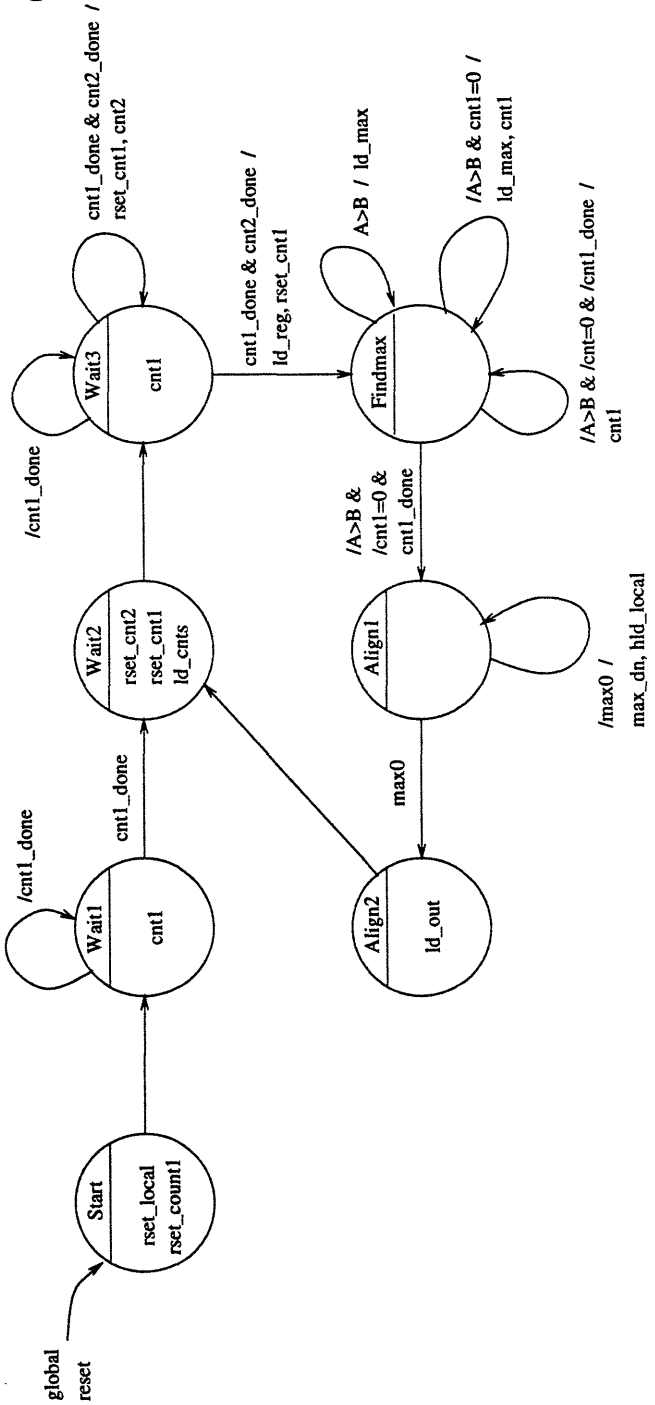

Figure D.1: State Diagram of Control FSM.

D.2 Fsmc input file

```
start:    if (RESET) then goto start;
          RST_LOCAL, RST_COUNT1;
          goto wait1;

wait1:    if (RESET) then goto start;
          COUNT1;
          if (COUNT1_DONE) then goto wait2;
          else stay;

wait2:    if (RESET) then goto start;
          RST_COUNT2, RST_COUNT1, LD_COUNTS;
          goto wait3;

wait3:    if (RESET) then goto start;
          COUNT1;
          if (COUNT1_DONE) then
            {
              RST_COUNT1;
              if (COUNT2_DONE) then
                {
                   LD_REG;
                   goto findmax;
                }
              else
                {
                   COUNT2;
                   stay;
                }
            }
          else stay;

findmax:  if (RESET) then goto start;
          if (A_GT_B) then
            {
              LD_MAX;
              stay;
            }
          else if (COUNT1_ZERO) then
            {
              LD_MAX, COUNT1;
              stay;
            }
          else
            {
              if (COUNT1_DONE) then goto align1;
```

```
            else
              {
                 COUNT1;
                 stay;
              }
          }

align1:  if (RESET) then goto start;
         if (MAX0) then
           {
             LD_OUT;
             goto align2;
           }
         else
           {
             MAX_DN, HLD_LOCAL;
             stay;
           }

align2:  if (RESET) then goto start;
         LD_OUT;
         goto wait2;

state8:  goto start;
```

D.3 FSM state file

```
State: start (0)

   RST_COUNT1 = /RESET
   RST_LOCAL =  /RESET

   Goto  start (0) on:
                RESET
   Goto  wait1 (1) on:
                /RESET

State: wait1 (1)

   COUNT1 =     /RESET

   Goto  start (0) on:
                RESET
   Goto  wait2 (2) on:
                /RESET * COUNT1_DONE
   Goto  wait1 (1) on:
                /RESET * /COUNT1_DONE
```

```
State: wait2 (2)

   LD_COUNTS =  /RESET
   RST_COUNT1 = /RESET
   RST_COUNT2 = /RESET

   Goto  start (0) on:
                RESET
   Goto  wait3 (3) on:
                /RESET

State: wait3 (3)

   COUNT2 =      /RESET * COUNT1_DONE * /COUNT2_DONE
   LD_REG =      /RESET * COUNT1_DONE * COUNT2_DONE
   RST_COUNT1 = /RESET * COUNT1_DONE
   COUNT1 =      /RESET

   Goto  start (0) on:
                RESET
   Goto  findmax (4) on:
                /RESET * COUNT1_DONE * COUNT2_DONE
   Goto  wait3 (3) on:
                /RESET * COUNT1_DONE * /COUNT2_DONE
   Goto  wait3 (3) on:
                /RESET * /COUNT1_DONE

State: findmax (4)

   LD_MAX =      /RESET * COUNT1_ZERO +
                /RESET * A_GT_B
   COUNT1 =      /RESET * /COUNT1_DONE * /A_GT_B +
                /RESET * /A_GT_B * COUNT1_ZERO

   Goto  start (0) on:
                RESET
   Goto  align1 (5) on:
                /RESET * COUNT1_DONE * /A_GT_B * /COUNT1_ZERO
   Goto  findmax (4) on:
                /RESET * /COUNT1_DONE * /A_GT_B * /COUNT1_ZERO
   Goto  findmax (4) on:
                /RESET * /A_GT_B * COUNT1_ZERO
   Goto  findmax (4) on:
                /RESET * A_GT_B
```

```
State: align1 (5)

   LD_OUT =      /RESET * MAX0
   MAX_DN =      /RESET * /MAX0
   HLD_LOCAL =   /RESET * /MAX0

   Goto  start (0) on:
                 RESET
   Goto  align1 (5) on:
                 /RESET * /MAX0
   Goto  align2 (6) on:
                 /RESET * MAX0

State: align2 (6)

   LD_OUT =      /RESET

   Goto  start (0) on:
                 RESET
   Goto  wait2 (2) on:
                 /RESET

State: state8 (7)

   Goto  start (0) on:
                 1

<<<Hidden States>>>
```

D.4 FSM equation file

```
register Q0;
register Q1;
register Q2;

Q0 =          /RESET * COUNT1_DONE * /A_GT_B * /COUNT1_ZERO * /Q0 * /Q1 +
              /RESET * /COUNT1_DONE * /Q2 +
              /RESET * /Q0 * /Q2 +
              /RESET * /COUNT2_DONE * Q1 * /Q2 +
              /RESET * /MAX0 * Q0 * /Q1 * Q2;

Q1 =          /RESET * MAX0 * Q0 * /Q1 * Q2 +
              /RESET * /COUNT1_DONE * Q1 * /Q2 +
              /RESET * /COUNT2_DONE * Q1 * /Q2 +
              /RESET * COUNT1_DONE * Q0 * /Q1 * /Q2 +
              /RESET * /Q0 * Q1;

Q2 =          /RESET * COUNT1_DONE * COUNT2_DONE * Q0 * Q1 * /Q2 +
              /RESET * /Q1 * Q2;

RST_LOCAL =   /RESET * /Q0 * /Q1 * /Q2;

RST_COUNT1 =  /RESET * COUNT1_DONE * Q1 * /Q2 +
              /RESET * /Q0 * /Q2;

COUNT1 =      /RESET * /A_GT_B * COUNT1_ZERO * /Q0 * /Q1 * Q2 +
              /RESET * Q0 * /Q2 +
              /RESET * /COUNT1_DONE * /A_GT_B * /Q0 * /Q1 * Q2;

RST_COUNT2 =  /RESET * /Q0 * Q1 * /Q2;

LD_COUNTS =   /RESET * /Q0 * Q1 * /Q2;

LD_REG =      /RESET * COUNT1_DONE * COUNT2_DONE * Q0 * Q1 * /Q2;

COUNT2 =      /RESET * COUNT1_DONE * /COUNT2_DONE * Q0 * Q1 * /Q2;

LD_MAX =      /RESET * COUNT1_ZERO * /Q0 * /Q1 * Q2 +
              /RESET * A_GT_B * /Q0 * /Q1 * Q2;

LD_OUT =      /RESET * MAX0 * Q0 * /Q1 * Q2 +
              /RESET * /Q0 * Q1 * Q2;

MAX_DN =      /RESET * /MAX0 * Q0 * /Q1 * Q2;

HLD_LOCAL =   /RESET * /MAX0 * Q0 * /Q1 * Q2;
```

Appendix E

Mentor Graphics Schematics

This appendix contains descriptions of each Mentor Graphics schematic. The descriptions are formatted in the style of data sheets, with a listing of the inputs, outputs, and method of use for each schematic. Each description also contains a figure showing the underlying logic of each schematic.

Block Name: *X*cnt (*X* is 7,11,15,19,23)

Synopsis:

Input Signals:

Clk: The clock signal.
Cnt: The count enable signal.
Rst_lo: The counter reset signal (negative assertion).

Output Signals:

Count<*n*:0>: The value stored in the counter (**n** is 2,3,4).
Done_lo: The carry signal (asserted when counter is about to roll back to zero).
Zero: A signal indicating when the counter holds a value of zero.

Parameters:

none

Functional Description:

This block counts from zero to **X**-1, then rolls over back to zero. It increments the counter once every other clock cycle, when **Cnt** is asserted. The reset is synchronous, and the rollover is synchronous and waits for **Cnt** to be asserted.

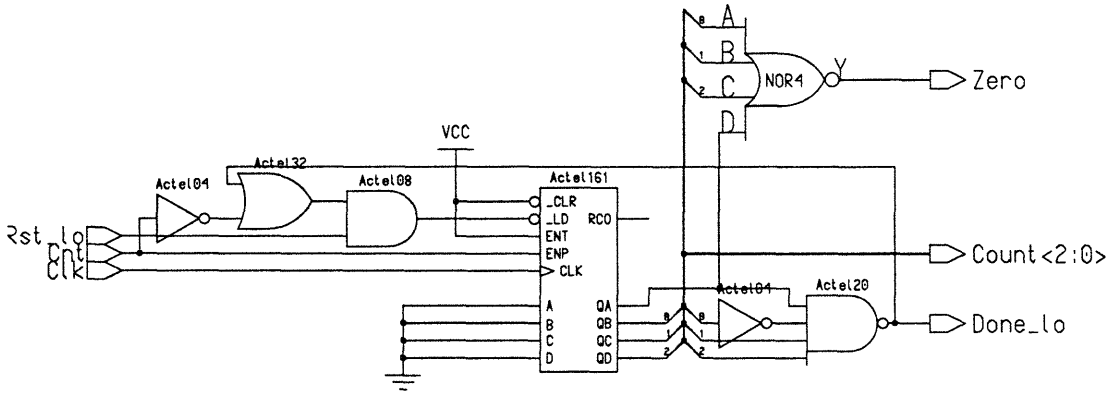

Figure E.1: 7cnt schematic

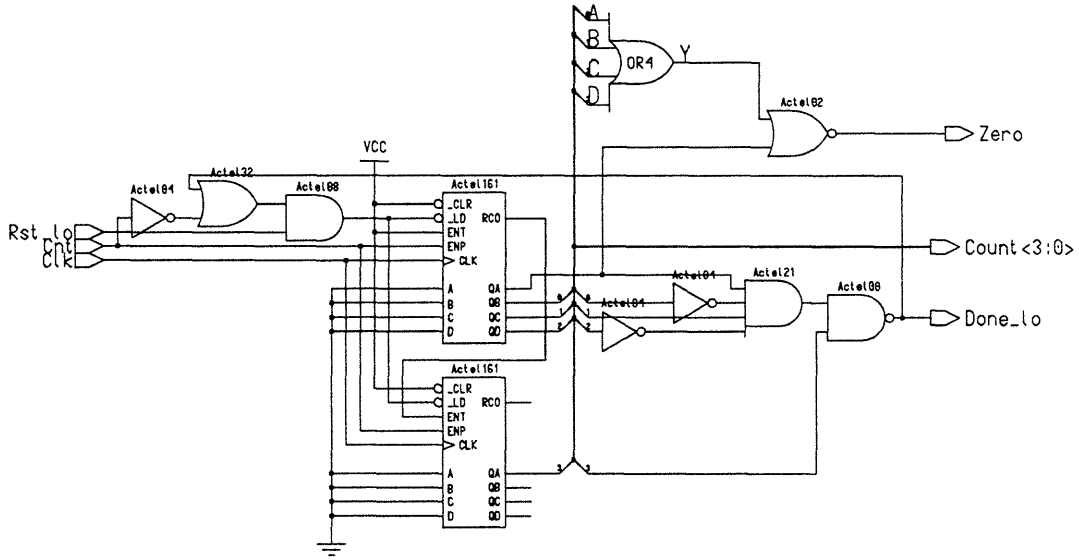

Figure E.2: 11cnt schematic

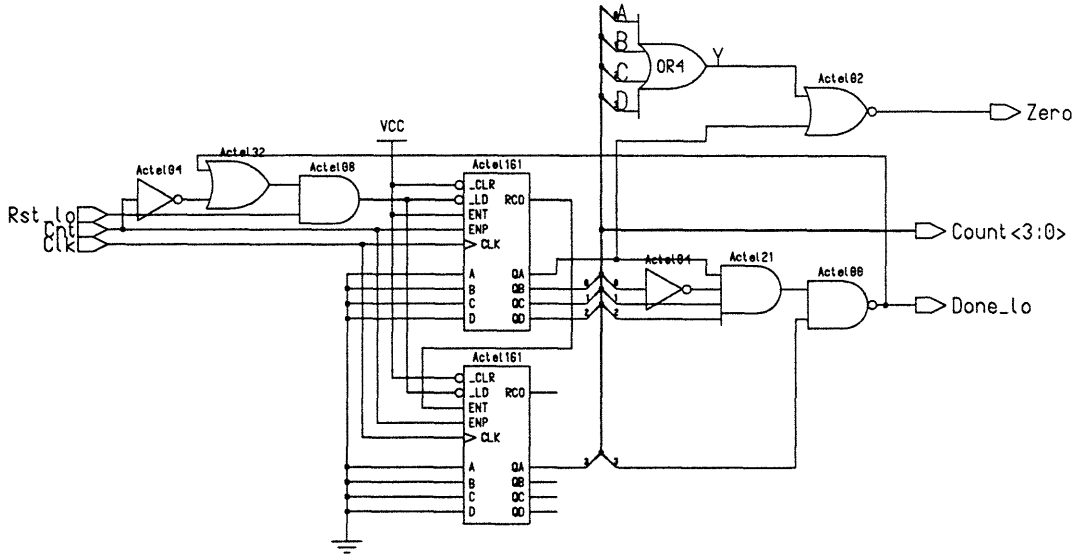

Figure E.3: 15cnt schematic

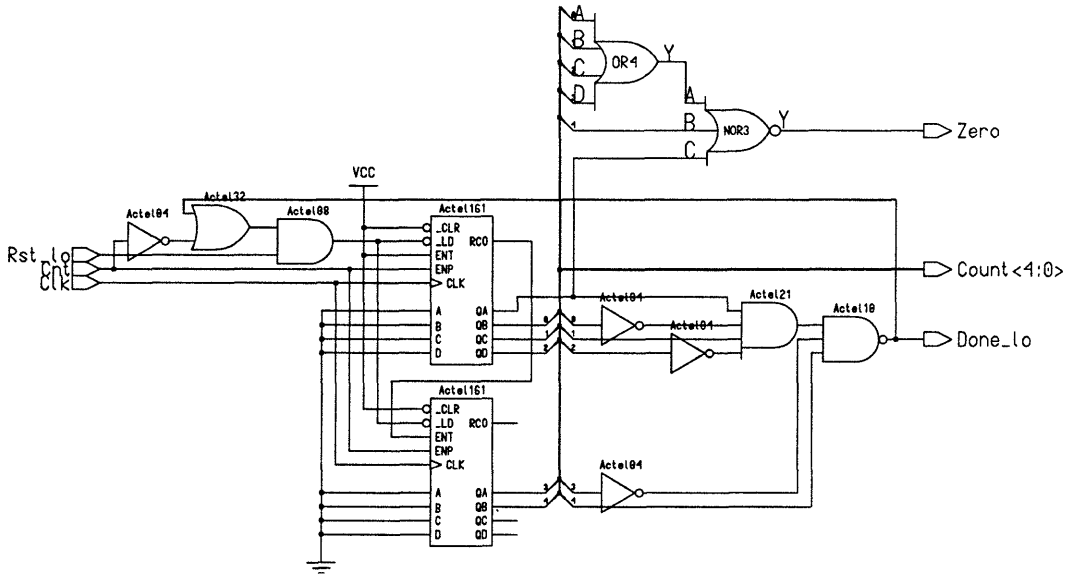

Figure E.4: 19cnt schematic

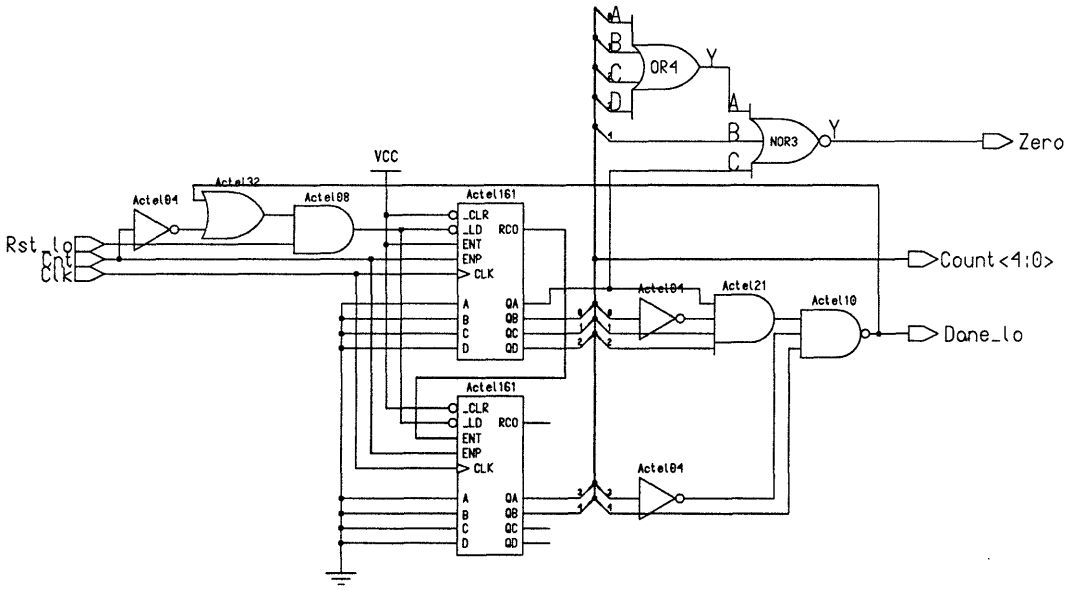

Figure E.5: 23cnt schematic

Block Name: *X*code (**X** is 7,11,15,19,23)

Synopsis:

Input Signals:

none

Output Signals:

*X*code<*n*:0>: A bit vector containing the length-**n** PN component code. (**X** is 7,11,15,19,23) (**n** is 6,10,14,18,22).

Parameters:

none

Functional Description:

This block generates a bit vector containing the length-**n** PN component code.

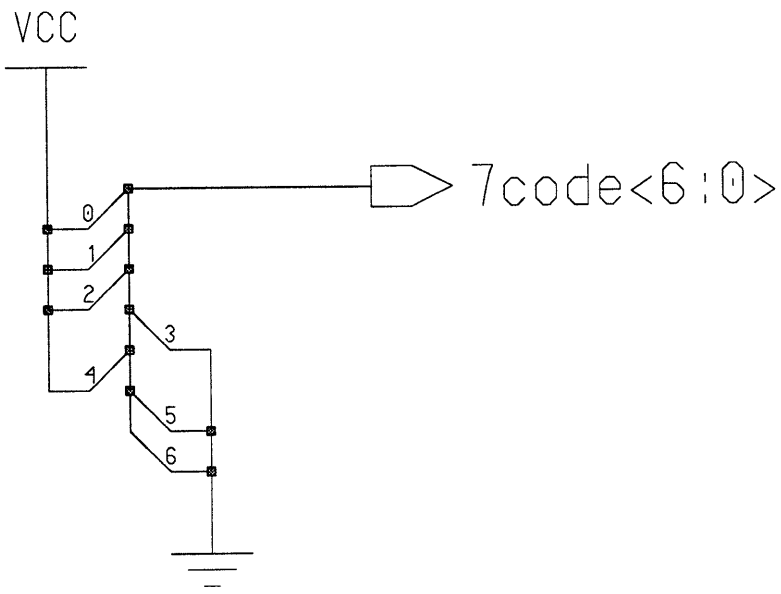

Figure E.6: 7code schematic

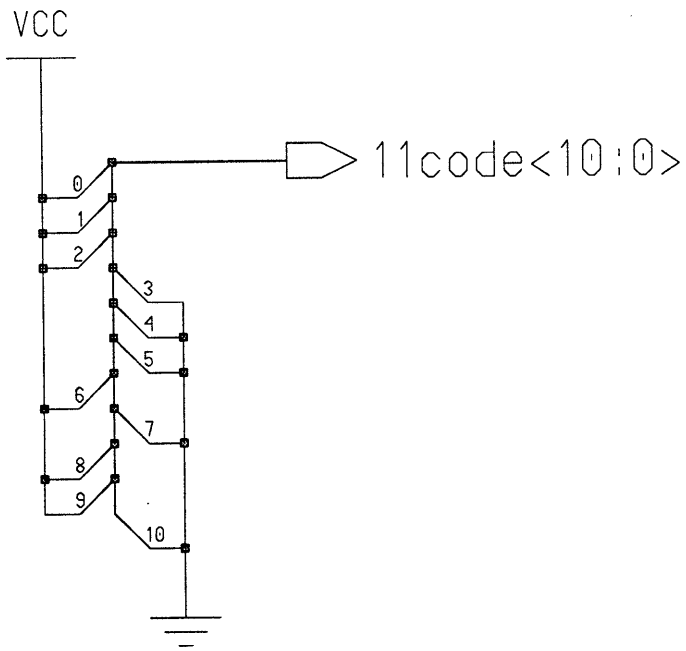

Figure E.7: 11code schematic

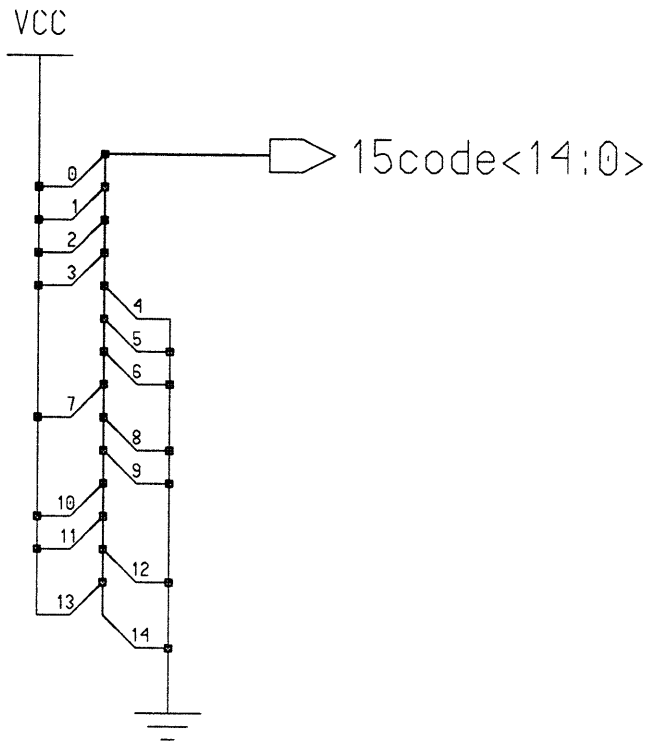

Figure E.8: 15code schematic

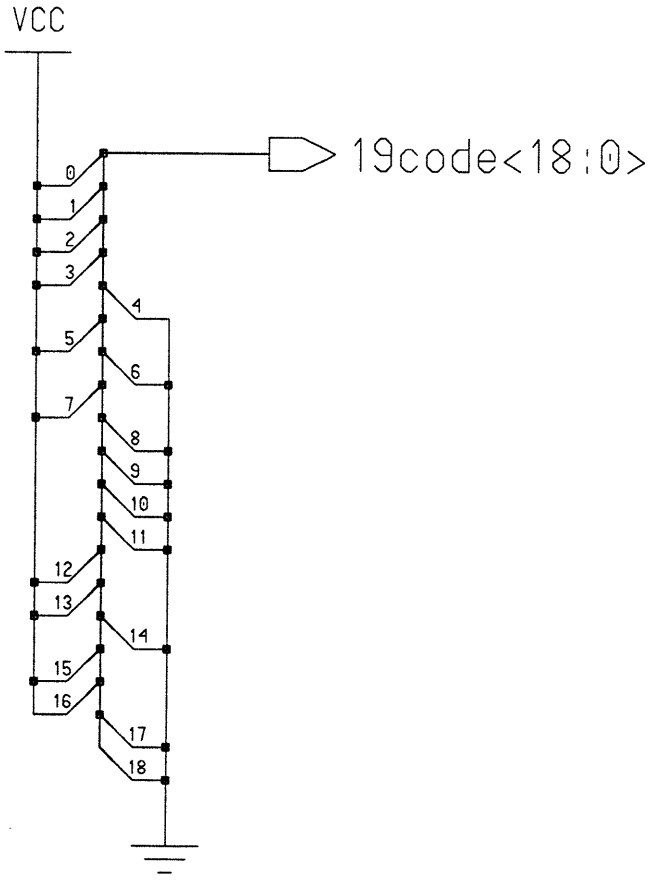

Figure E.9: 19code schematic

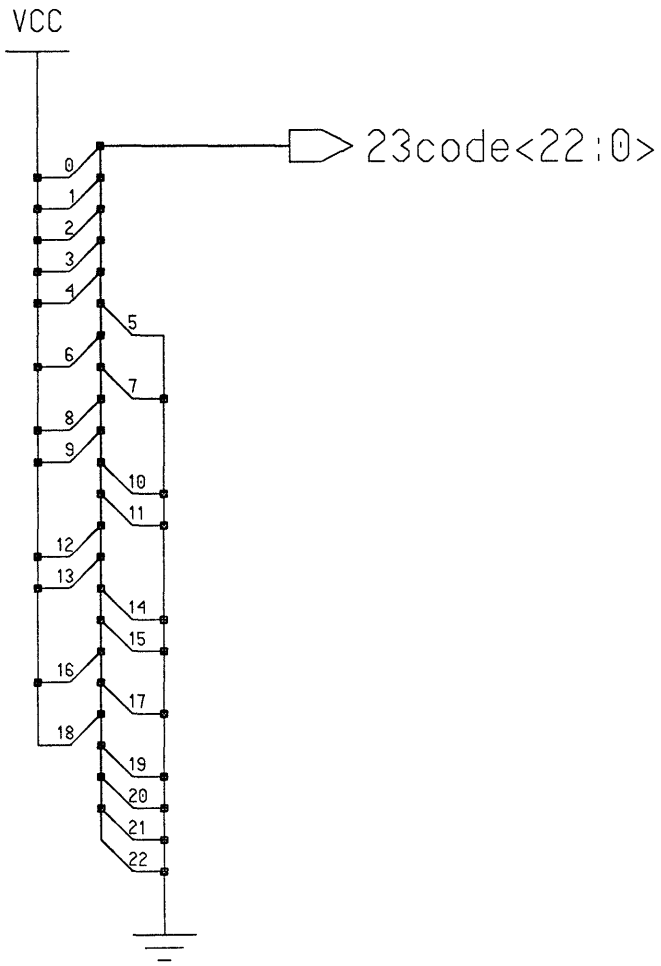

Figure E.10: 23code schematic

Block Name: *X*decode (**X** is 7,11,15,19,23)

Synopsis:

Input Signals:

Count<*n*:0>: The inputs to be decoded (**n** is 2,3,4).

Output Signals:

Reg_oe_lo<*m*:0>: The decoded outputs (negative assertion) (**m** is 6,10,14,18,22).

Parameters:

none

Functional Description:

The decoder functions to assert one bit of the output at a time, depending on which bit is specified by the input bus. For example, if **Count<2:0>** has the value 101, then **Reg_oe_lo<5>** will be asserted (low), and all other bits of **Reg_oe_lo** (<6>,<4:0>) will be deasserted (high).

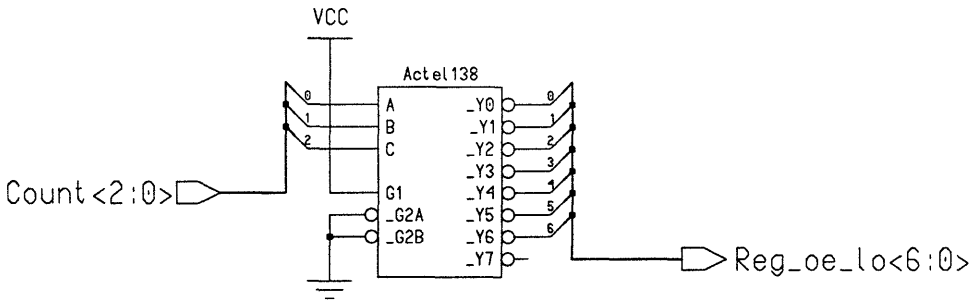

Figure E.11: 7decode schematic

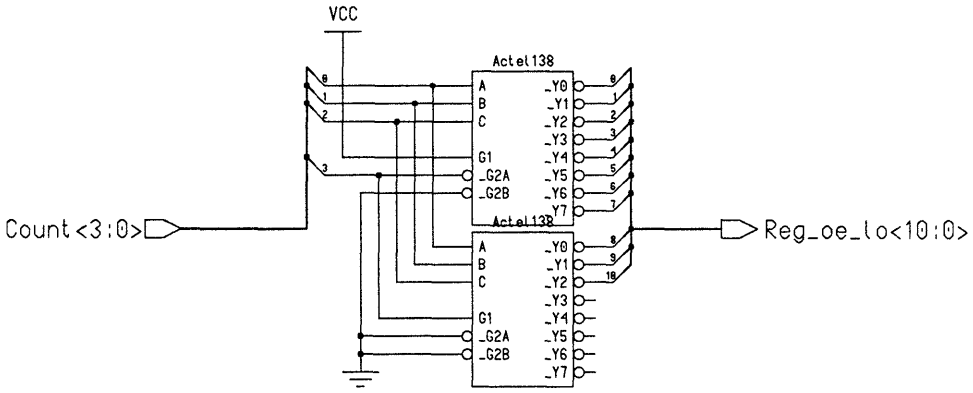

Figure E.12: 11decode schematic

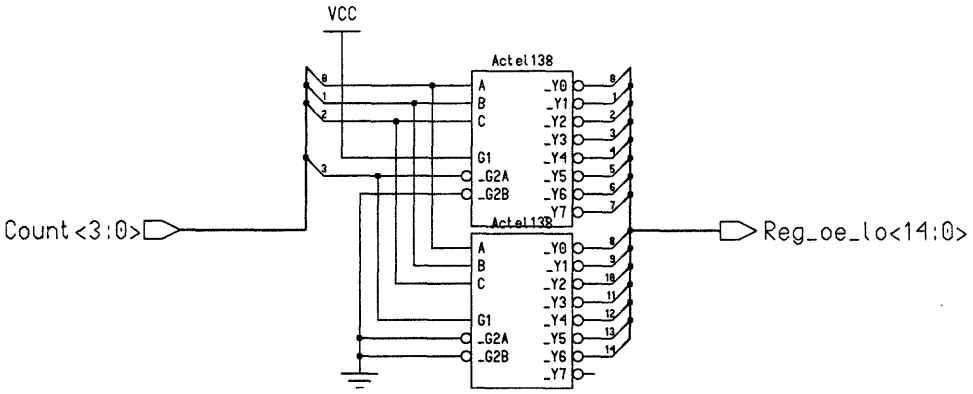

Figure E.13: 15decode schematic

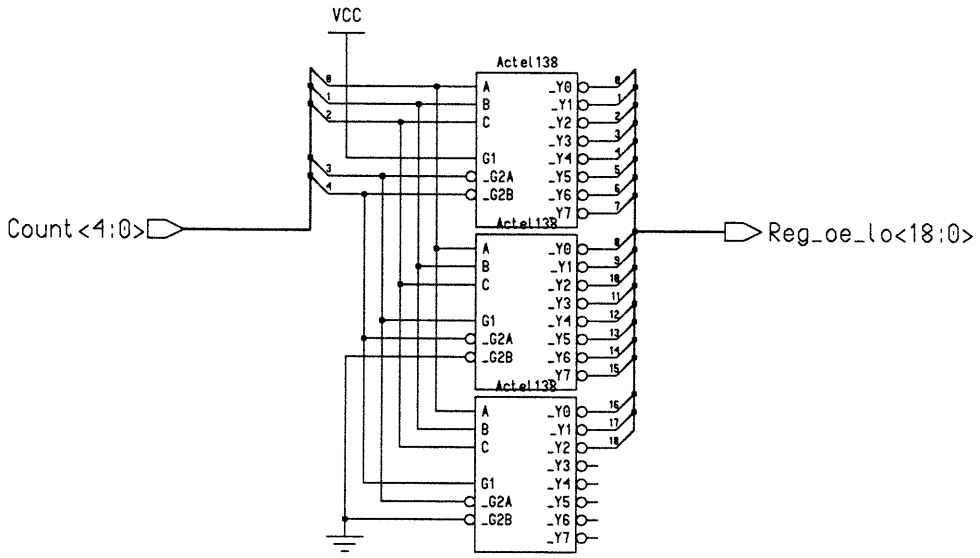

Figure E.14: 19decode schematic

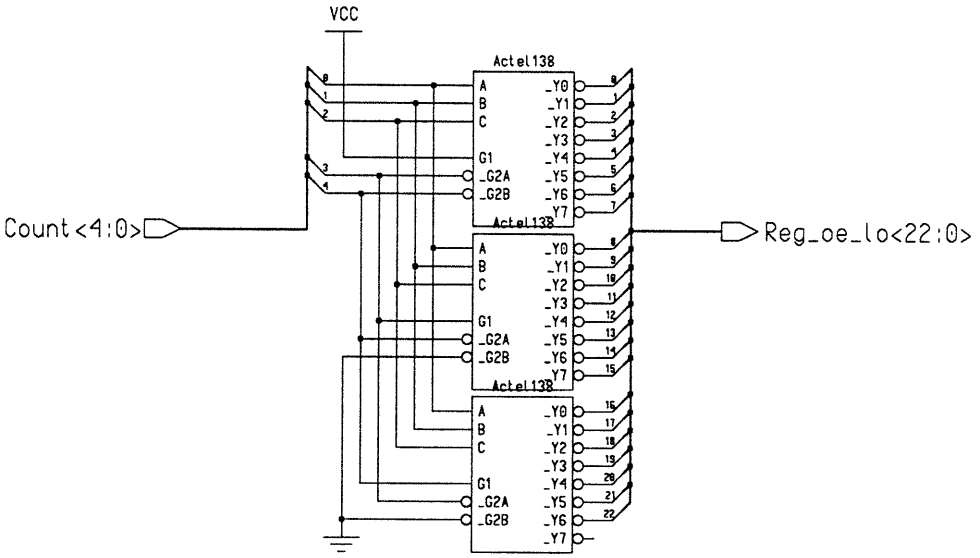

Figure E.15: 23decode schematic

Block Name: *X*gen (**X** is 7,11,15,19,23)

Synopsis:

Input Signals:

Clk: The clock signal.
Hlda_lo: A control signal that holds the generator (negative assertion).
Hldb_lo: A control signal that holds the generator (negative assertion).
In_prl<*n*:0>: Parallel input for PN component code (**n** is 6,10,14,18,22).
LD: The parallel load signal.

Output Signals:

Out_prl<*n*:0>: The parallel output for exporting internal state (**n** is 6,10,14,18,22).
Out_ser: The serial output (low bit of **Out_prl**).

Parameters:

none

Functional Description:

This block is essentially a circular shift register. Once loaded with the PN component code (via the parallel input), it repetitively outputs the code, bit by bit. When **LD** is asserted, the registers are loaded from the parallel input. When either **Hlda_lo** or **Hldb_lo** is asserted, and **LD** is not asserted, the shift register is held, and doesn't shift. If no control signals are asserted, the normal state is for the register to shift every clock cycle.

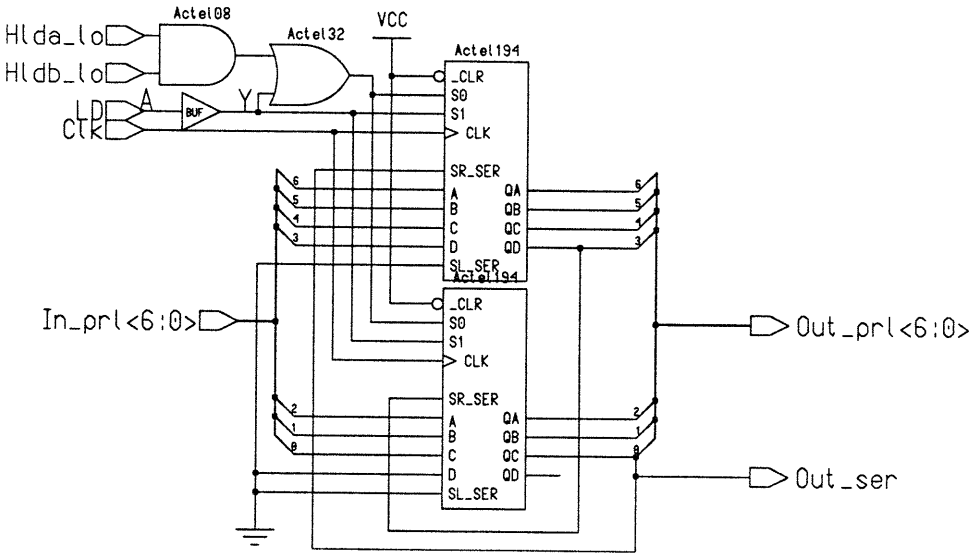

Figure E.16: 7gen schematic

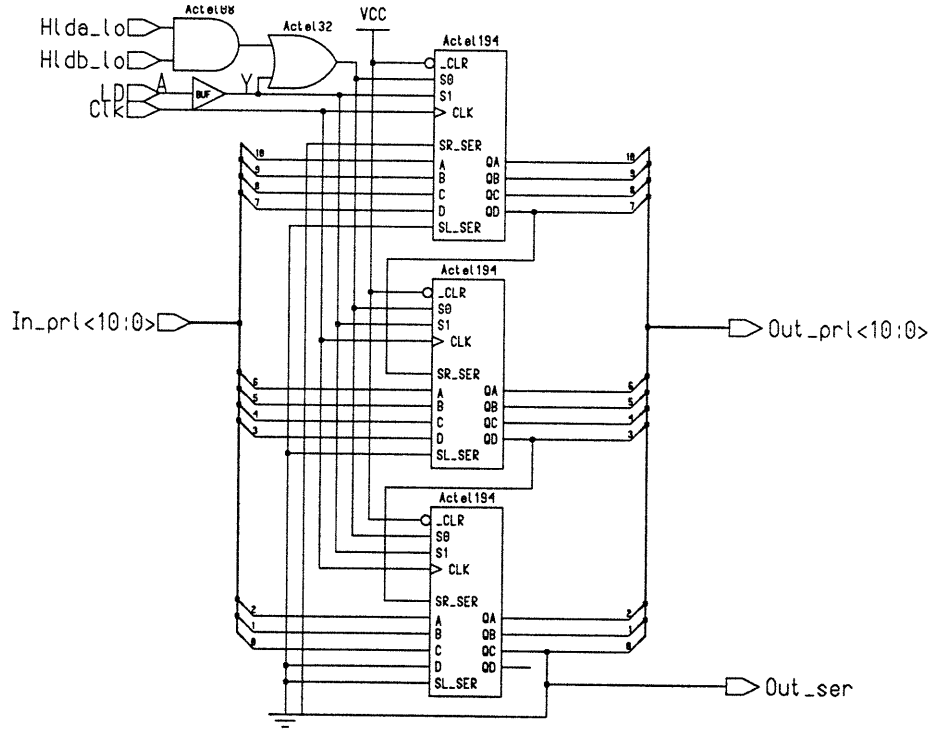

Figure E.17: 11gen schematic

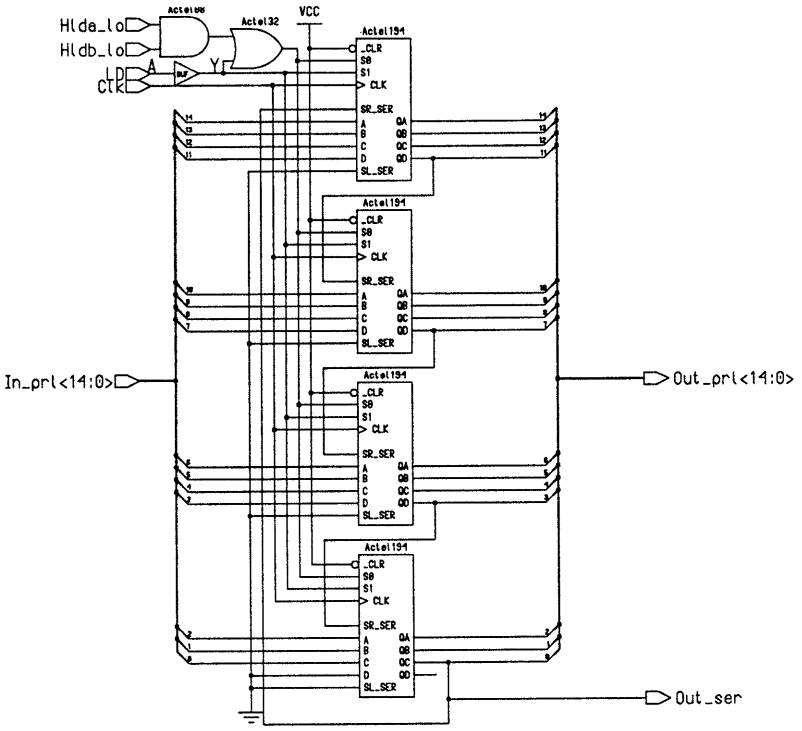

Figure E.18: 15gen schematic

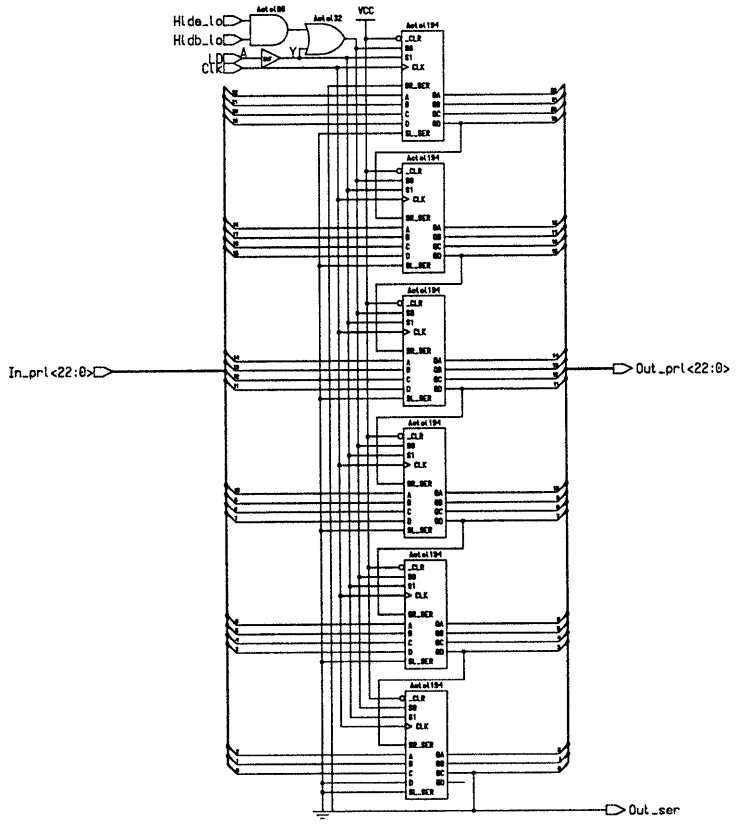

Figure E.20: 23gen schematic

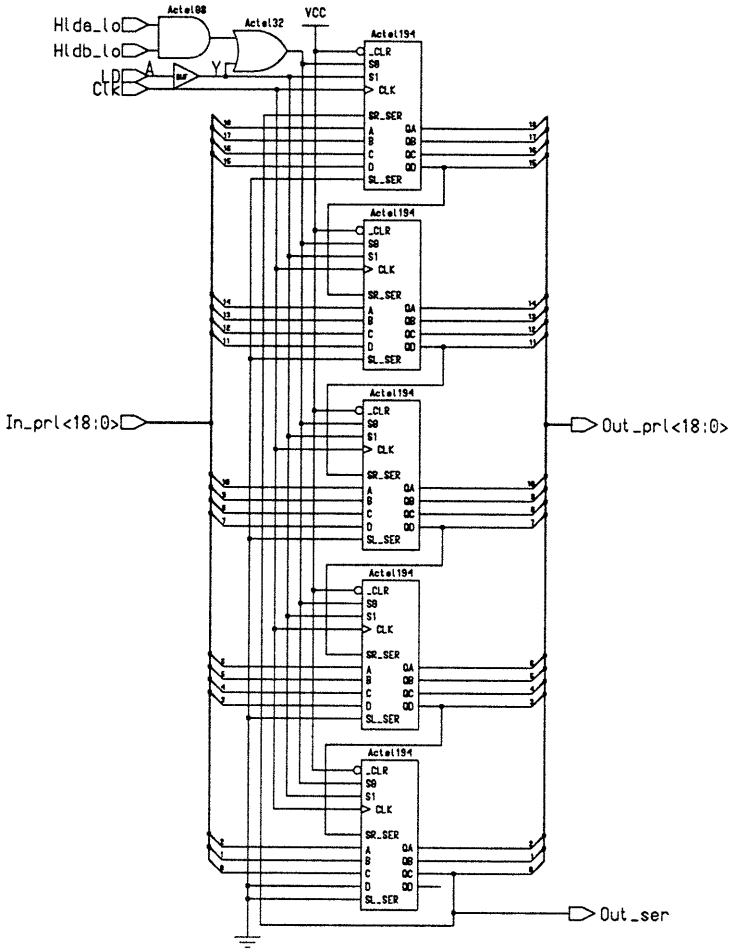

Figure E.19: 19gen schematic

Block Name: *X*rec (X is 7,11,15,19,23)

Synopsis:

Input Signals:

2xClk: A clock signal at twice the frequency of the PN clock component.
2xClk_ph2: A clock signal at twice the frequency of the PN clock component, advanced by 90^0 from **2xClk**.
4xClk: A clock signal at four times the frequency of the PN clock component.
PN_seq<7:0>: The PN sequence, digitized with 8-bit precision.
reset_lo: The reset signal (negative assertion).

Output Signals:

code_phase<*n*:0>: The state of the internal PN component generator (**n** is 6,10,14,18,22).
ld_out: A signal indicating when the internal state should be loaded by the external generator.

Parameters:

none

Functional Description:

This block performs the PN component recovery. It correlates a locally generated PN component against the received PN sequence for 64 periods of the component, then finds the peak in the correlation, and aligns the local generator so that the peak will occur at zero delay. It then asserts **ld_out**, on the first quarter of a **2xClk** cycle, to tell the external PN component generator to reload its state from the internal generator. Because the load operation takes a clock cycle, the local state is shifted back by one bit before it is exported.

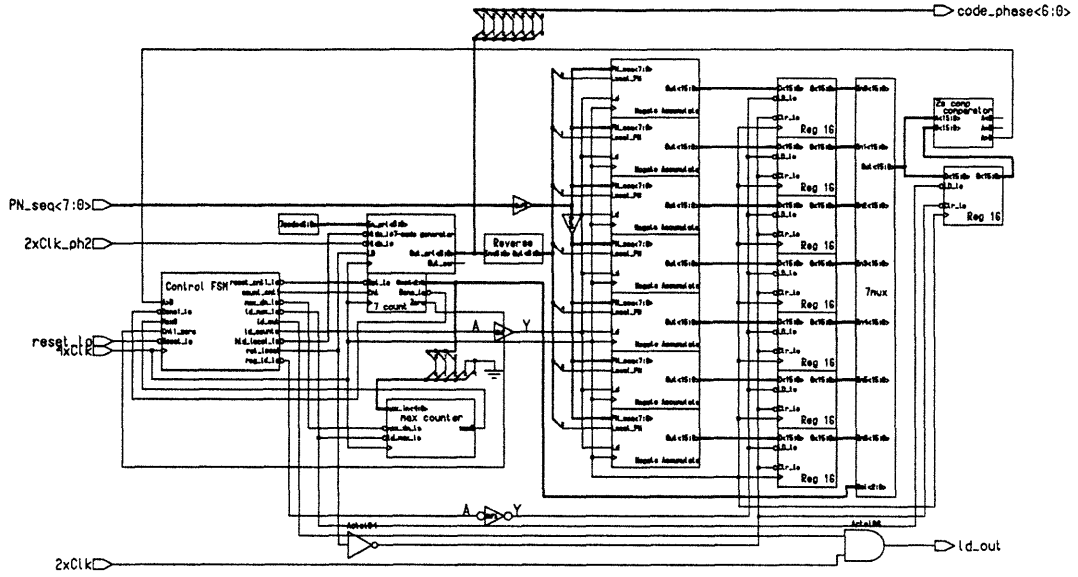

Figure E.21: 7rec schematic

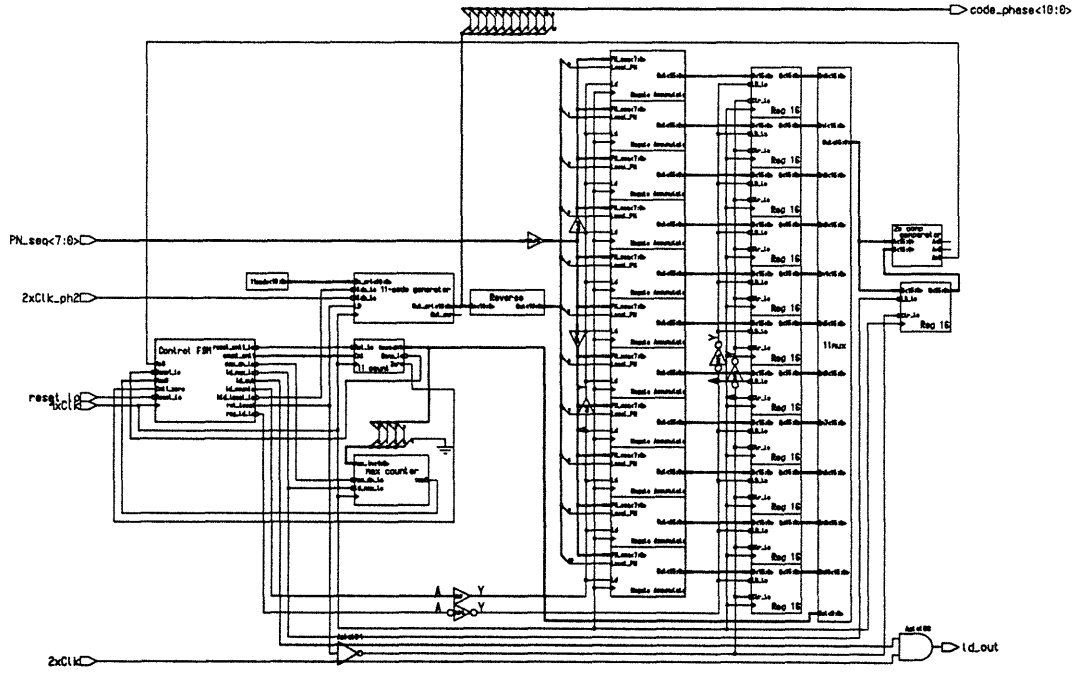

Figure E.22: 11rec schematic

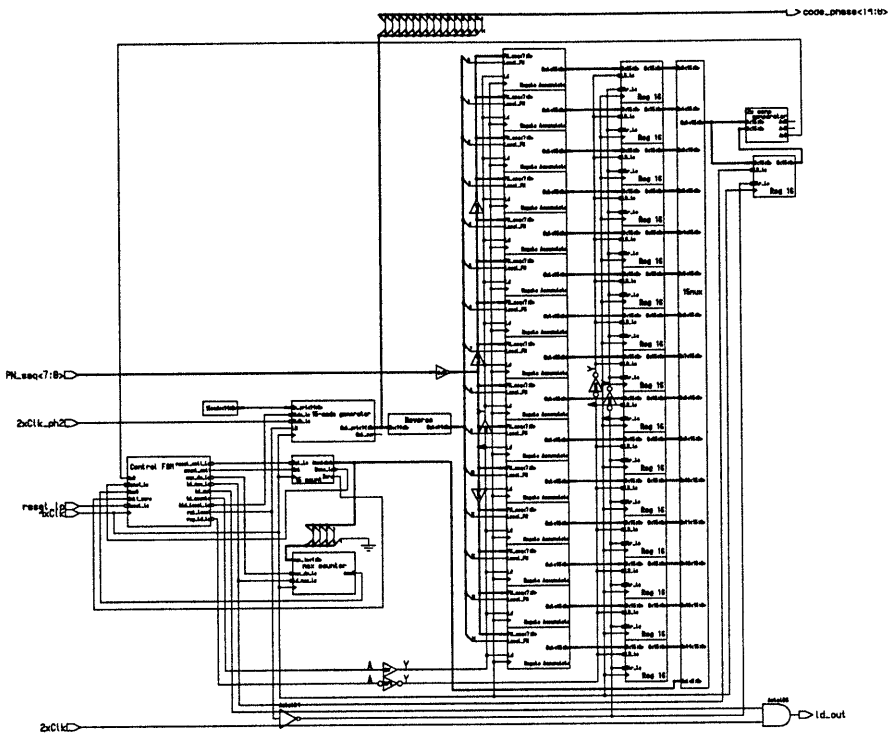

Figure E.23: 15rec schematic

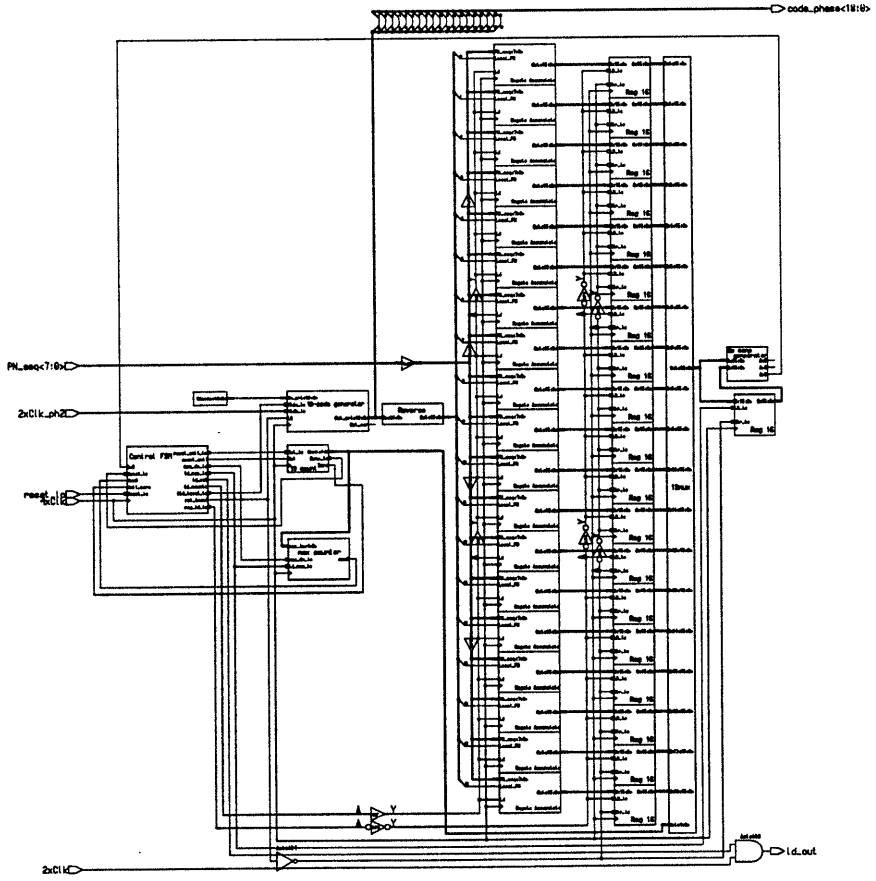

Figure E.24: 19rec schematic

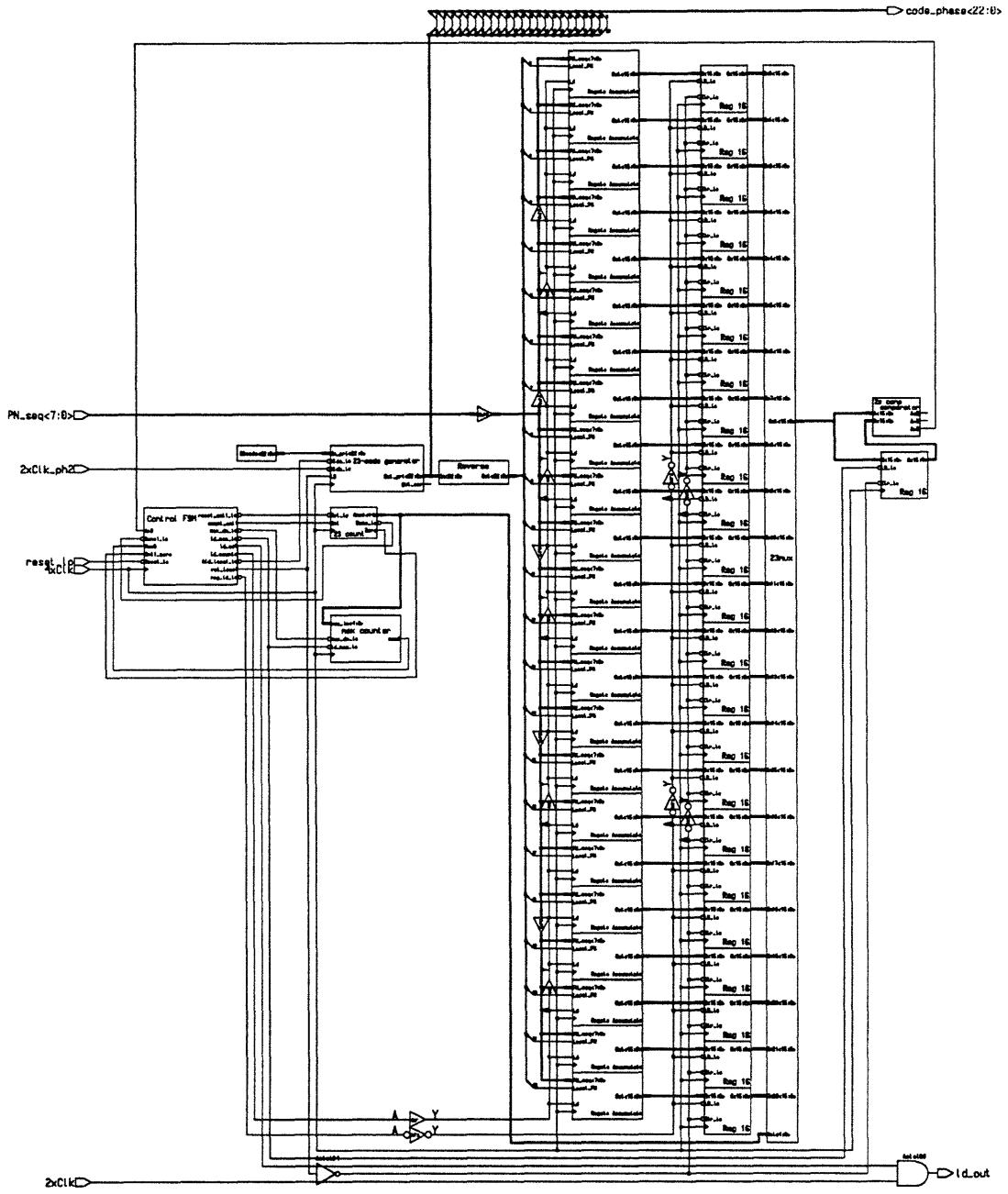

Figure E.25: 23rec schematic

Block Name: *X*reverse (X is 7,11,15,19,23)

Synopsis:

Input Signals:

In<*n*:0>: The input signal (n is 6,10,14,18,22).

Output Signals:

Out<*m*:0>: The output signal (m is 6,10,14,18,22).

Parameters:

none

Functional Description:

This block reverses the bits on the input bus, sending the result to the output. For example, if the input vector is 1100110, the output vector will be 0110011.

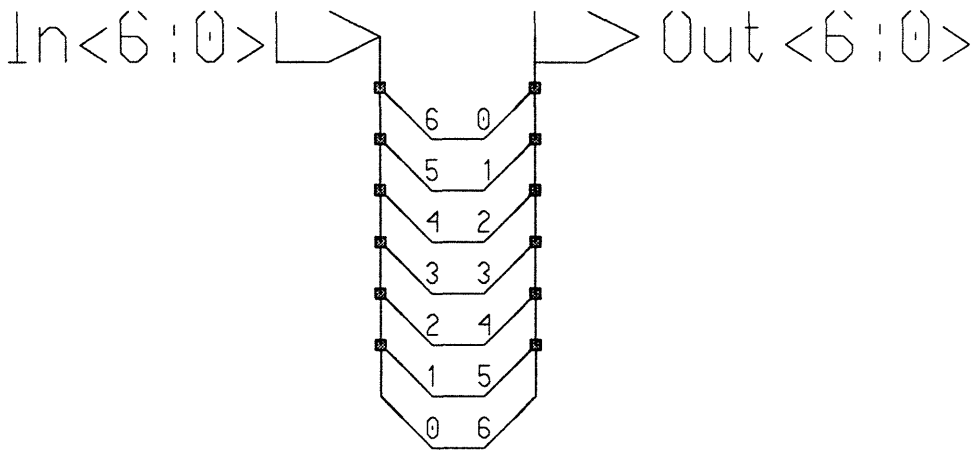

Figure E.26: 7reverse schematic

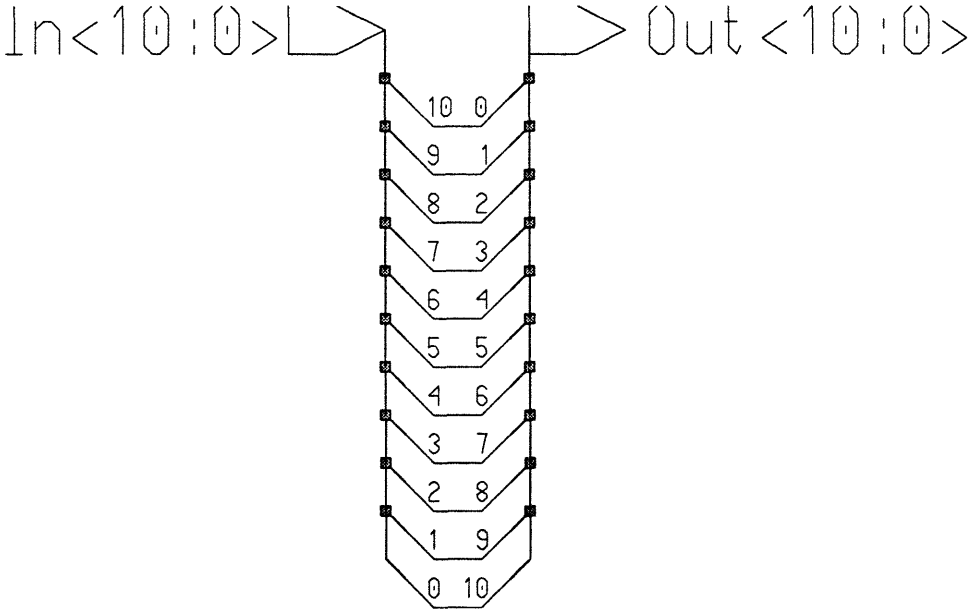

Figure E.27: 11reverse schematic

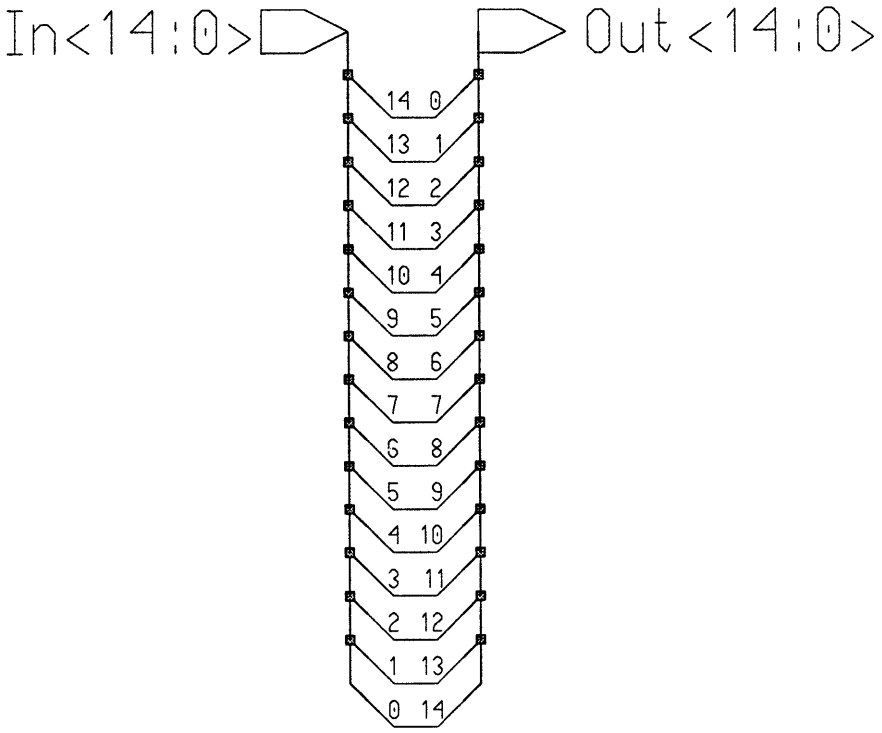

Figure E.28: 15reverse schematic

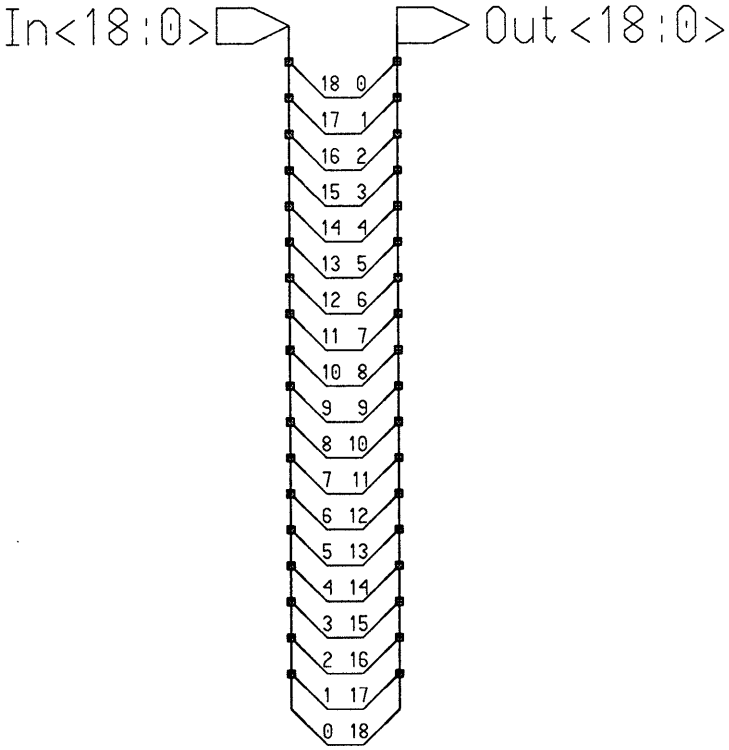

Figure E.29: 15reverse schematic

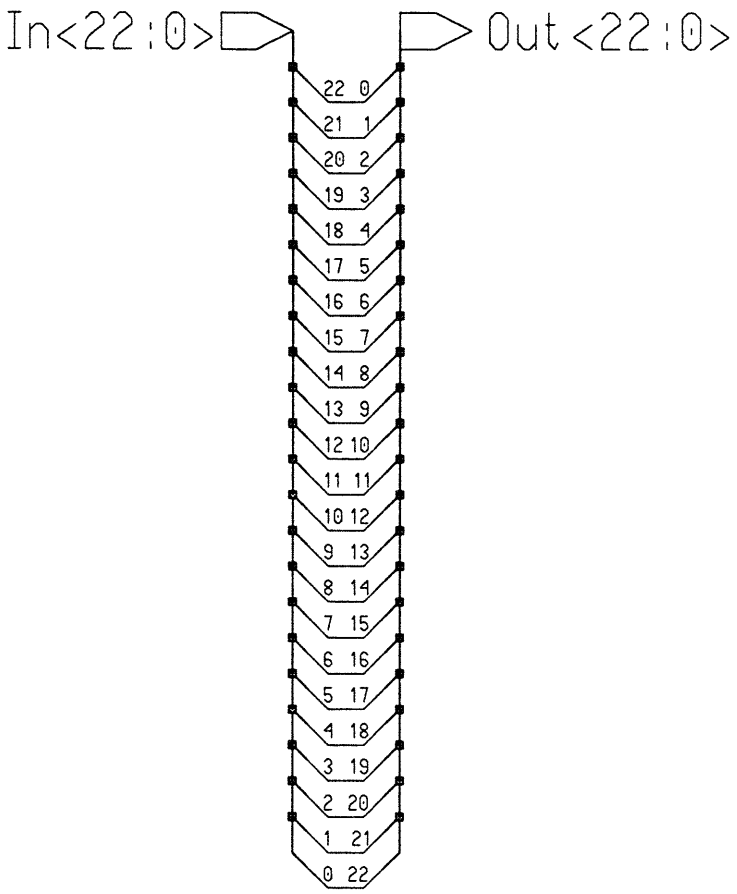

Figure E.30: 23reverse schematic

Block Name: 32cnt

Synopsis:

Input Signals:

Clk: The clock signal.
Cnt: The count enable signal.
Rst_lo: The counter reset signal (negative assertion).

Output Signals:

Done_lo: The carry signal (asserted when counter is about to roll back to zero).

Parameters:

none

Functional Description:

This block counts from zero to 31, then rolls over back to zero. It increments the counter once every other clock cycle, when **Cnt** is asserted. The reset is synchronous, and the rollover is synchronous and waits for **Cnt** to be asserted.

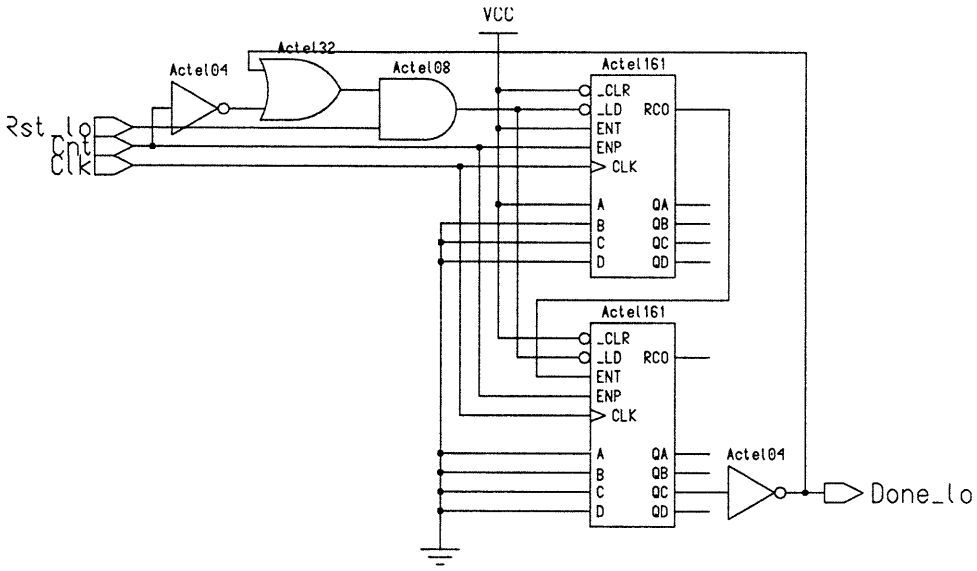

Figure E.31: 32cnt schematic

Block Name: **and16**

Synopsis:

Input Signals:

A<15:0>: A sixteen-bit input word.
B: A one-bit input value.

Output Signals:

Out<15:0>: A sixteen-bit output word.

Parameters:

none

Functional Description:

This block computes the bitwise logical AND of **A** and **B**. The result is placed into **Out**.

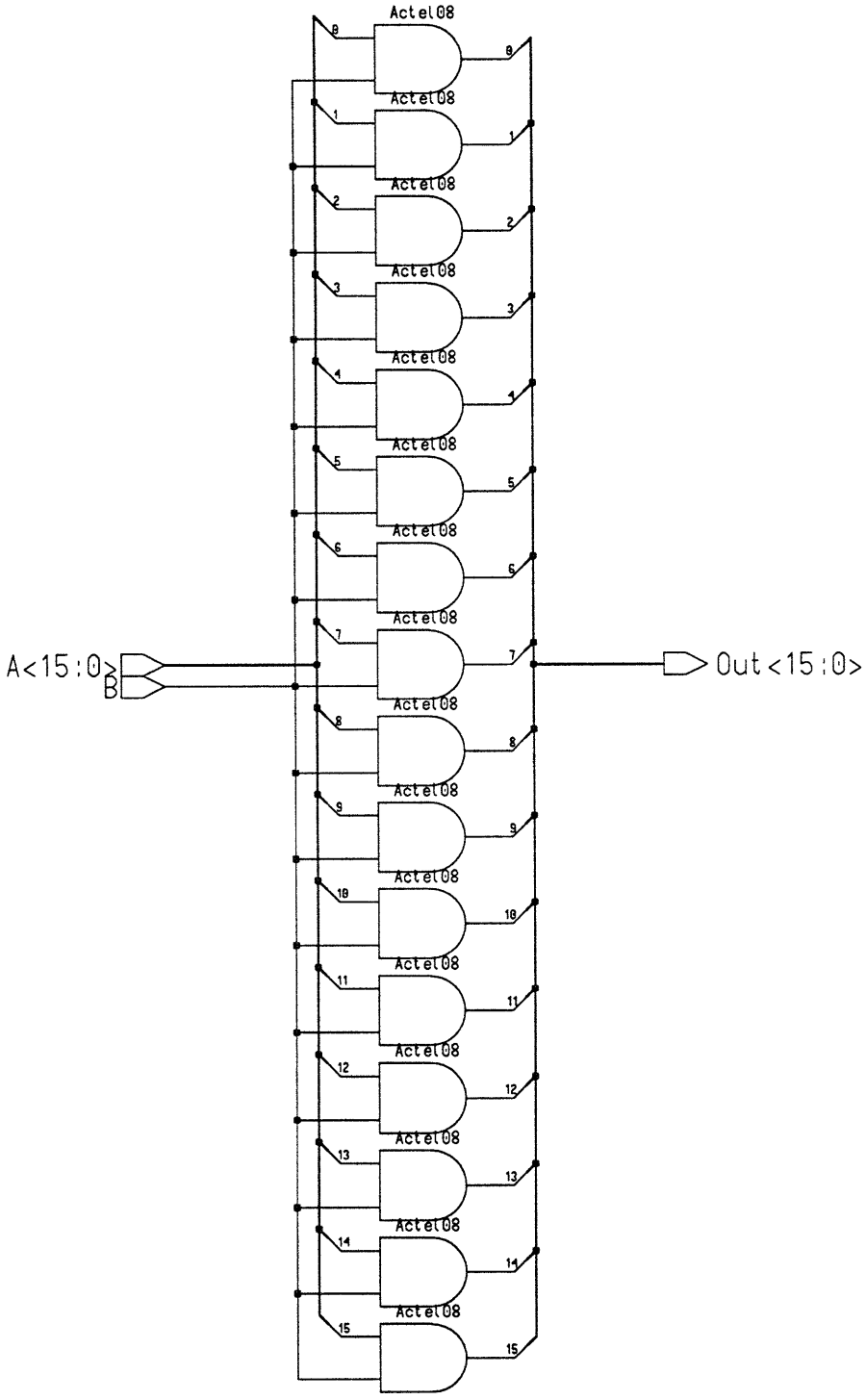

Figure E.32: and16 schematic

Block Name: **cmp16**

Synopsis:

Input Signals:

A<15:0>: A sixteen-bit input word (twos complement encoded).
B<15:0>: A sixteen-bit input word (twos complement encoded).

Output Signals:

A<B: An output that is high when **A** is less than **B**.
A=B: An output that is high when **A** is equal to **B**.
A>B: An output that is high when **A** is greater than **B**.

Parameters:

none

Functional Description:

This block compares **A** and **B**, setting its outputs based on their numerical values (which are twos complement numbers). Because the twos complement encoding needs to be taken into account, the MSB of each input is inverted, converting the inputs to straight binary.

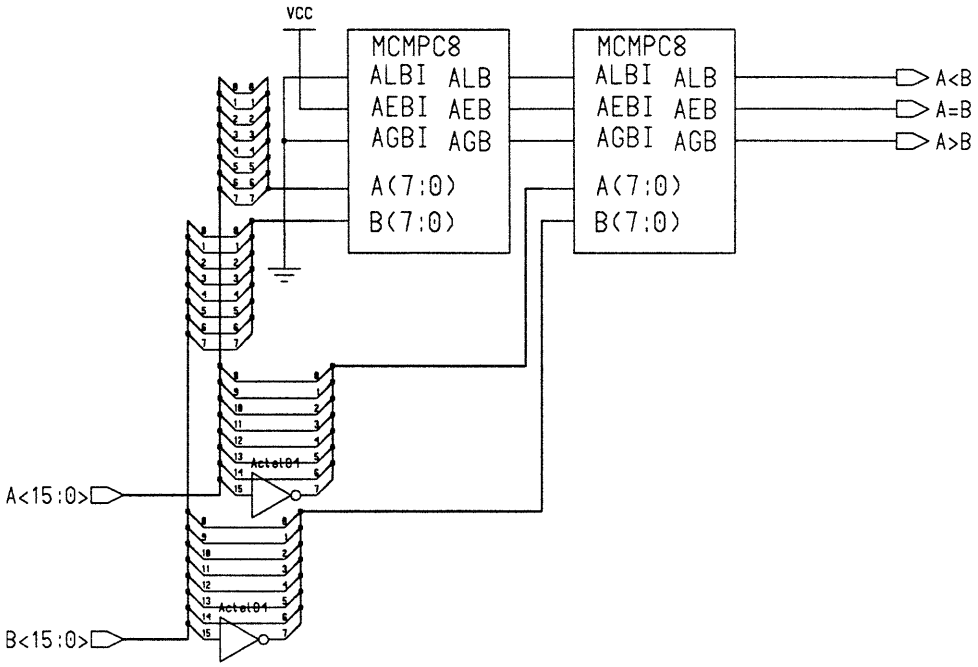

Figure E.33: cmp16 schematic

Block Name: **combiner**

Synopsis:

Input Signals:

clock: The PN clock component.
In<4:0>: A bit vector containing the other five PN components.

Output Signals:

Out: The combined PN sequence.

Parameters:

none

Functional Description:

This block combines the PN component into the final PN sequence. The algorithm it follows is:

$$Out = clock \cdot OR(In) + AND(In) \tag{E.1}$$

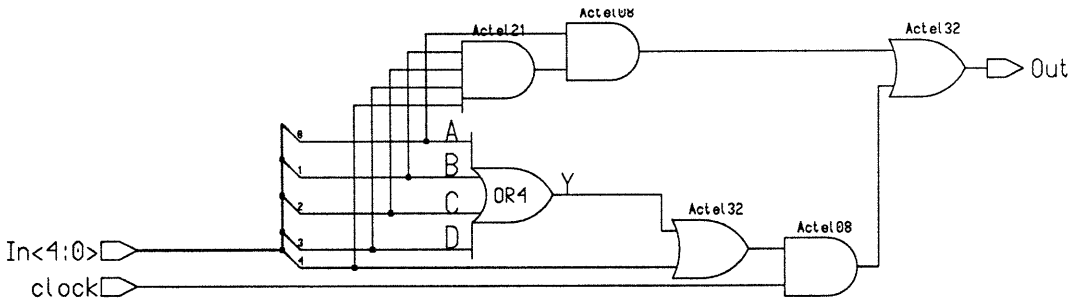

Figure E.34: combiner schematic

Block Name: control

Synopsis:

Input Signals:

A>B: A signal indicating that value on bus is greater than value in register.
Clk: The clock signal.
Cnt1_zero: A signal indicating that all bits of counter 1 are zero.
Done1_lo: A signal indicating that counter 1 is about to roll back to zero (negative assertion).
Max0: A signal indicating that all bits of maxcount are zero.
Reset_lo: The local reset signal (negative assertion).

Output Signals:

count_cnt1: A signal telling counter 1 to increment.
hld_local_lo: A signal telling the local PN generator to hold (negative assertion).
ld_counts: A signal that resets the correlation accumulators.
ld_max_lo: A signal that loads the output of counter 1 into maxcount (- logic).
ld_out: A signal telling the external PN generator to load new state.
max_dn_lo: A signal telling maxcount to decrement (negative assertion).
reg_ld_lo: A signal that loads the buffers with the values in the accumulators (-).
reset_cnt1_lo: A signal telling counter 1 to reset (negative assertion).
rst_local: A signal telling the internal PN generator to load new state.

Parameters:

none

Functional Description:

This block is the control FSM for each component recovery system.

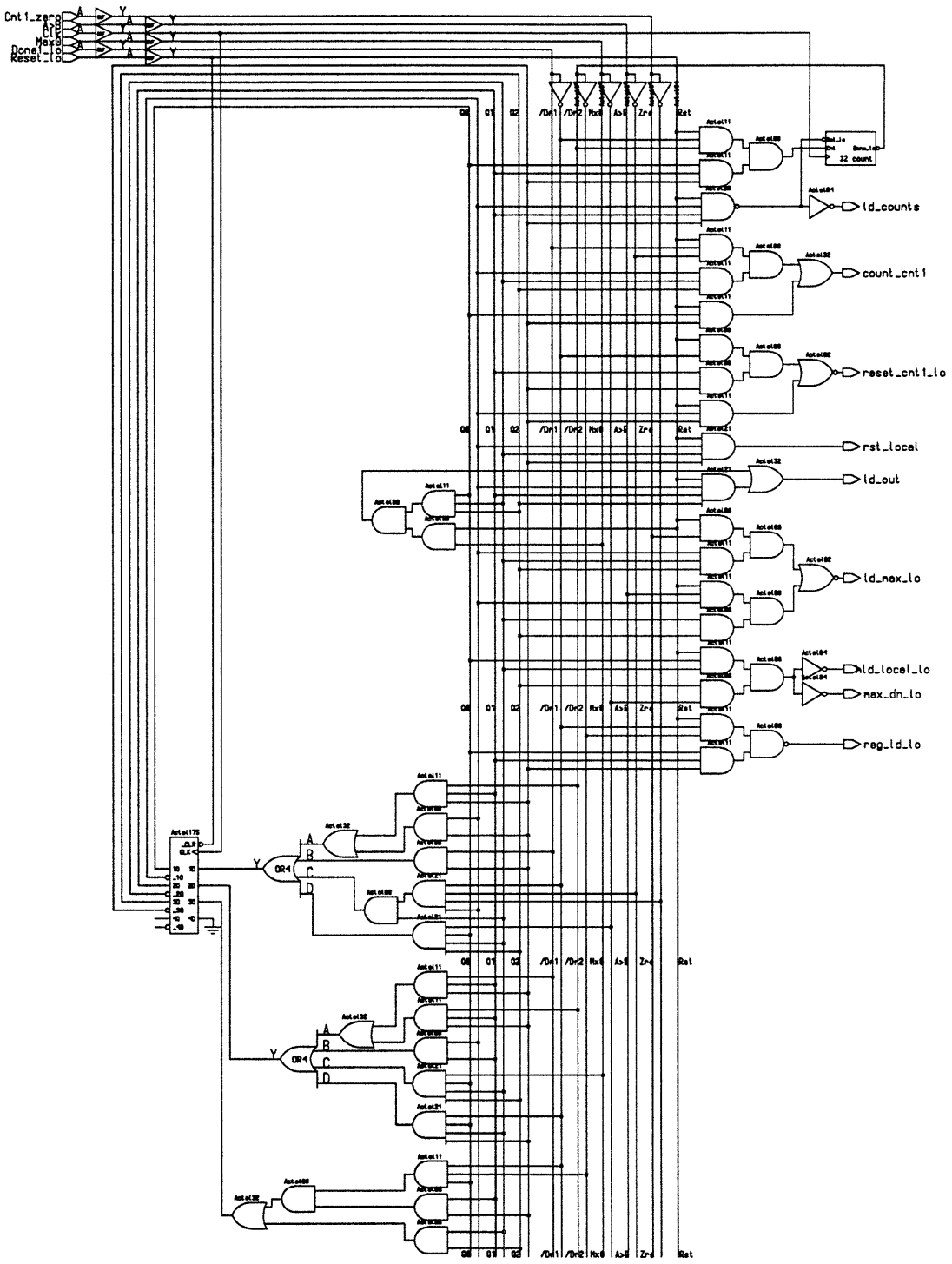

Figure E.35: control schematic

Block Name: generate

Synopsis:

Input Signals:

2xClk_ph2: A clock at twice the frequency of the PN clock component, and delayed by 90^{o}.

4xClk: A clock at four times the frequency of the PN clock component.

Hld_lo<4:0>: Signals to hold each individual PN component generator (negative assertion).

Noise<7:0>: Noise to be added to the final PN sequence.

PN_Clk: The PN clock component.

reset: A local reset signal.

Output Signals:

PN_seq<7:0>: The combined PN sequence, with 8 bits of precision, and added noise.

Parameters:

none

Functional Description:

This block generates all the PN components, combines them, and adds noise. The output is an 8-bit representation of the combined PN sequence, with the provided noise added in.

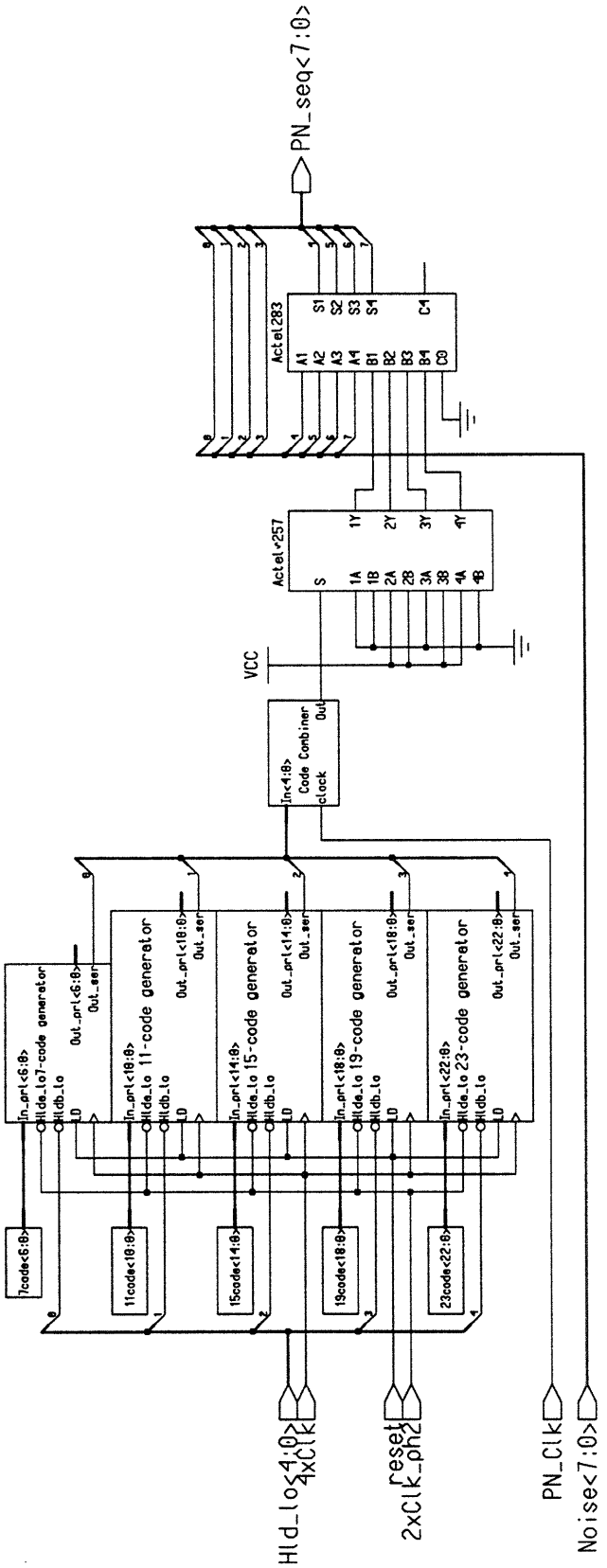

Figure E.36: generate schematic

Block Name: **maxcount**

Synopsis:

Input Signals:

Clk: The clock signal.
ld_max_lo: A signal telling maxcount to load a new value from **max_in** (negative assertion).
max_dn_lo: A signal telling maxcount to decrement (negative assertion).
max_in<4:0>: A parallel load input for the counter.

Output Signals:

max0: A signal indicating when all the bits of maxcount are zero.

Parameters:

none

Functional Description:

This block is a loadable down-counter. It loads a new value equal to twice **max_in** when **ld_max_lo** is asserted, then decrements each time **max_dn_lo** is asserted. When the counter reaches zero, **max0** is asserted.

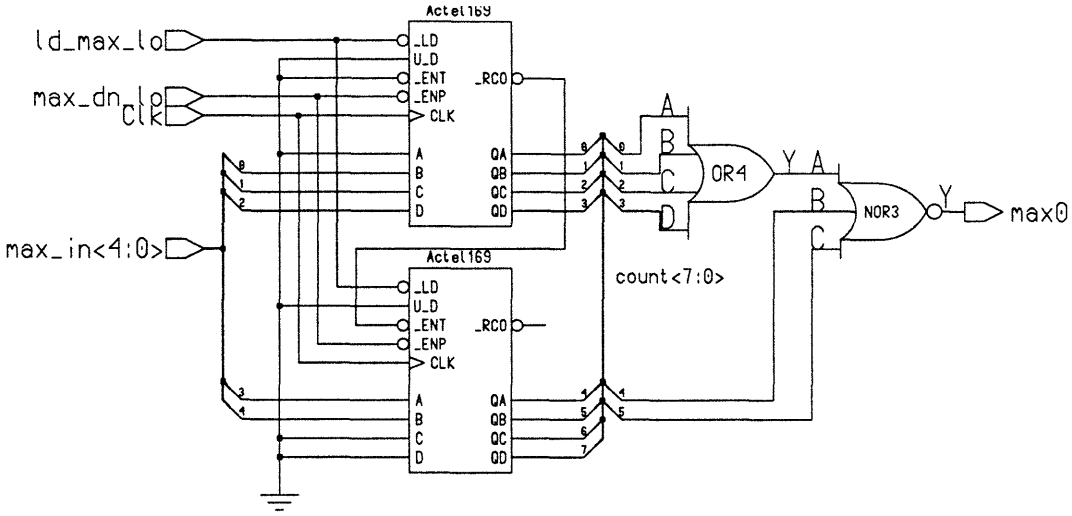

Figure E.37: maxcount schematic

Block Name: negacc

Synopsis:

Input Signals:

Clk: The clock signal.
Ld: A signal that forces the accumulator to clear (load, actually).
Local_PN: The locally generated PN component code.
PN_seq<7:0>: The received PN sequence.

Output Signals:

Out<15:0>: The output of the accumulator.

Parameters:

none

Functional Description:

This block performs the multiply and accumulate portions of a correlation. If the **Ld** signal is asserted, the accumulator is loaded with the current value, effectively clearing it. Otherwise, if the **Local_PN** signal is high, the value in **PN_seq** is sign-extended, and added into the accumulator, and if the **Local_PN** signal is low, the value in **PN_seq** is sign-extended, and subtracted from the accumulator (actually, it's 2's complement negated (invert all bits and add 1) and added).

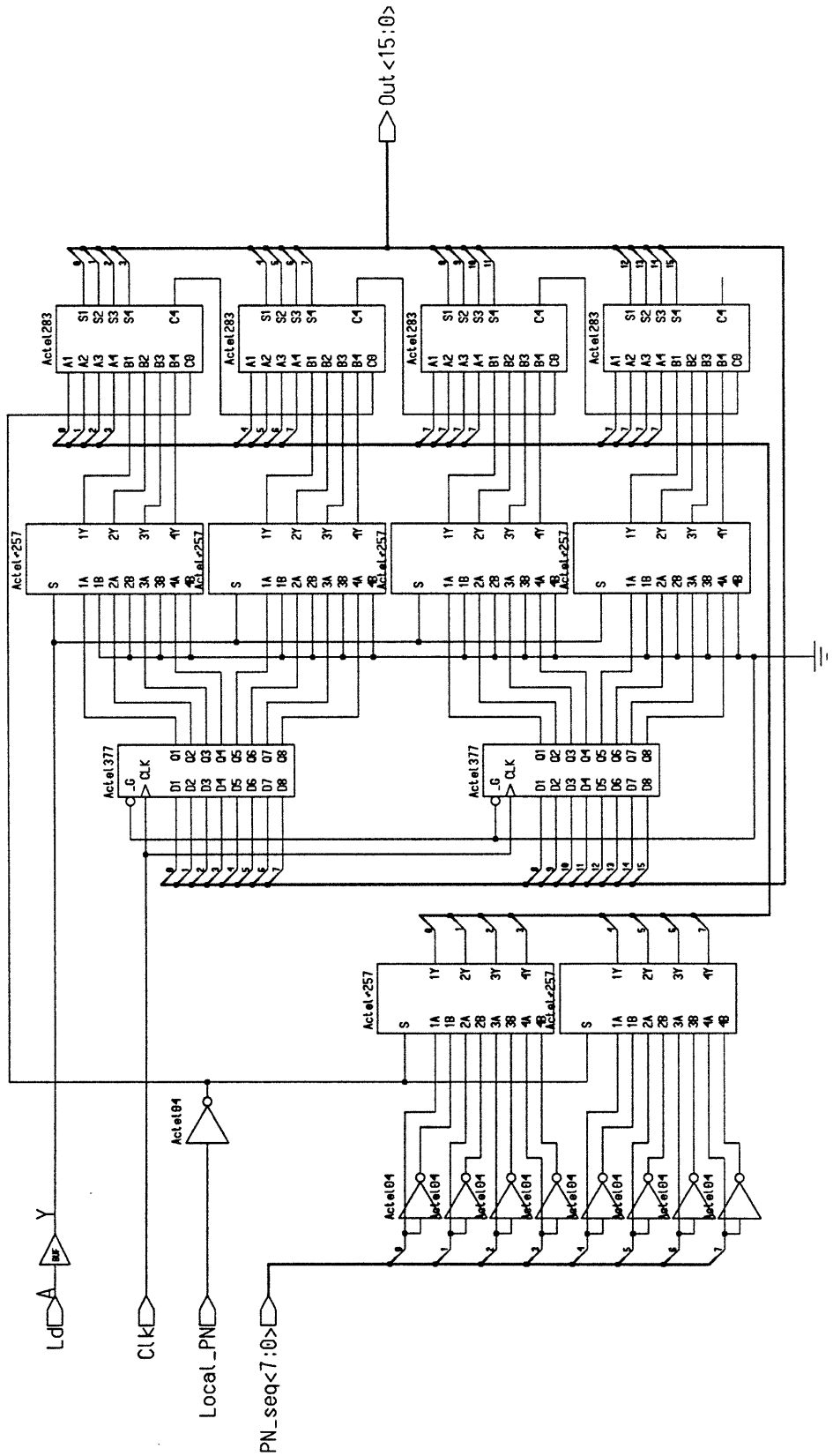

Figure E.38: negacc schematic

Block Name: **reg16**

Synopsis:

Input Signals:

Clk: The clock signal.
Clr_lo: The clear signal.
D<15:0>: The sixteen-bit D input signal.
LD_lo: The load signal.
OE_lo: The output enable signal.

Output Signals:

Q<15:0>: The registered sixteen-bit output.

Parameters:

none

Functional Description:

This block is a sixteen-bit D-type flip-flop with load, clear, and output enable. Clear takes precedence over load.

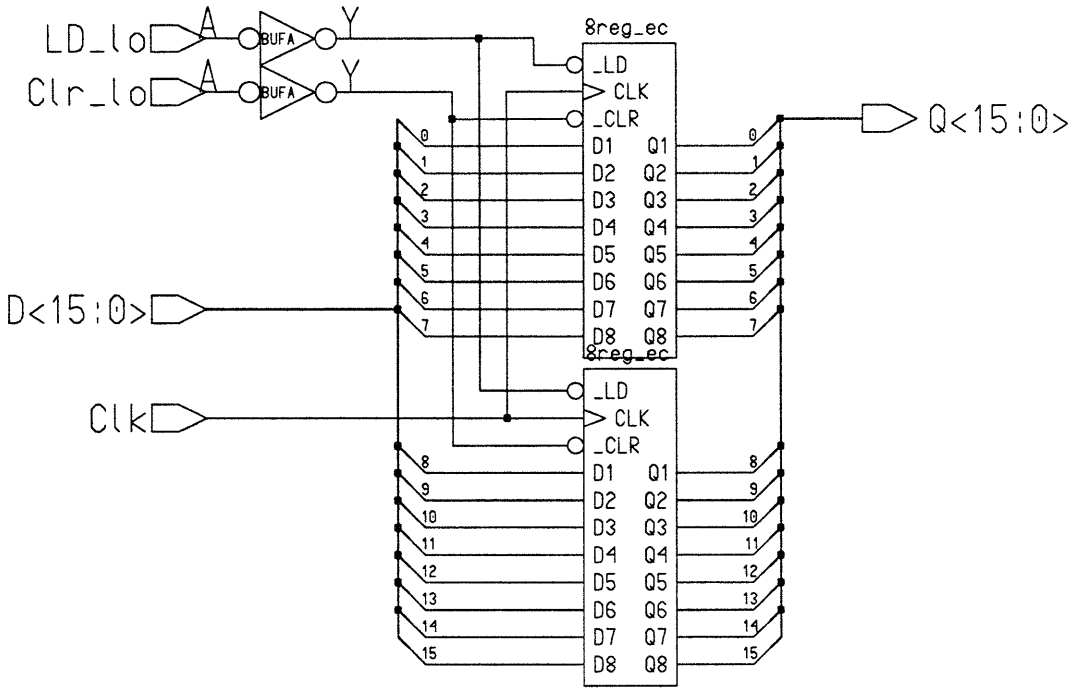

Figure E.39: reg16 schematic

Block Name: **regenerate**

Synopsis:

Input Signals:

2xClk: A clock signal at twice the frequency of the PN clock component.
2xClk_ph2: A clock signal at twice the frequency of the PN clock component, advanced by 90^{0} from **2xClk**.
4xClk: A clock signal at four times the frequency of the PN clock component.
PN_Clk: The PN clock component.
PN_seq<7:0>: The received PN sequence, digitized with eight-bit precision.
reset_lo: The local reset signal (negative assertion).

Output Signals:

recovered_PN: The combined, recovered PN sequence.

Parameters:

none

Functional Description:

This block is the high-level block for the regeneration system. It recovers each PN component code, then recombines them into the PN sequence.

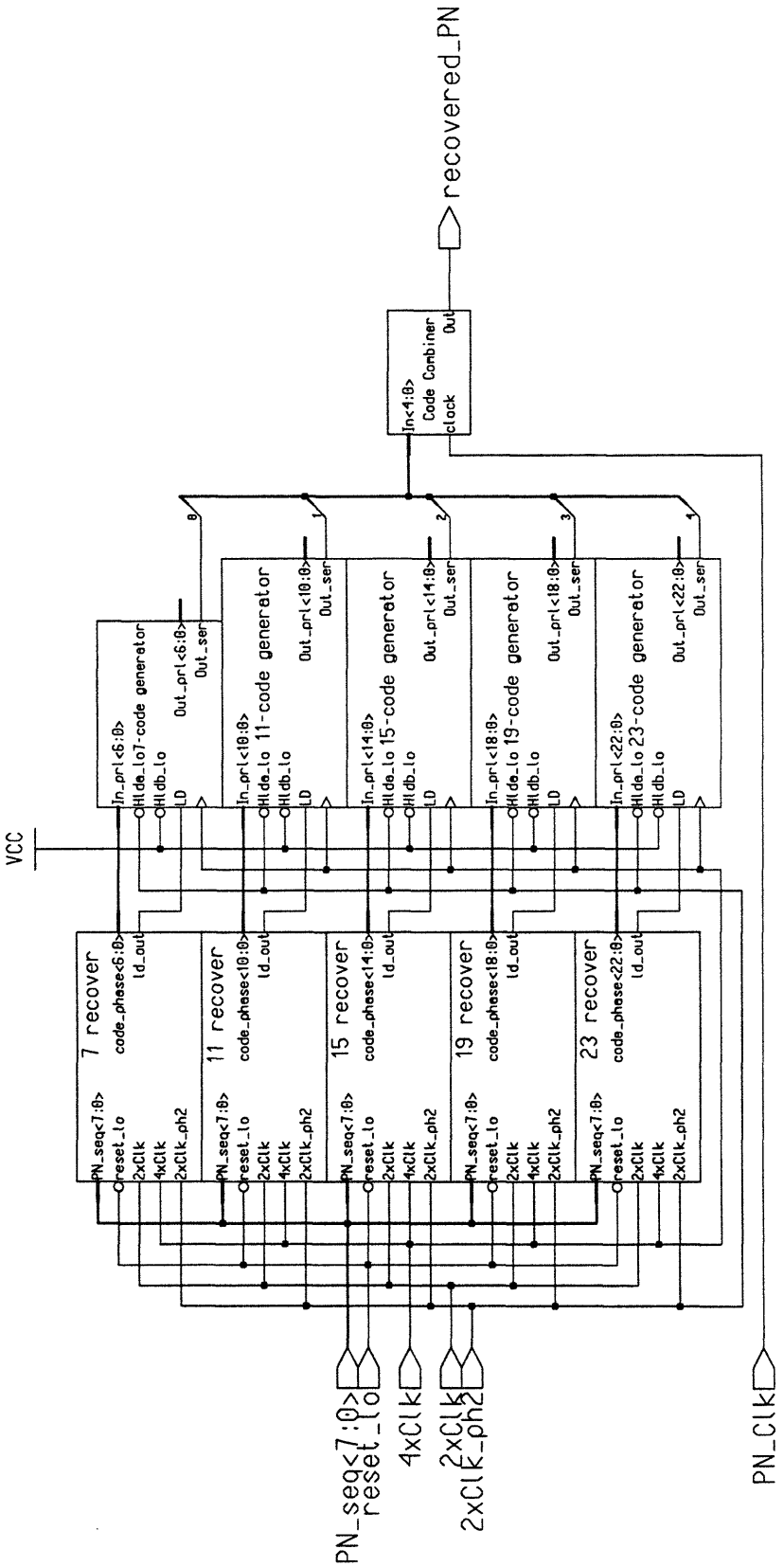

Figure E.40: regenerate schematic

Block Name: test_system

Synopsis:

Input Signals:

4xClk: A clock signal at four times the frequency of the PN clock component.
Hld_lo<4:0>: Signals to allow each component of the PN generator to be held.
Noise<7:0>: The noise to add to the generated PN sequence.
reset_clk_lo: The clock reset signal (negative assertion).
reset_gen: The generator reset signal.
reset_regen_lo: The regenerator reset signal (negative assertion).

Output Signals:

PN_seq<7:0>: The generated PN sequence, with noise.
recovered_PN: The regenerated PN sequence, without noise.

Parameters:

none

Functional Description:

This is the high-level simulation block for the regenerative ranging system. This block generates a PN sequence, adds noise to it, then recovers the PN codes, and outputs a regenerated PN sequence that matches the generated sequence, only without noise.

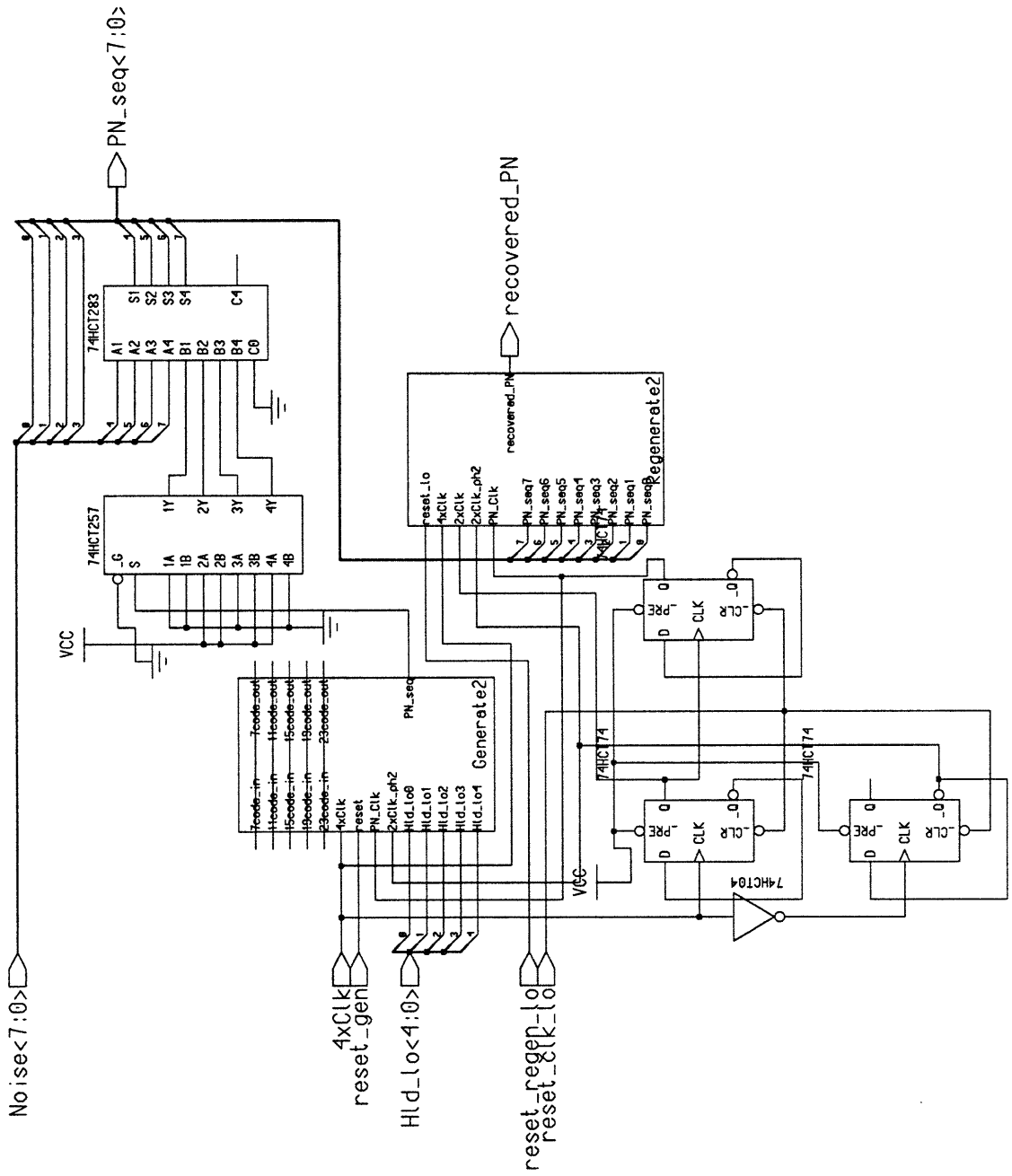

Figure E.41: test_system schematic

Appendix F

Nonstandard Actel Macros

This appendix contains descriptions of each non-standard Actel macro used in the Mentor Graphics designs. The descriptions are formatted in the style of data sheets, with a listing of the inputs, outputs, and method of use for each macro. Each description also contains a figure showing the underlying logic of each macro.

Block Name: 1reg_c

Synopsis:

Input Signals:

_Clr: The asynchronous clear input.
Clk: The clock input.
D: The D input.

Output Signals:

_Q: An inverted version of the registered output.
Q: The registered output.

Parameters:

none

Functional Description:

This macro is a standard rising-edge triggered D-type flip-flop with asynchronous clear, and with both noninverted and inverted outputs.

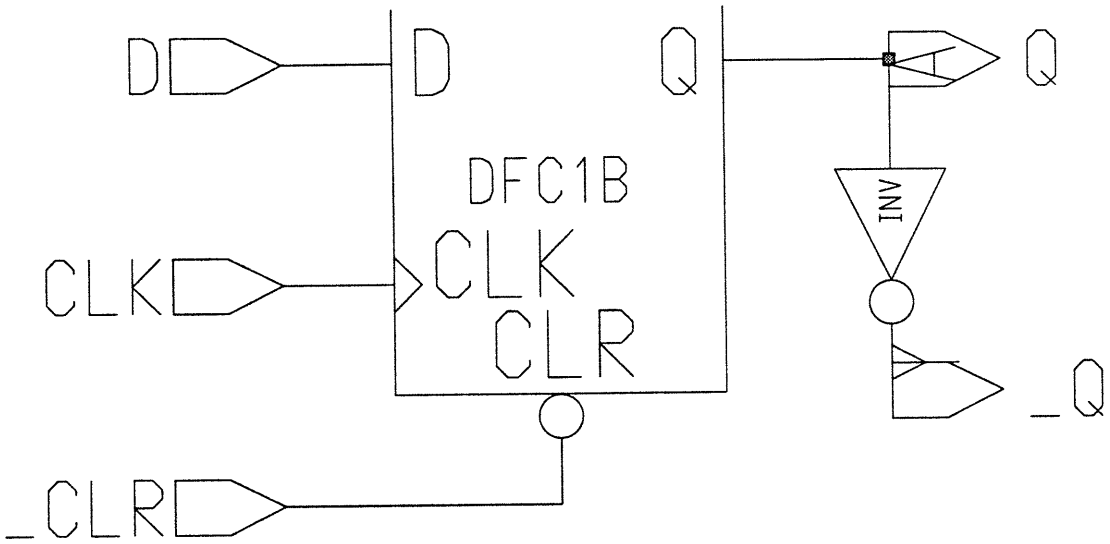

Figure F.1: 1rec_c schematic.

Block Name: 8reg_ec

Synopsis:

Input Signals:

_Clr: The asynchronous clear input.
_Ld: The synchronous load enable.
Clk: The clock input.
D1 - D8: The eight D inputs.

Output Signals:

Q1 - Q8: The eight registered outputs.

Parameters:

none

Functional Description:

This macro is a standard rising-edge triggered octal D-type flip-flop with load enable and asynchronous clear.

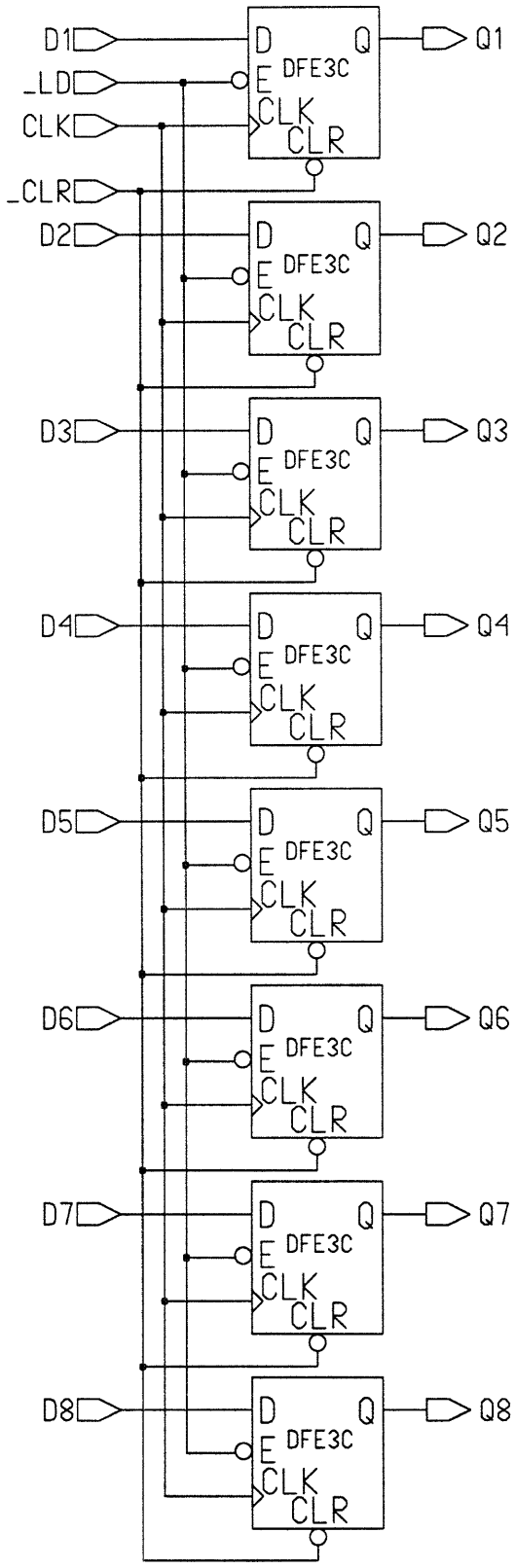

Figure F.2: 8reg_ec schematic.

Block Name: **buf*X*** (*X* is 3,4, 5, 8)

Synopsis:

Input Signals:

In<n:0>: The *X*-bit input. (n=*X*-1)

Output Signals:

Out<n:0>: The *X*-bit buffered output. (n=*X*-1)

Parameters:

none

Functional Description:

This macro is a simple *X*-bit buffer.

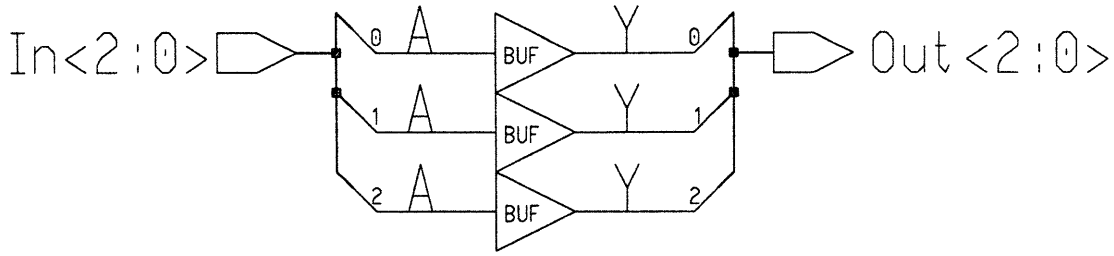

Figure F.3: buf3 schematic.

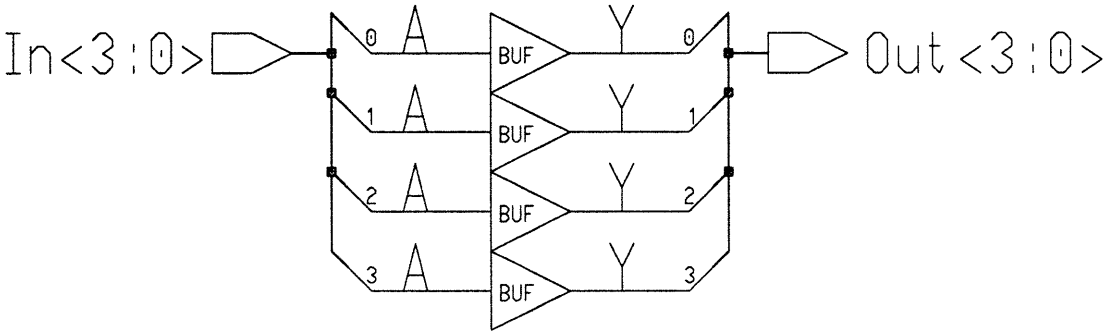

Figure F.4: buf4 schematic.

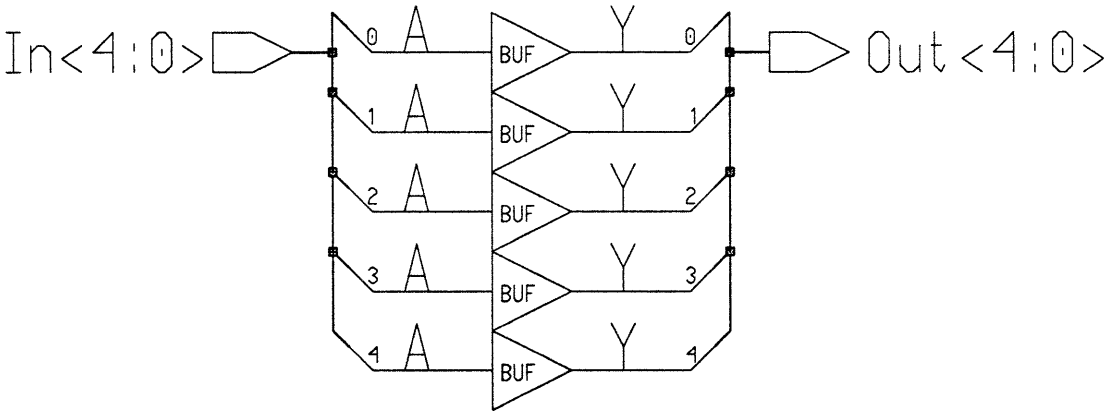

Figure F.5: buf5 schematic.

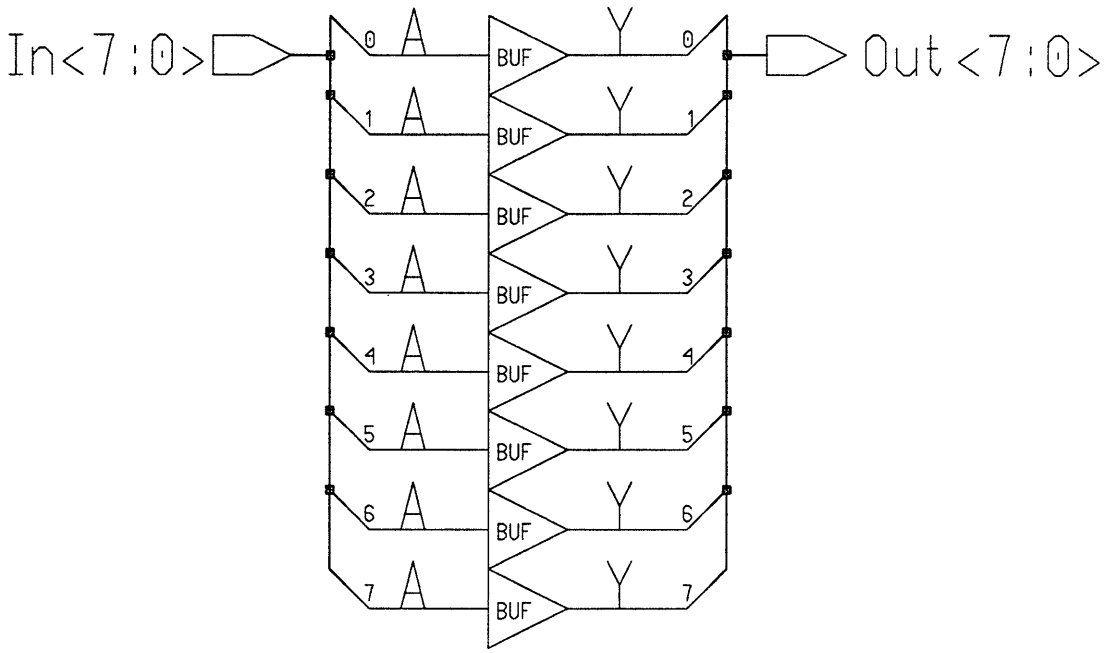

Figure F.6: buf8 schematic.

Block Name: **mux24**

Synopsis:

Input Signals:

S0 - S4: The five-bit select input.
D0 - D23: The 24 input bits.

Output Signals:

Y: The output bit.

Parameters:

none

Functional Description:

This macro is a simple 24-bit multiplexor.

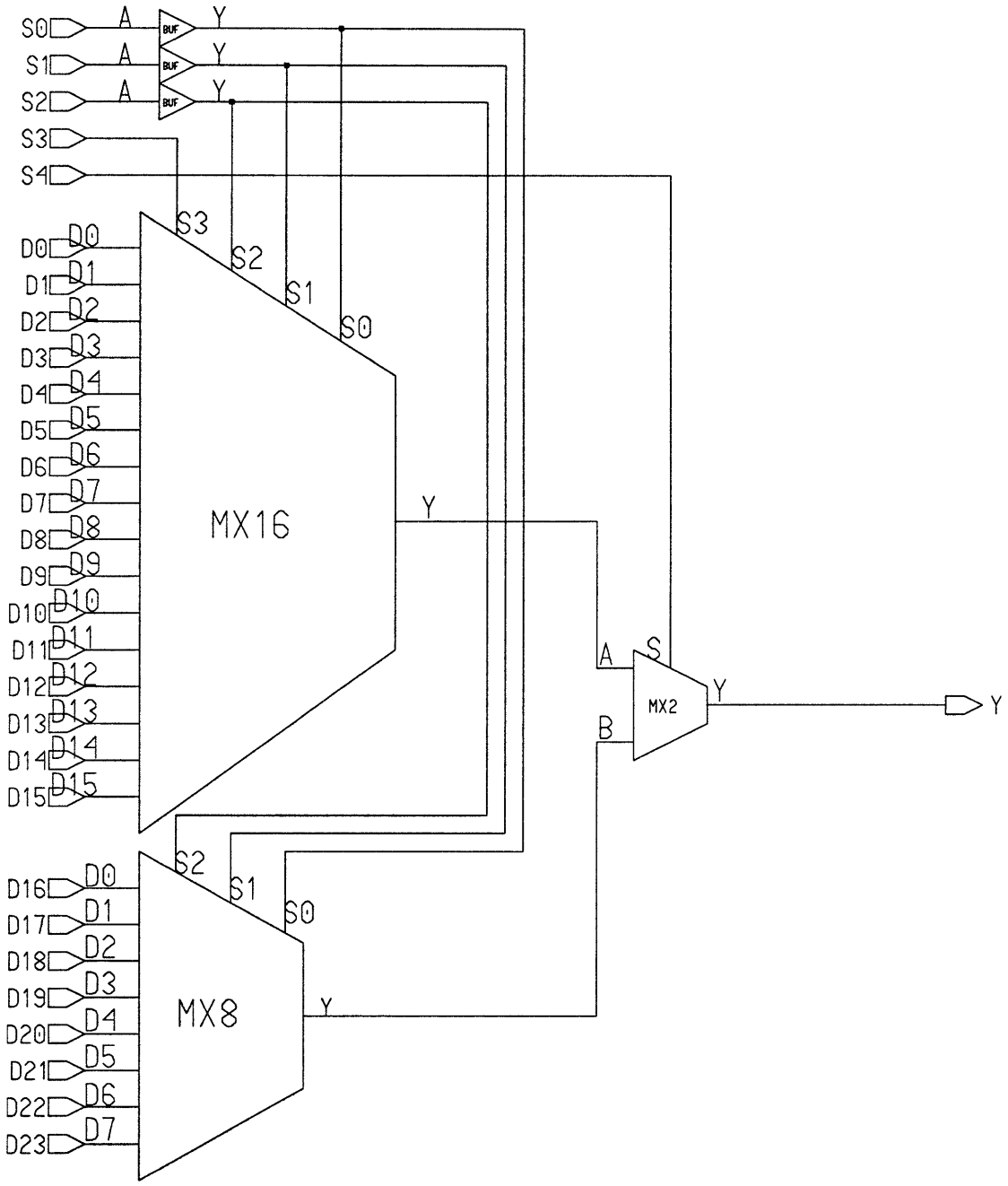

Figure F.7: mux24 schematic.

Block Name: **ttl***X* (*X* is 00, 02, 04, 08, 10, 11, 20, 21, 32, 138, 161, 169, 194, 377)

Synopsis:

Input Signals:

Varied

Output Signals:

Varied

Parameters:

none

Functional Description:

These macros emulate a 74LS*X*. They were entered to create Mentor Graphics symbols that matched the LS library in Mentor.

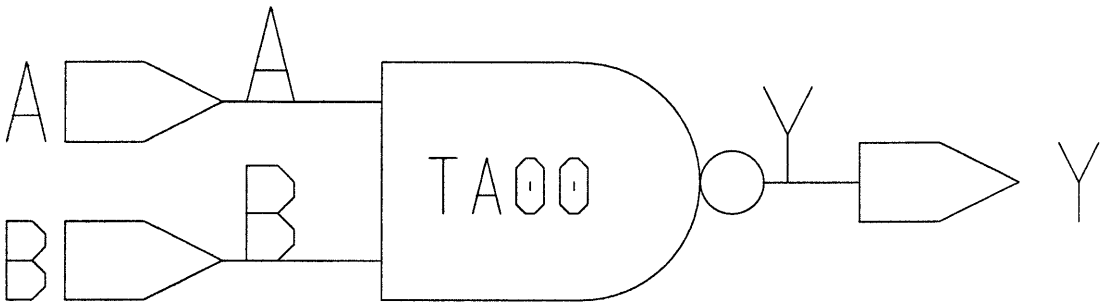

Figure F.8: ttl00 schematic.

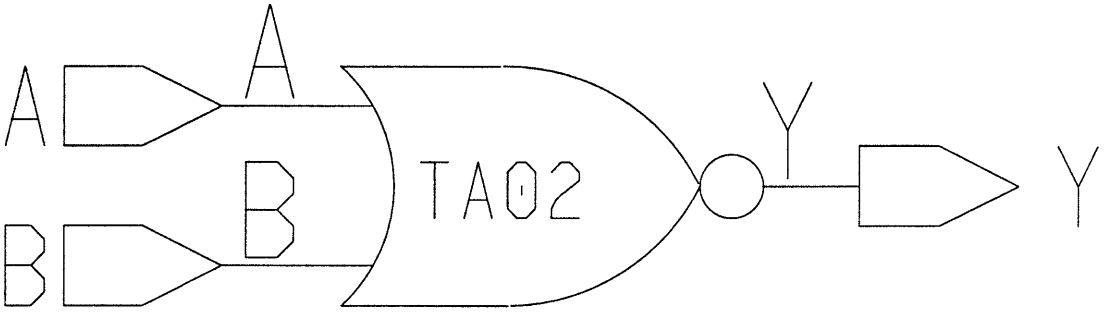

Figure F.9: ttl02 schematic.

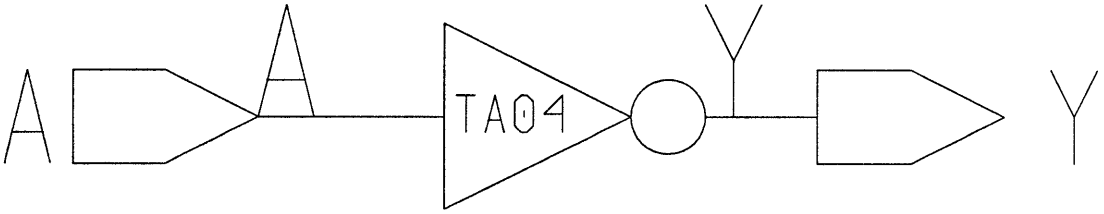

Figure F.10: ttl04 schematic.

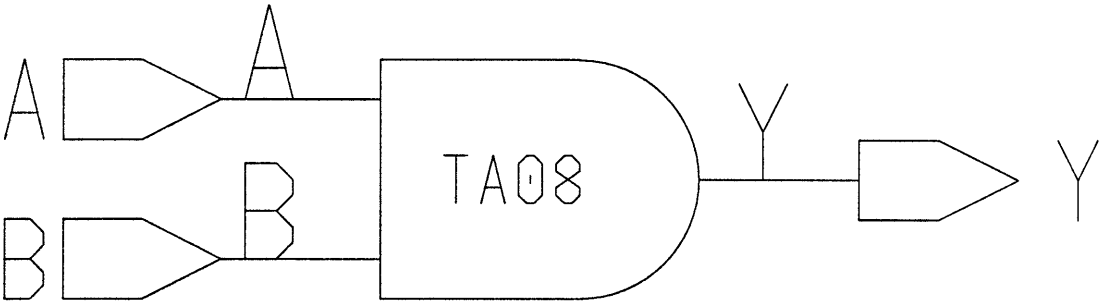

Figure F.11: ttl08 schematic.

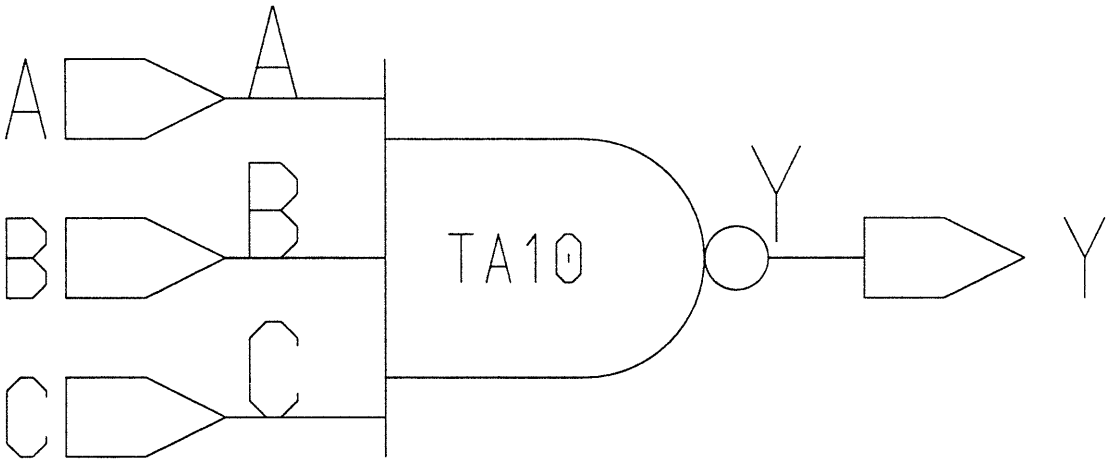

Figure F.12: ttl10 schematic.

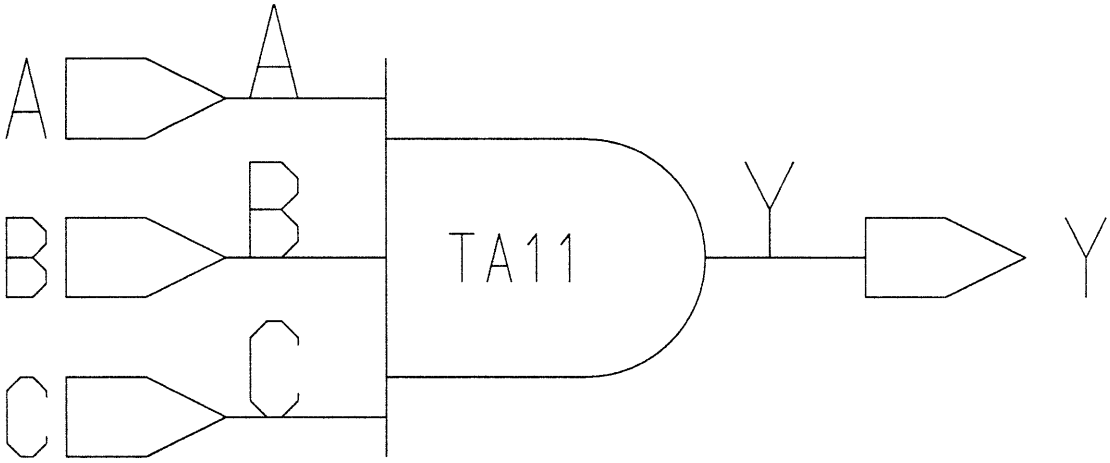

Figure F.13: ttl11 schematic.

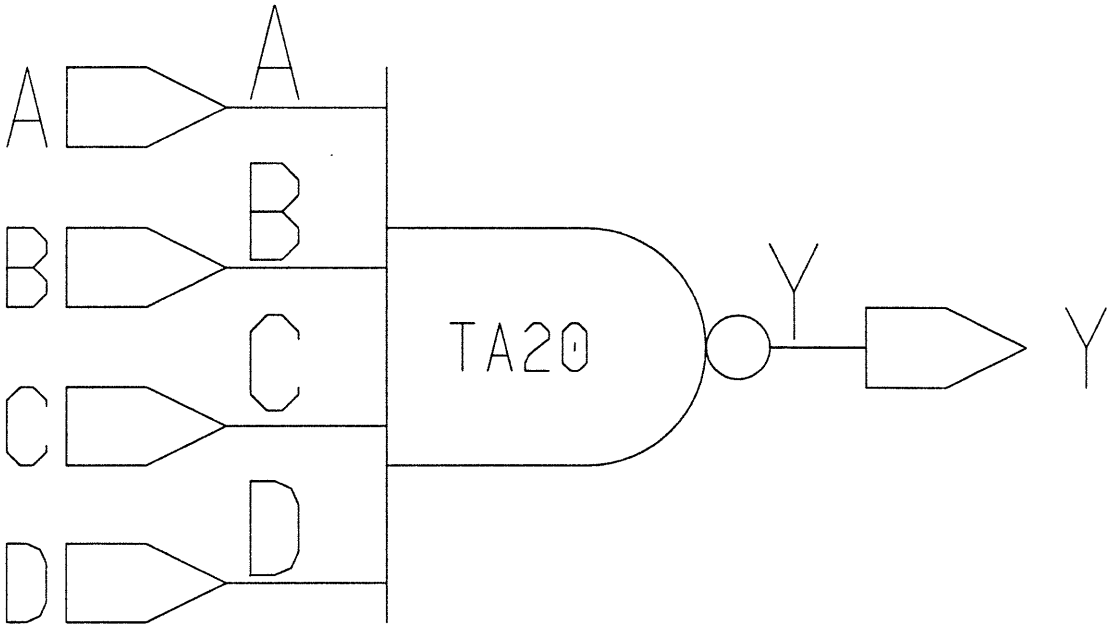

Figure F.14: ttl20 schematic.

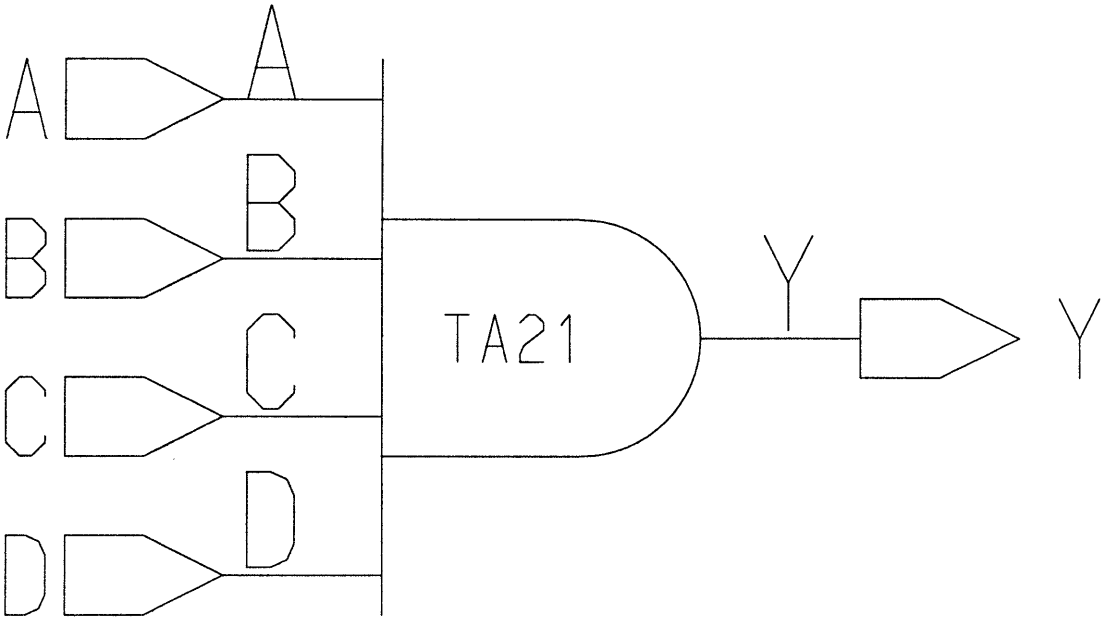

Figure F.15: ttl21 schematic.

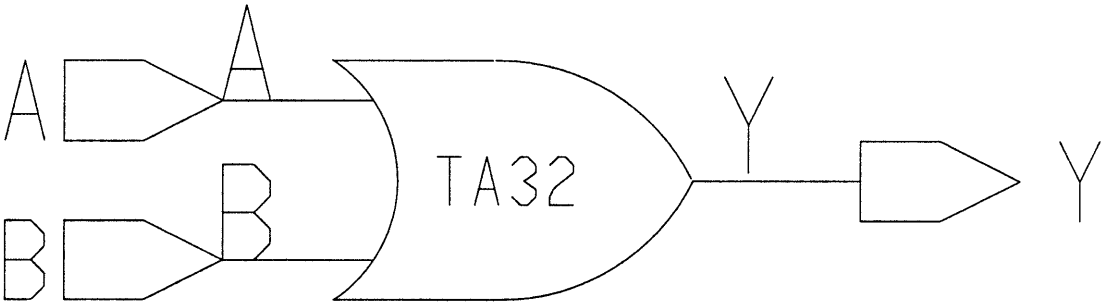

Figure F.16: ttl32 schematic.

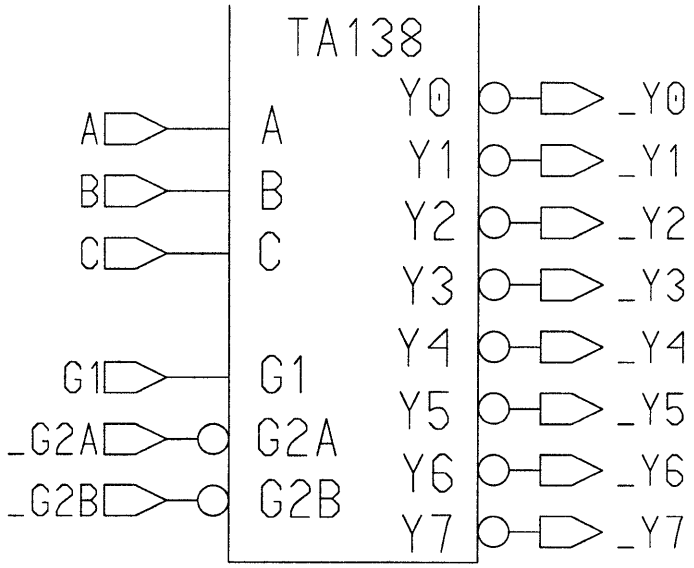

Figure F.17: ttl138 schematic.

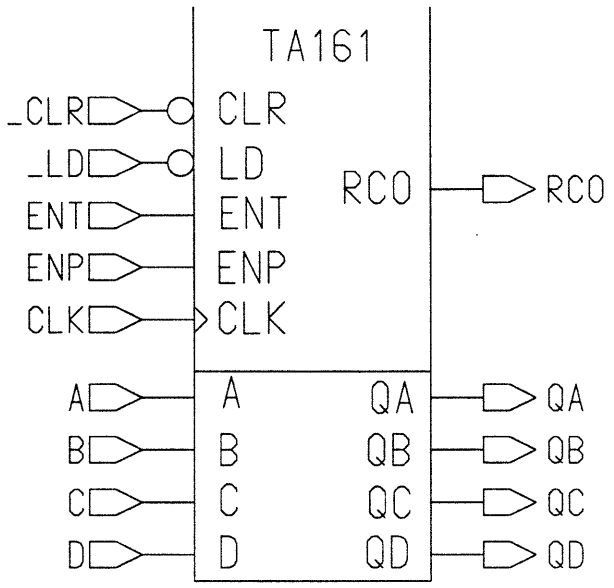

Figure F.18: ttl161 schematic.

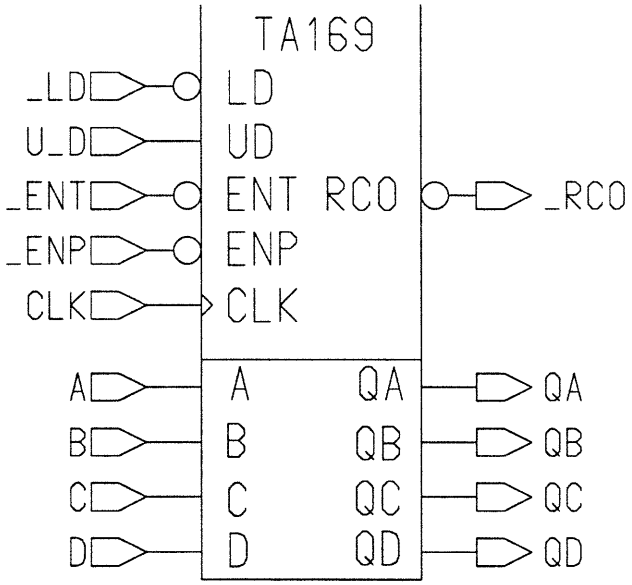

Figure F.19: ttl169 schematic.

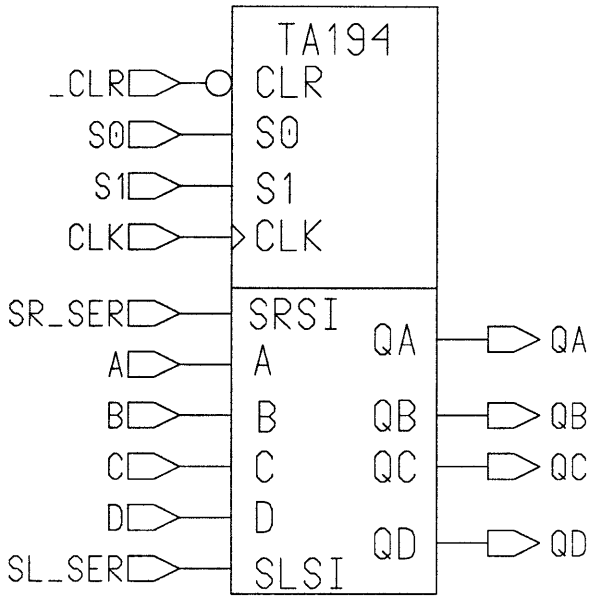

Figure F.20: ttl194 schematic.

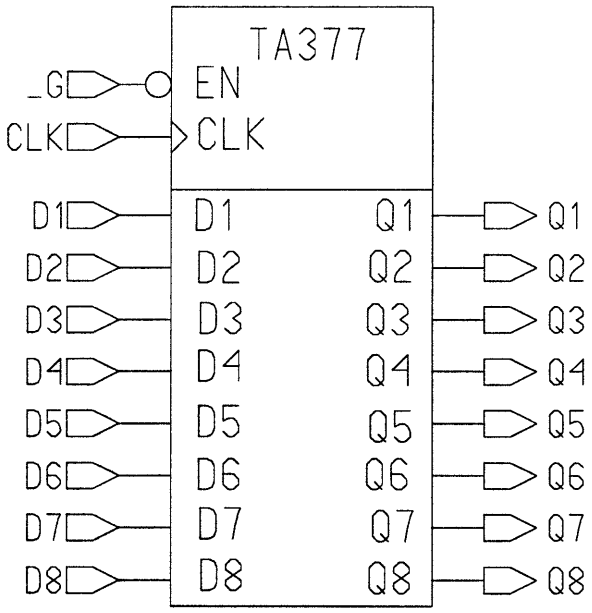

Figure F.21: ttl377 schematic.

Block Name: ttl175

Synopsis:

Input Signals:

_Clr: The asynchronous clear input.
1D-4D: The four D inputs.
Clk: The clock input.

Output Signals:

_1Q - _4Q: The four inverted, registered outputs.
1Q - 4Q: The four registered outputs.

Parameters:

none

Functional Description:

This macro is a rising-edge triggered quad D-type flip-flop with asynchronous clear, and inverted and noninverted outputs.

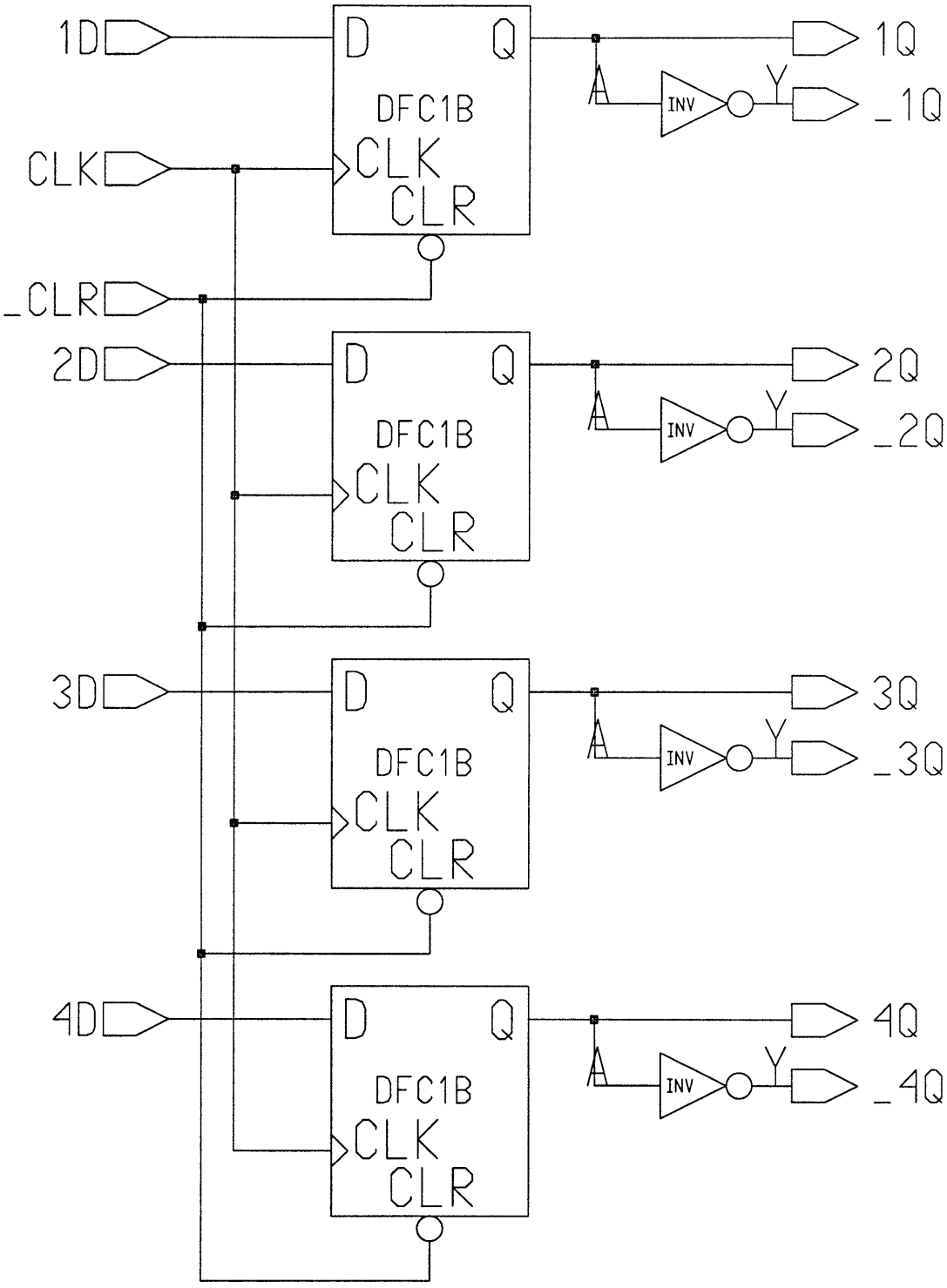

Figure F.22: ttl175 schematic.

Block Name: ttl244

Synopsis:

Input Signals:

_1G, _2G: The two output enables (one for each bank).
1A1 - 1A4: The four bits of input to bank one.
2A1 - 2A4: The four bits of input to bank two.

Output Signals:

1Y1 - 1Y4: The four bits of buffered output from bank one.
2Y1 - 2Y4: The four bits of buffered output from bank two.

Parameters:

none

Functional Description:

This macro is a simple octal tristate buffer. It has two input banks, and two associated output banks. Each bank has an output enable that allows the entire bank to be tristated.

This macro can only be used at the output of a chip.

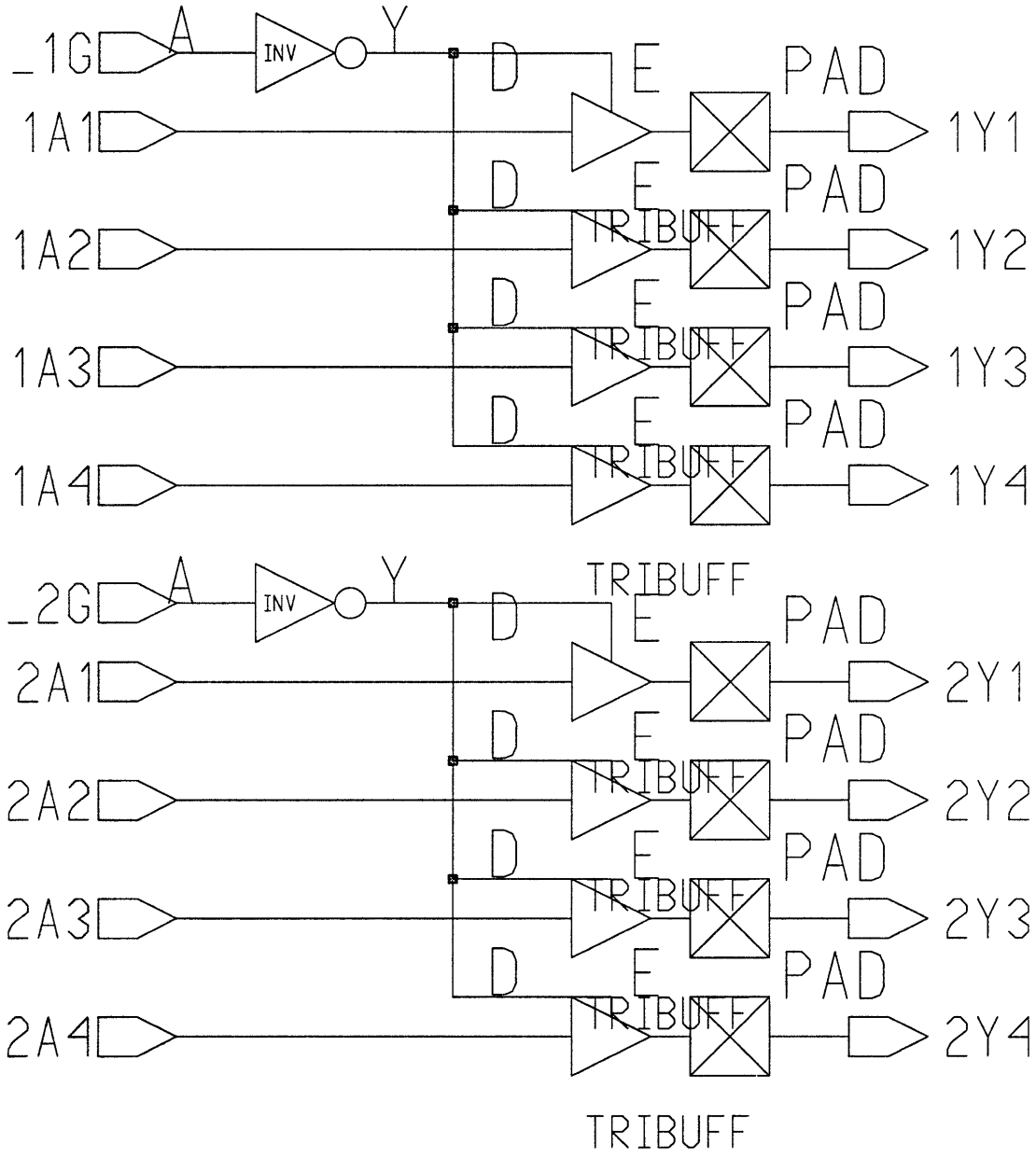

Figure F.23: ttl244 schematic.

Block Name: ttl257

Synopsis:

Input Signals:

1A - 4A: The four A inputs.
1B - 4B: The four B inputs.
S: The select input.

Output Signals:

1Y - 4Y: The four outputs.

Parameters:

none

Functional Description:

This macro is a quad 2-1 multiplexor. The select input decides which bank of inputs will be pass through to the outputs. If S is low, the A inputs are passed through, if S is high, the B inputs are passed through.

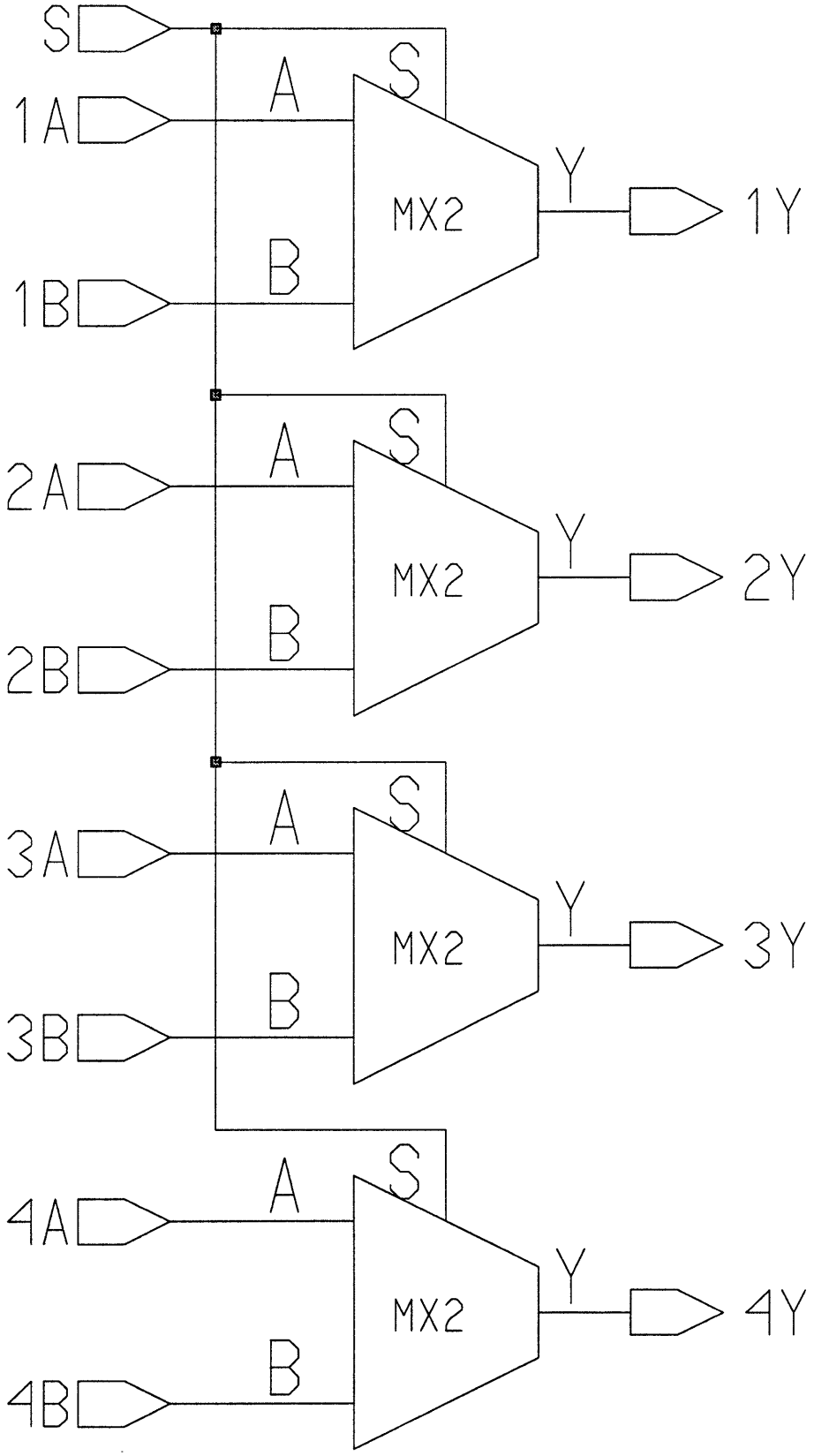

Figure F.24: ttl257 schematic.

Block Name: ttl283

Synopsis:

Input Signals:

A1 - A4: The four-bit A input.
B1 - B4: The four-bit B input.
C0: The input carry bit.

Output Signals:

C4: The carry output.
S1 - S4: The four-bit sum output.

Parameters:

none

Functional Description:

This macro is a four-bit adder. The output, **S**, is equal to the sum of the inputs, **A+B**, plus the carry bit, **C0**. For **A**, **B**, and **S**, bit 4 is the MSB.

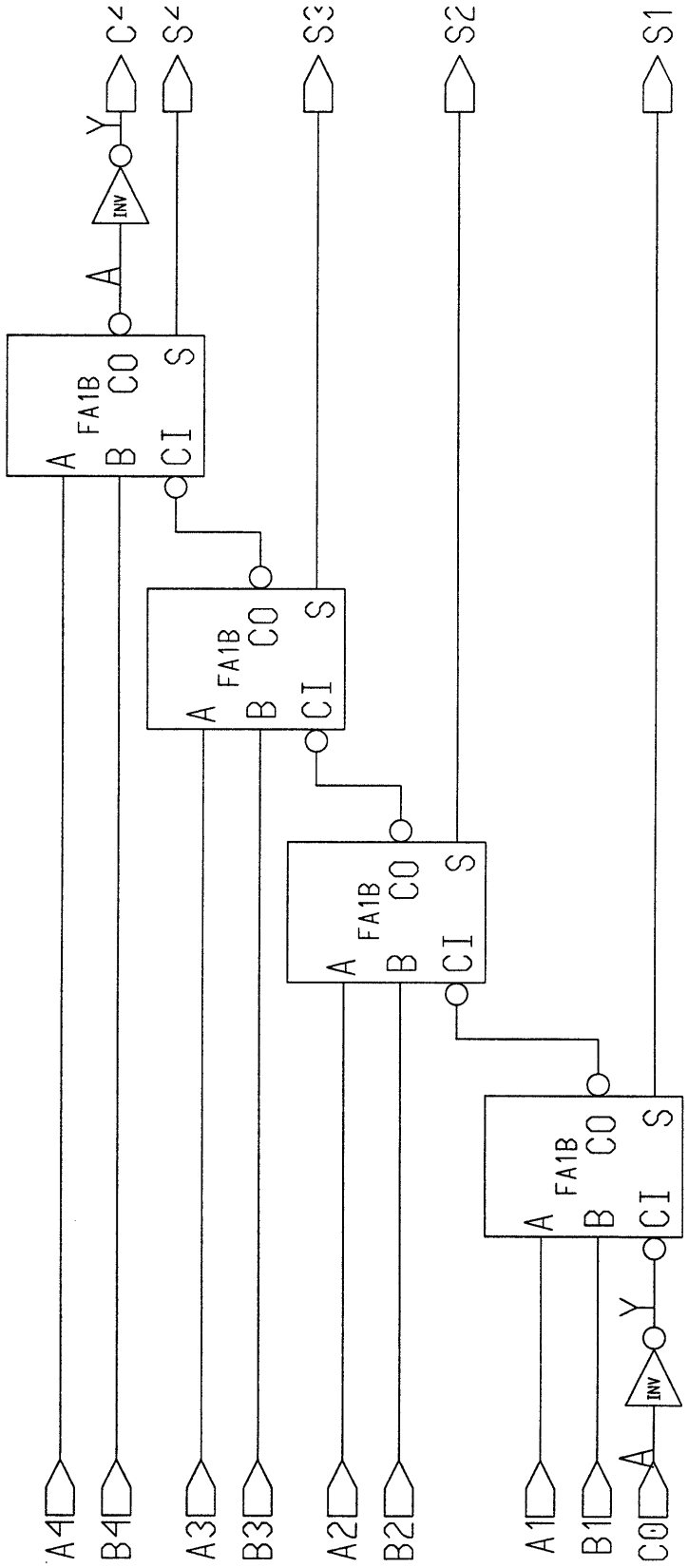

Figure F.25: ttl283 schematic.

Appendix G

Revised Mentor Graphics designs for Actel FPGAs

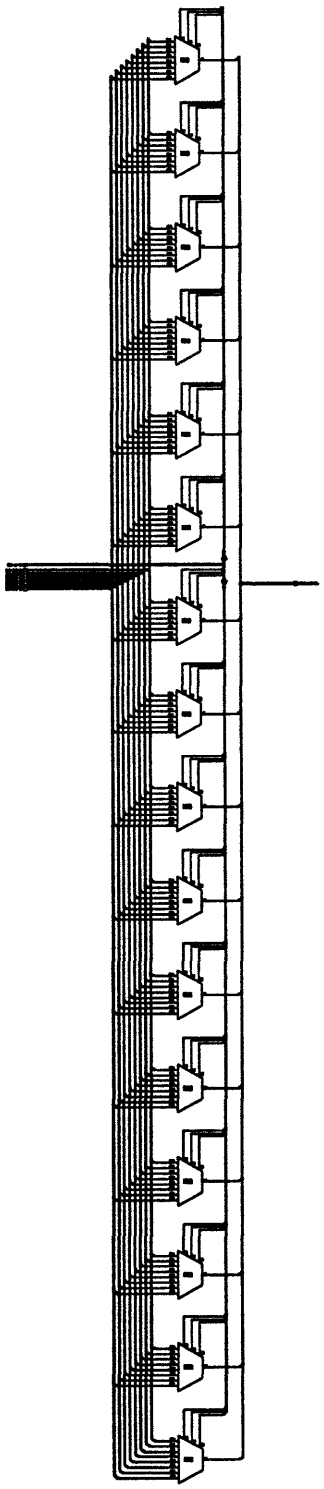

Figure G.1: 11mux_1 schematic

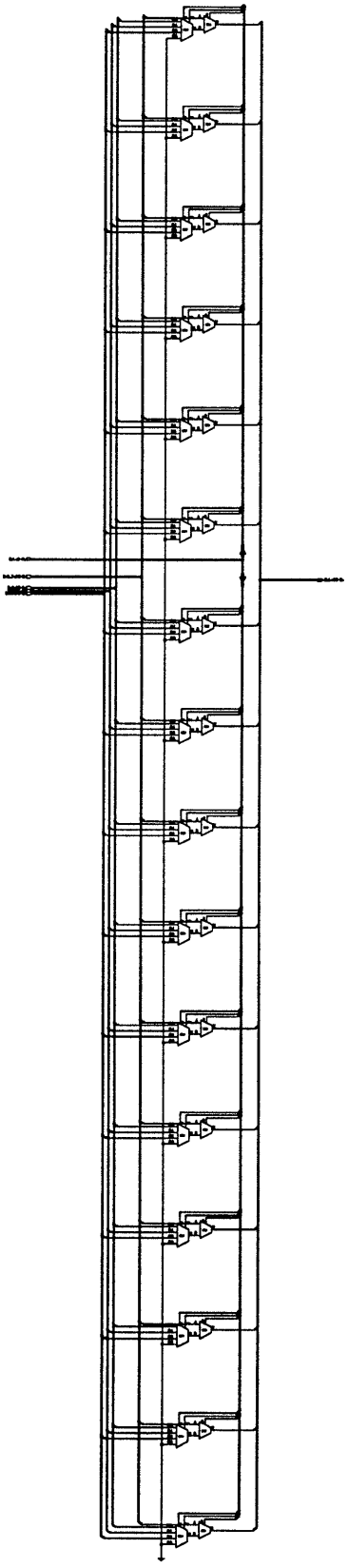

Figure G.2: 11mux_2 schematic

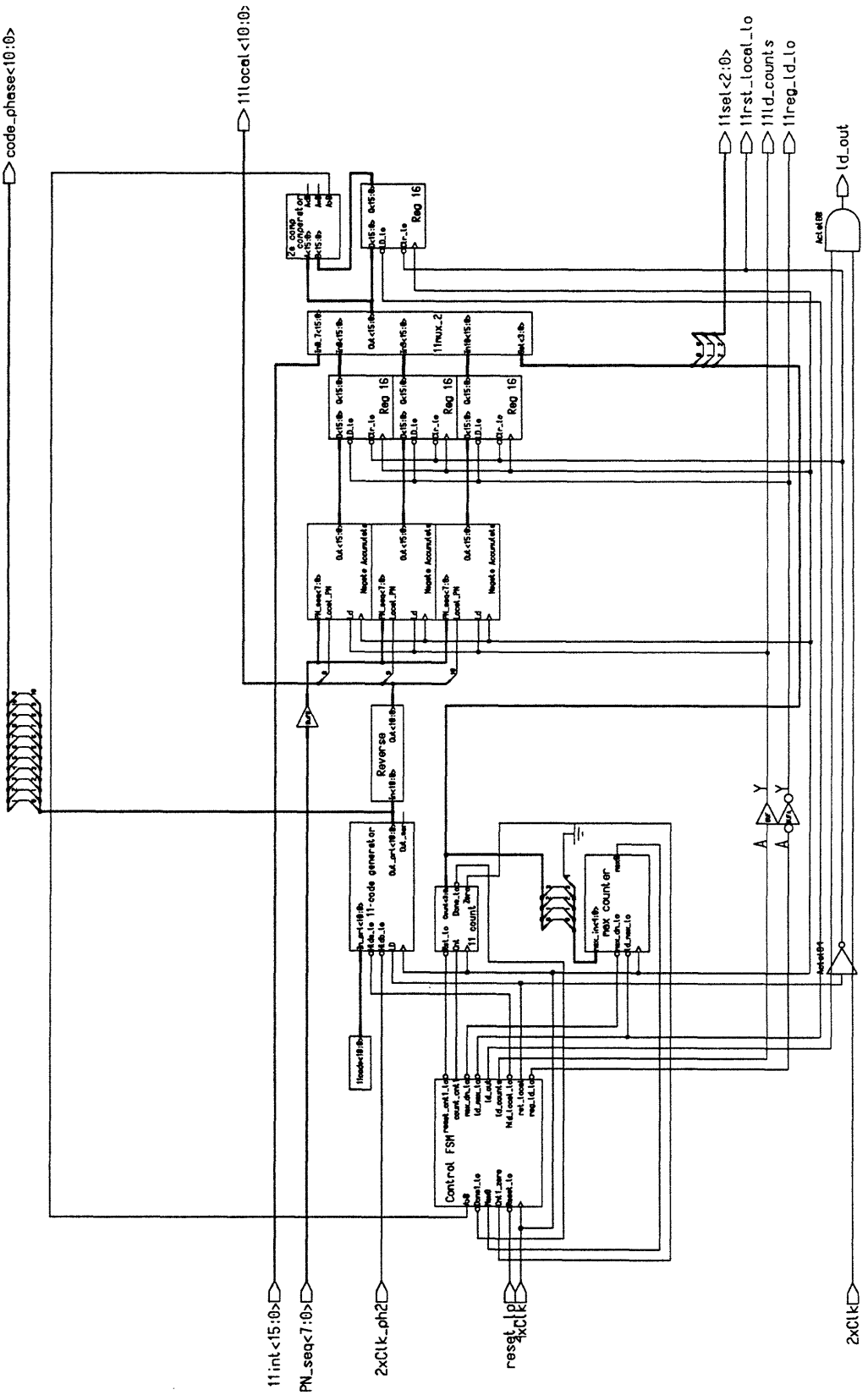

Figure G.3: 11rec_1 schematic

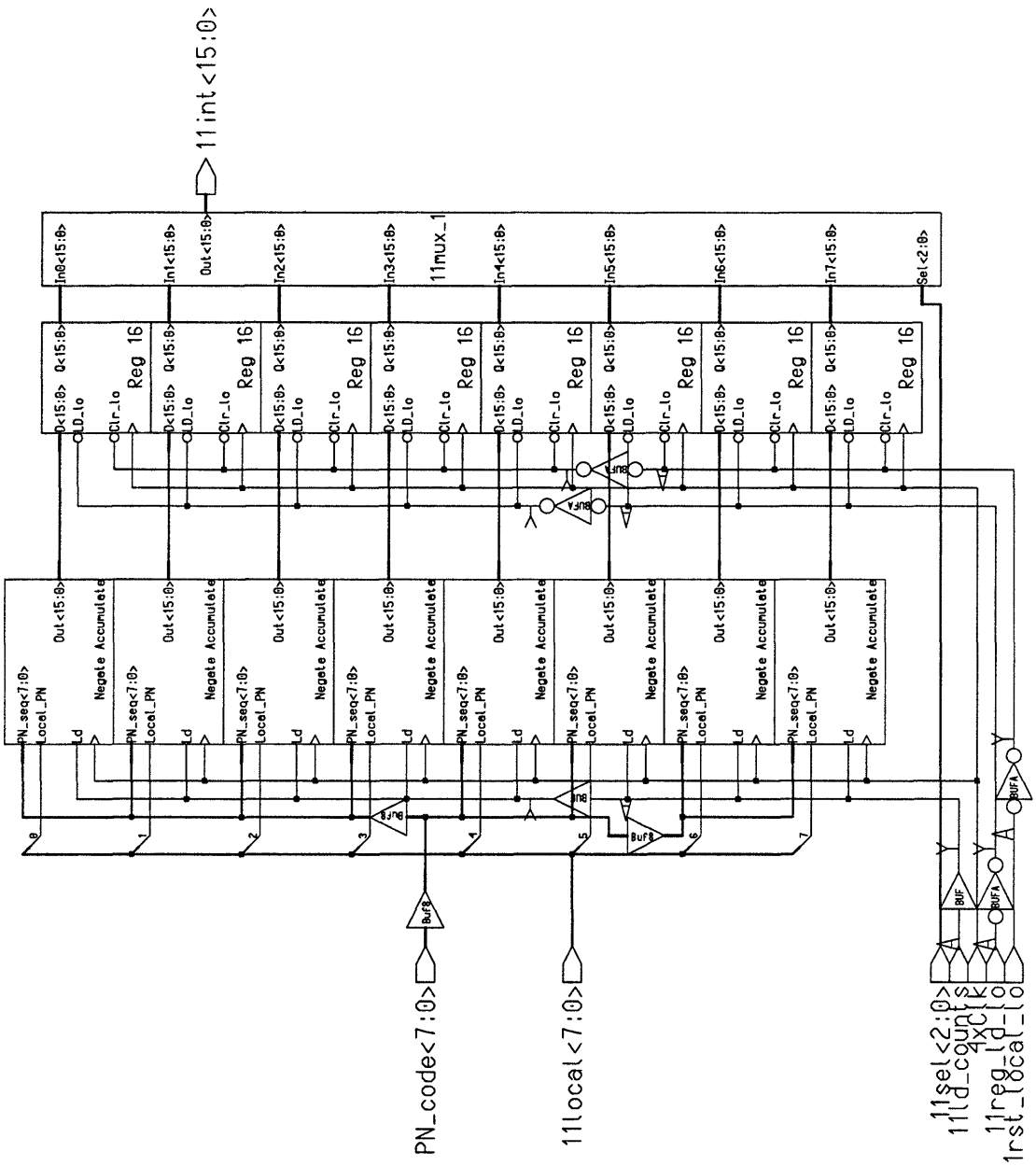

Figure G.4: 11rec_2 schematic

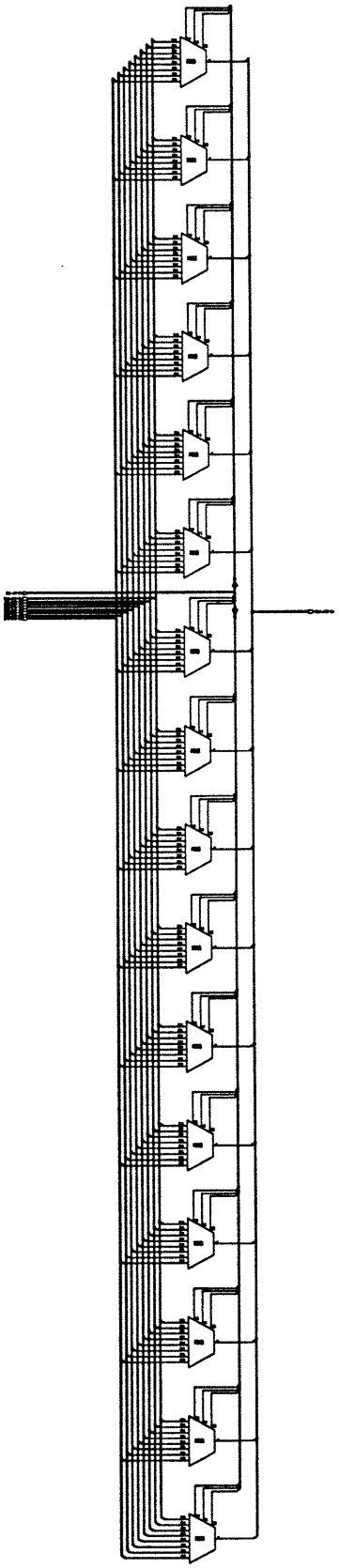

Figure G.5: 15mux_1 schematic

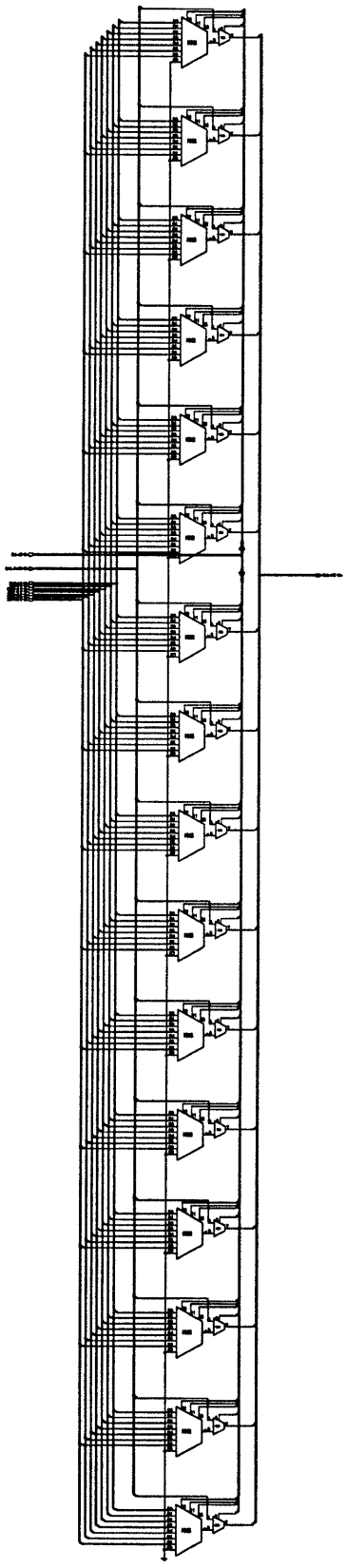

Figure G.6: 15mux_2 schematic

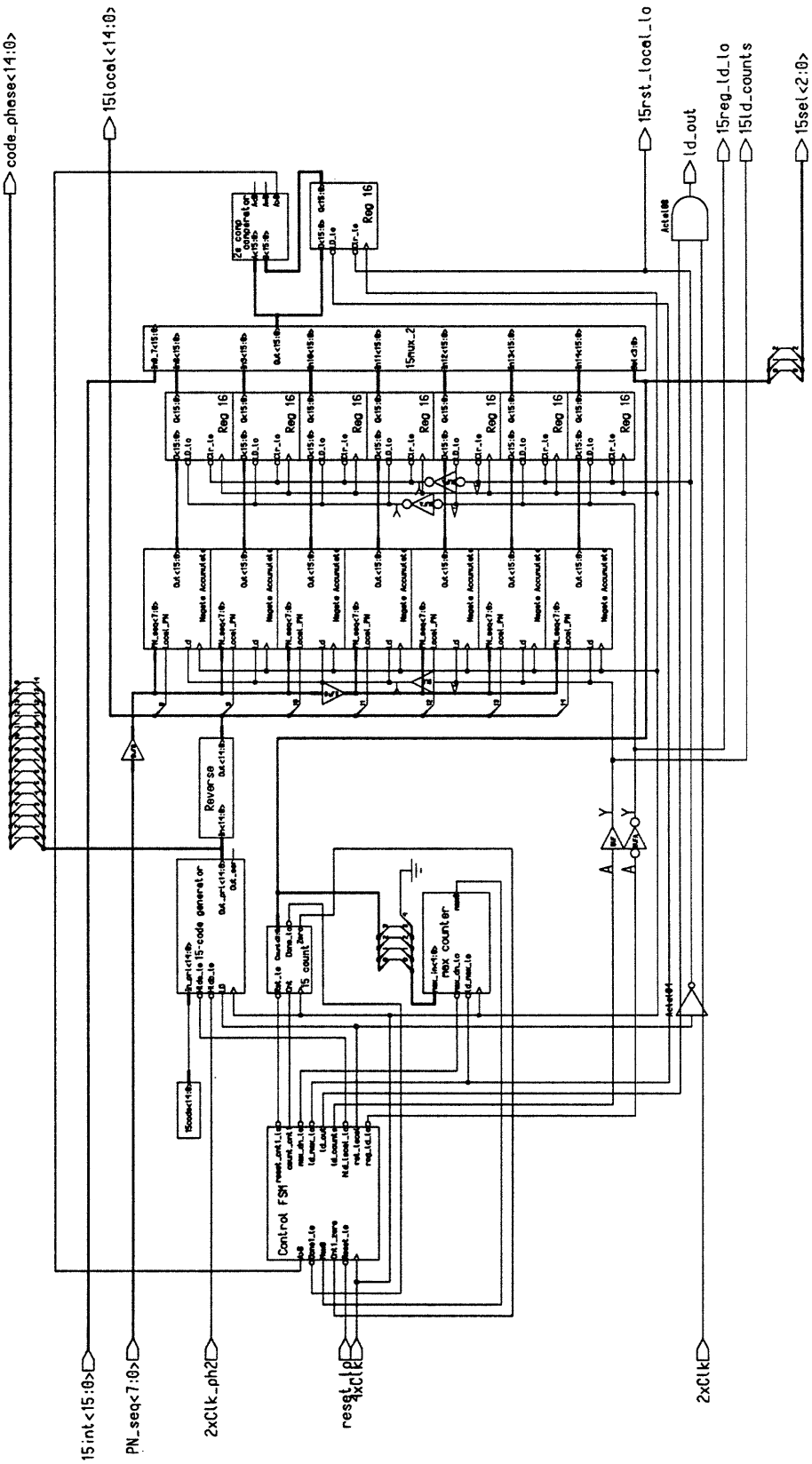

Figure G.7: 15rec_1 schematic

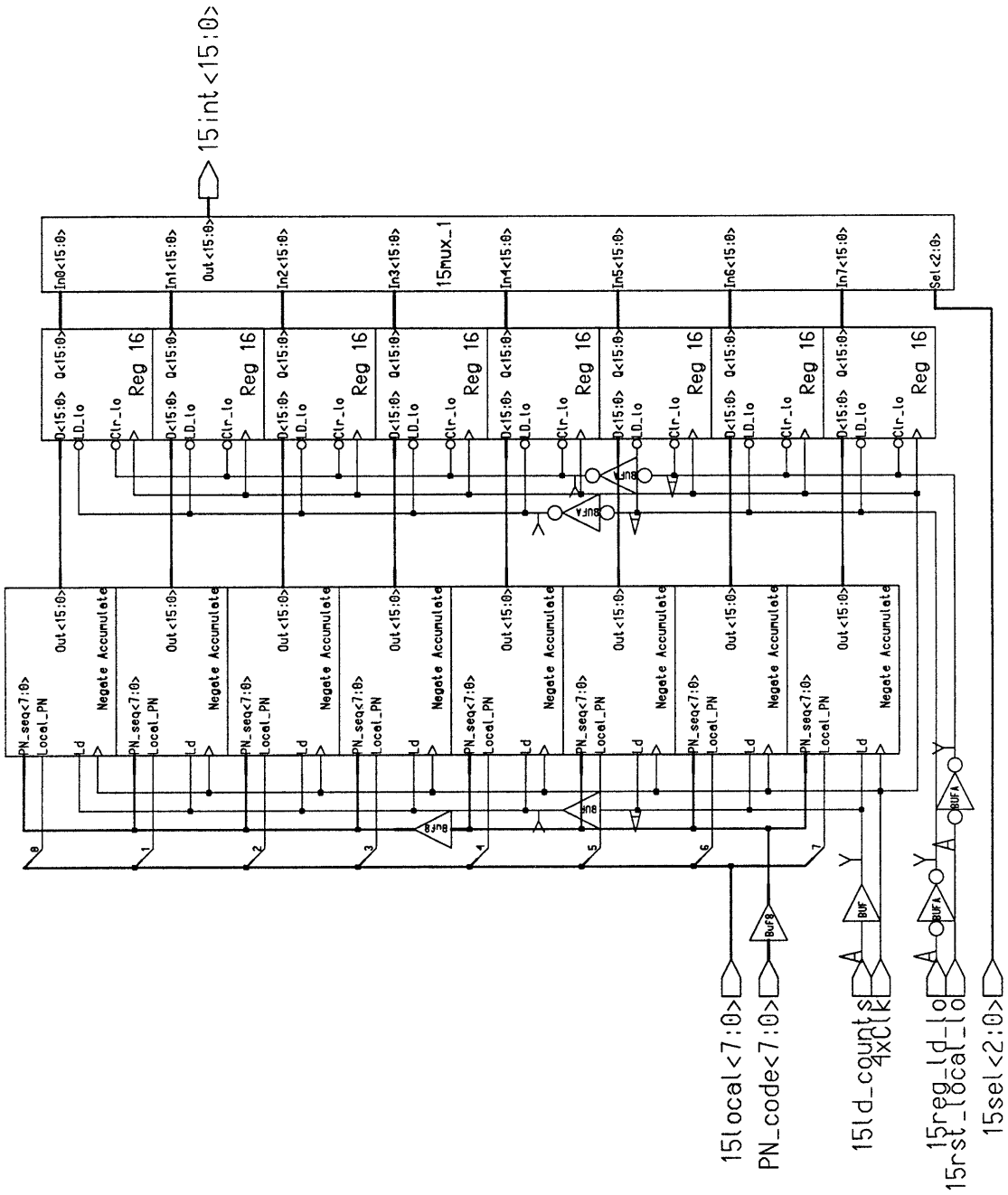

Figure G.8: 15rec_2 schematic

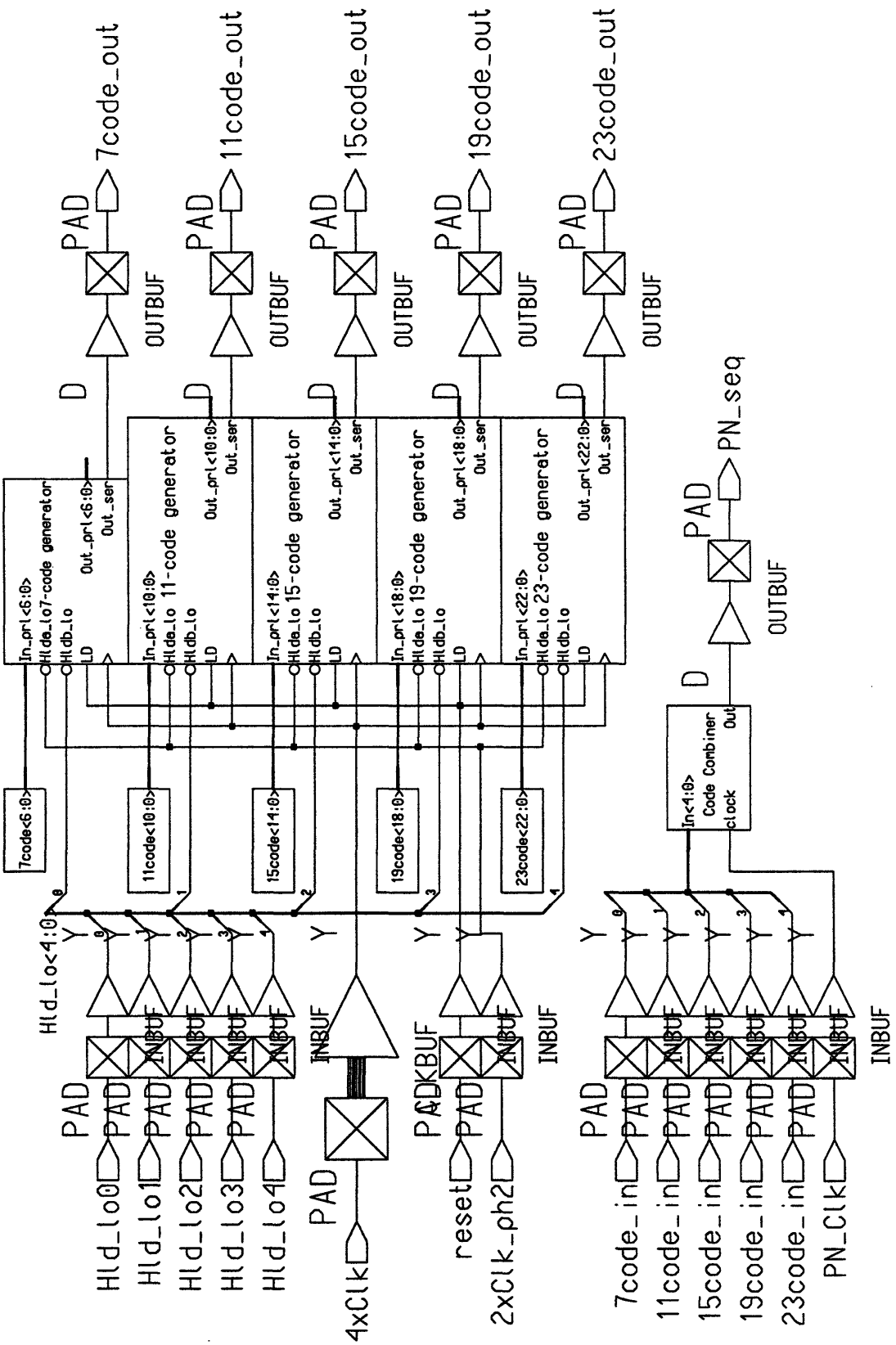

Figure G.9: generate2 schematic

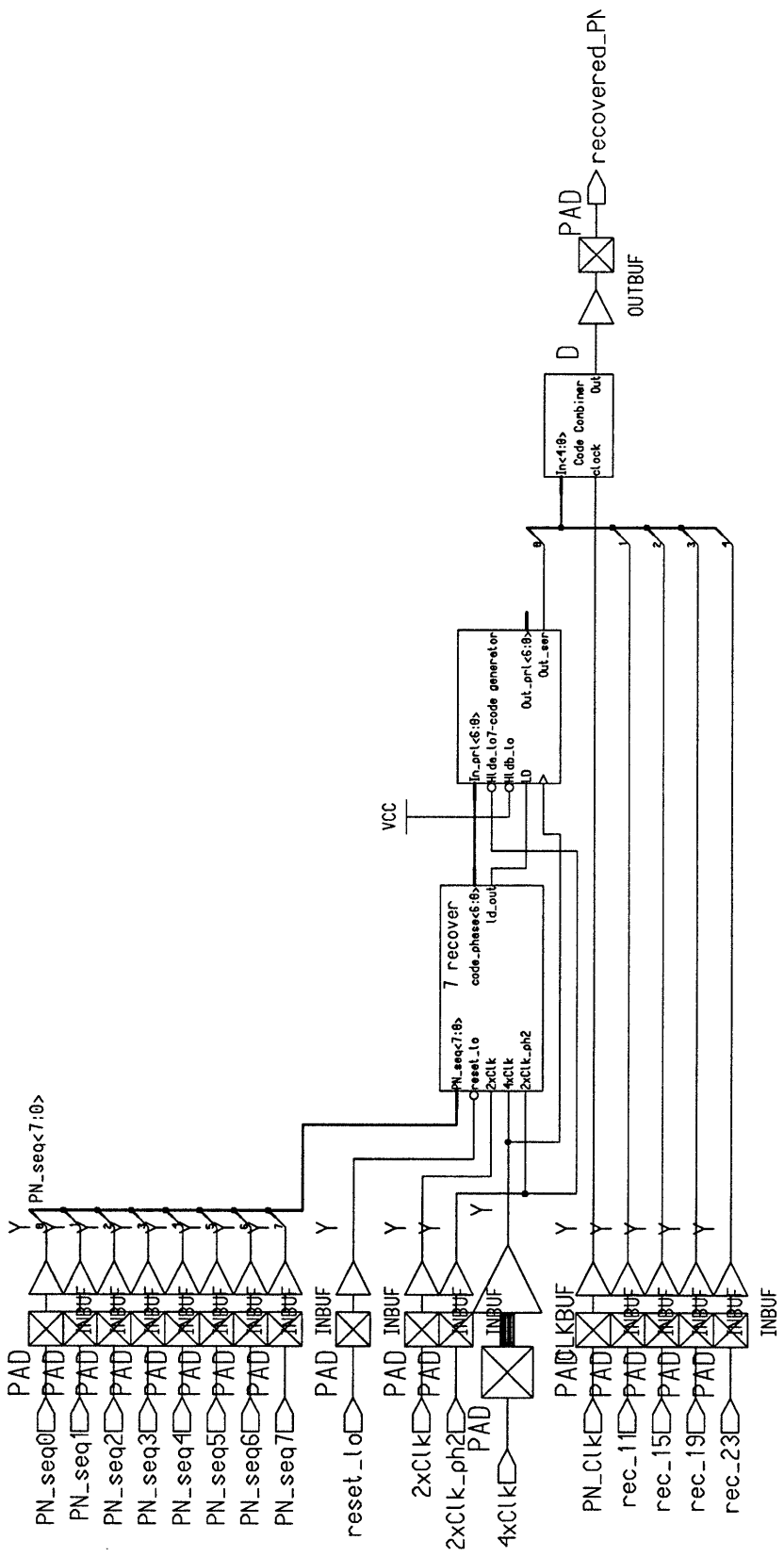

Figure G.10: regen1 schematic

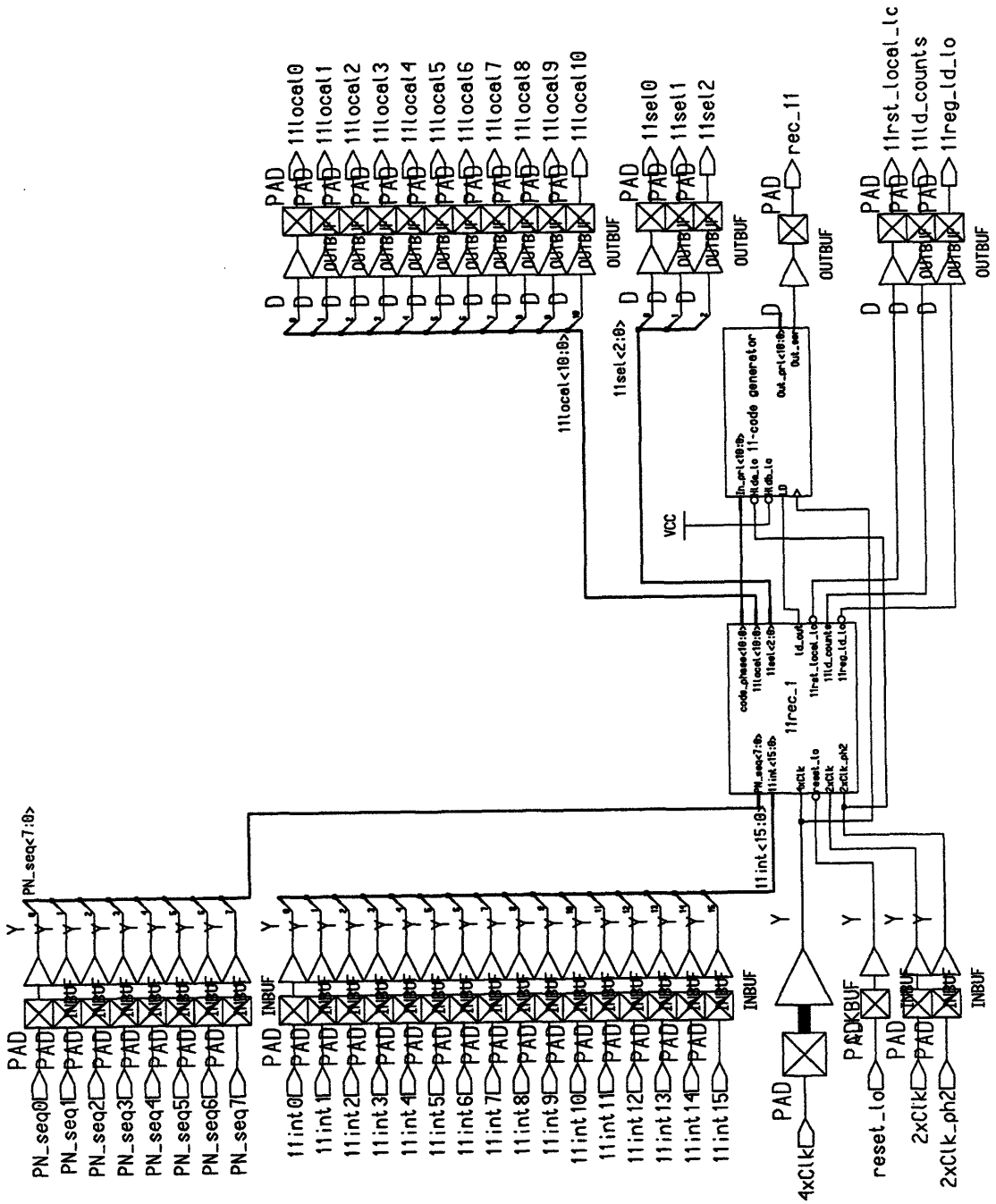

Figure G.11: regen2 schematic

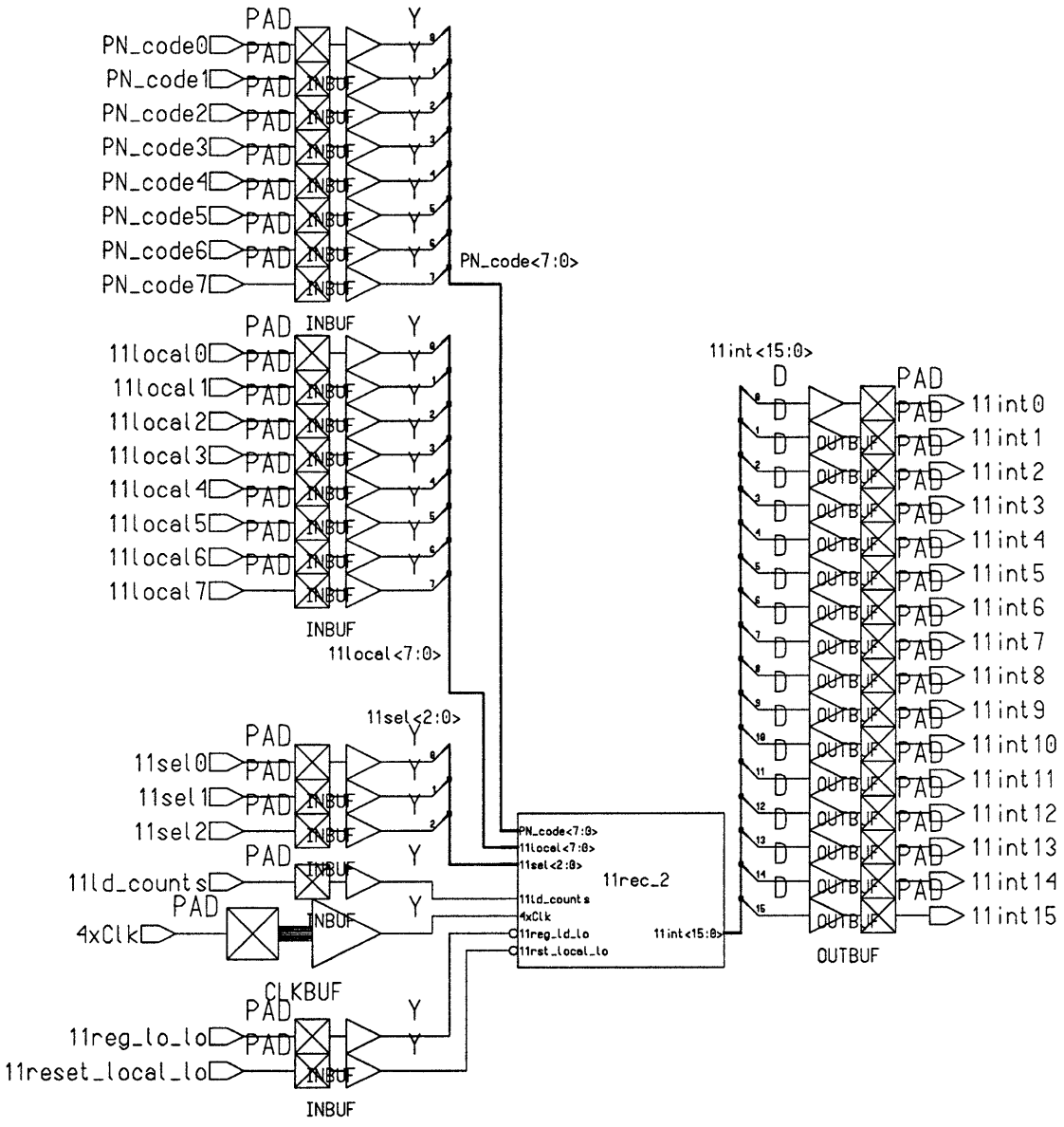

Figure G.12: regen3 schematic

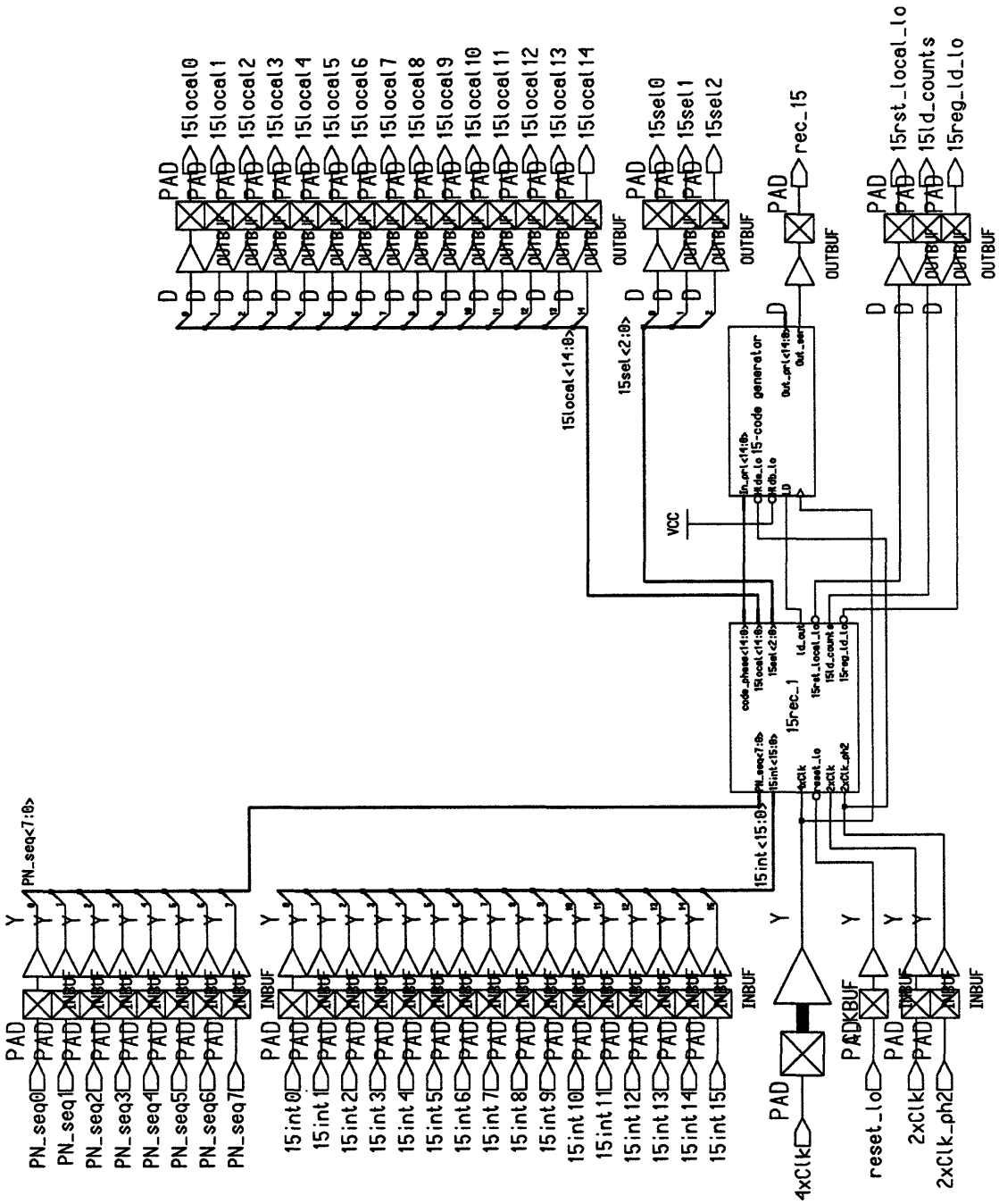

Figure G.13: regen4 schematic

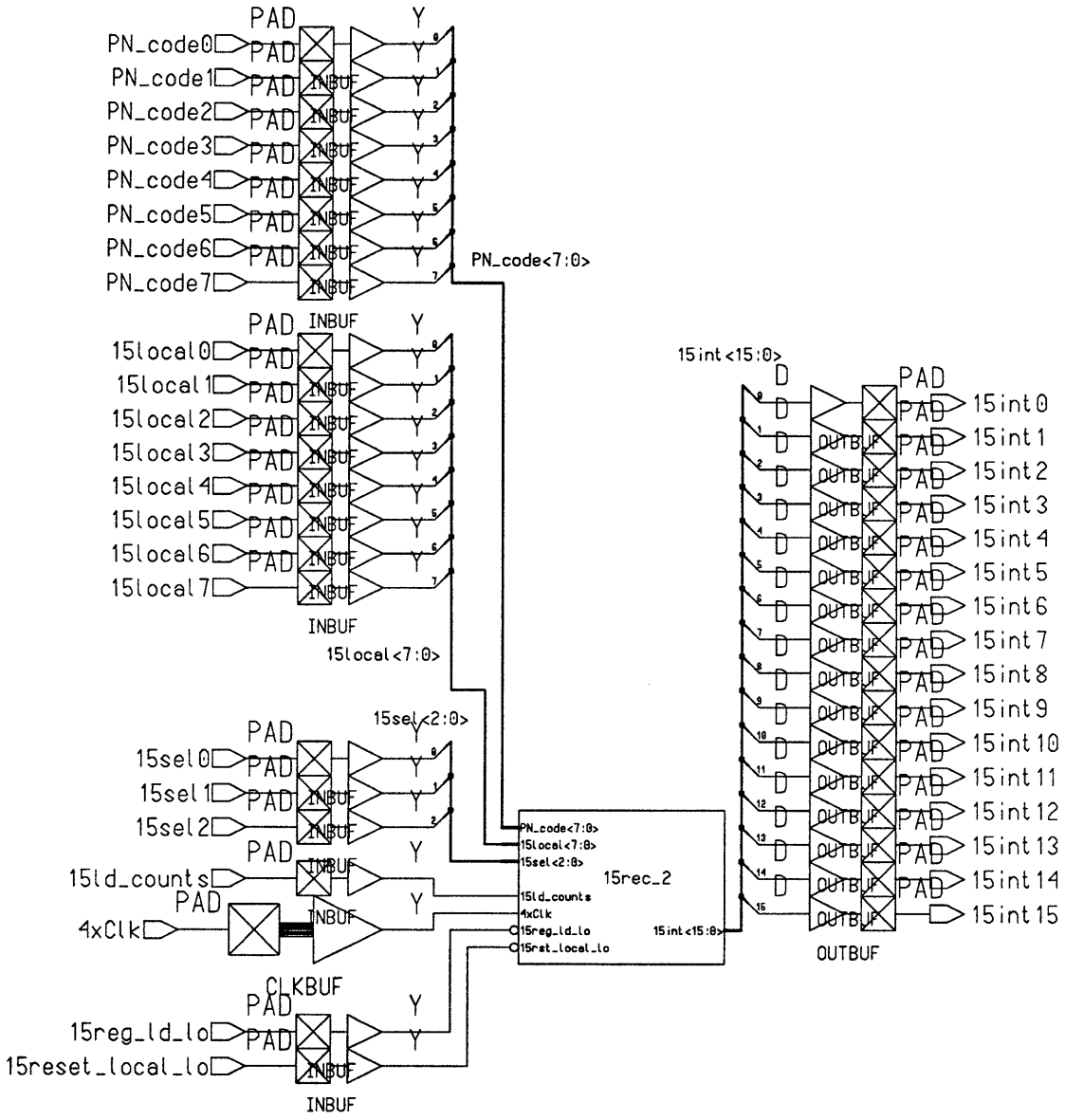

Figure G.14: regen5 schematic

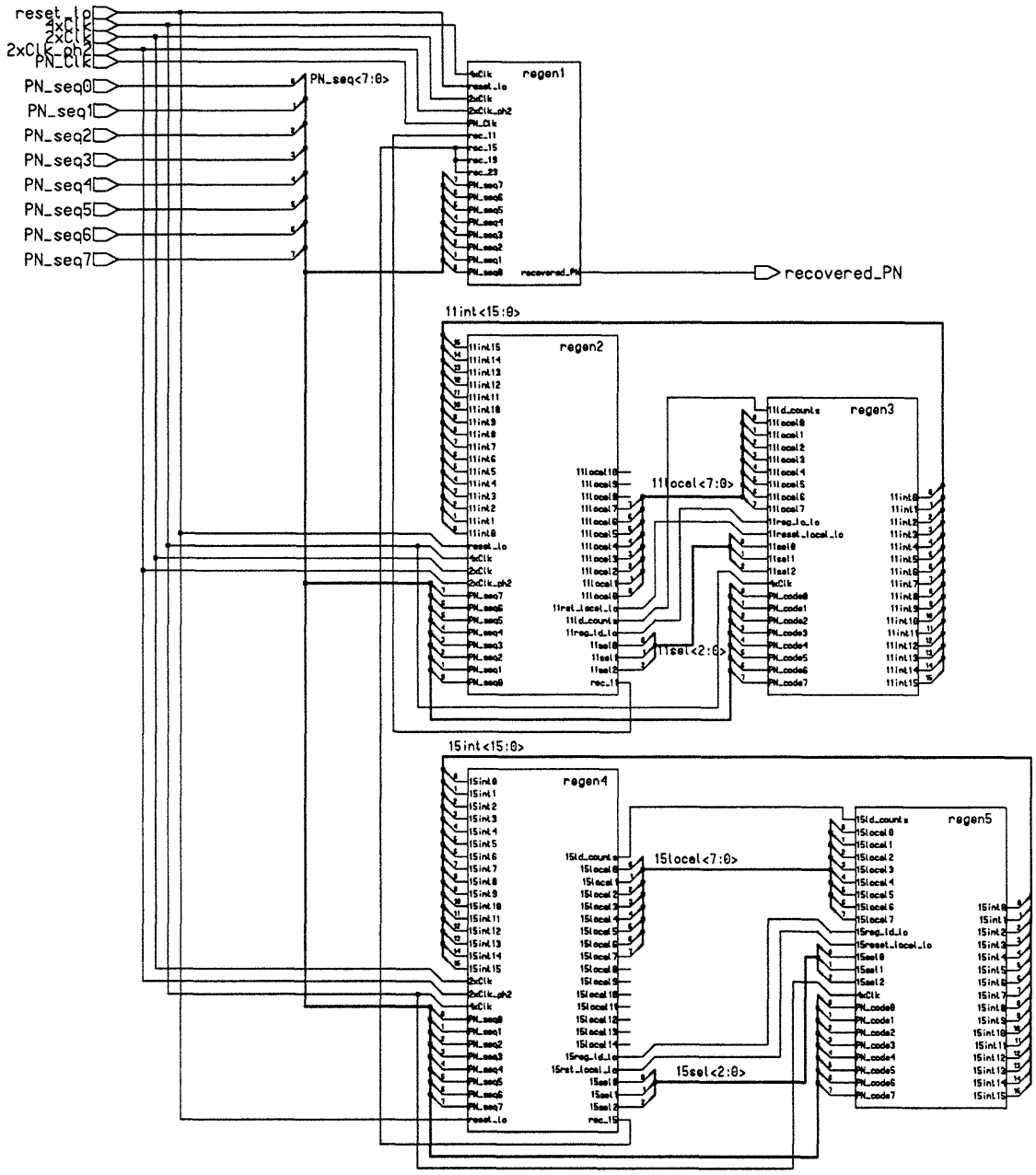

Figure G.15: regenerate2 schematic

Appendix H

Test Vector Generation Code

H.1 C Code Used to Generate Test Vectors

```
#include <stdio.h>
#include <sys/time.h>

void main(int argc, char *argv[]);
void usage();
void noise(int noise_ceiling, int time, int decimal);

void main(int argc, char *argv[])
{
  int i,j,d7,d11,d15,d19,d23;
  struct timeval time;
  int noise_ceiling, end_time=-1;

  if ((argc > 3) || (argc < 2))
    usage();

  if (sscanf(argv[1], "%d", &noise_ceiling) == -1)
    usage();
  if (argc == 3)
    if (sscanf(argv[2], "%d", &end_time) == -1)
      usage();

  if (end_time < 300000) end_time = 300000;

  gettimeofday(&time,(struct timezone *)0);
  srandom(time.tv_usec ^ time.tv_sec ^ getpid());

  d7 = (int) (random() % 7);
  d11 = (int) (random() % 11);
  d15 = (int) (random() % 15);
  d19 = (int) (random() % 19);
  d23 = (int) (random() % 23);

  for (i = 750 ; (d7+d11+d15+d19+d23) > -5 ; i += 125)
    {
      if (d7 == 0)
         printf("FORCe /Hld_lo<0> 1 %d.5\n",(i-63));
      if (d11 == 0)
         printf("FORCe /Hld_lo<1> 1 %d.5\n",(i-63));
```

```
      if (d15 == 0)
         printf("FORCe /Hld_lo<2> 1 %d.5\n",(i-63));
      if (d19 == 0)
         printf("FORCe /Hld_lo<3> 1 %d.5\n",(i-63));
      if (d23 == 0)
         printf("FORCe /Hld_lo<4> 1 %d.5\n",(i-63));

      noise(noise_ceiling, i, 0);
      printf("FORCe /4xClk 1  %d.0 -Abs\n",i);
      noise(noise_ceiling, (i+62), 5);
      printf("FORCe /4xClk 0  %d.5 -Abs\n",(i+62));

      if (d7 >= 0) d7--;
      if (d11 >= 0) d11--;
      if (d15 >= 0) d15--;
      if (d19 >= 0) d19--;
      if (d23 >= 0) d23--;
    }

  for (j = i ; j < i+250 ; j += 125)
    {
      noise(noise_ceiling, j, 0);
      printf("FORCe /4xClk 1  %d.0 -Abs\n",j);
      noise(noise_ceiling, (j+62), 5);
      printf("FORCe /4xClk 0  %d.5 -Abs\n",(j+62));
    }
  printf("FORCe /reset_regen_lo 0  %d.5 -Abs\n",(j-63));

  for (i = j ; i < j+250 ; i += 125)
    {
      noise(noise_ceiling, i, 0);
      printf("FORCe /4xClk 1  %d.0 -Abs\n",i);
      noise(noise_ceiling, (i+62), 5);
      printf("FORCe /4xClk 0  %d.5 -Abs\n",(i+62));
    }

  printf("FORCe /reset_regen_lo 1  %d.5 -Abs\n",(i-63));

  for (j = i ; j < end_time+1 ; j +=125)
    {
      noise(noise_ceiling, j, 0);
      printf("FORCe /4xClk 1  %d.0 -Abs\n",j);
      noise(noise_ceiling, (j+62), 5);
      printf("FORCe /4xClk 0  %d.5 -Abs\n",(j+62));
    }
}
```

```
void usage()
{
  printf("Usage: force_gen noise_ceiling [end_time]\n");
  printf("  where noise_ceiling and end_time are integers\n");
  exit(2);
}

void noise(int noise_ceiling, int time, int decimal)
{
  int noise_dec = (int) (random() % noise_ceiling);
  int flip_sign = (int) (random() & 1);
  int bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8;

  if (flip_sign)
    {
      noise_dec--;
      bit1 = ((noise_dec & 1) == 0);
      bit2 = ((noise_dec & 2) == 0);
      bit3 = ((noise_dec & 4) == 0);
      bit4 = ((noise_dec & 8) == 0);
      bit5 = ((noise_dec & 16) == 0);
      bit6 = ((noise_dec & 32) == 0);
      bit7 = ((noise_dec & 64) == 0);
      bit8 = ((noise_dec & 128) == 0);
    }
  else
    {
      bit1 = ((noise_dec & 1) != 0);
      bit2 = ((noise_dec & 2) != 0);
      bit3 = ((noise_dec & 4) != 0);
      bit4 = ((noise_dec & 8) != 0);
      bit5 = ((noise_dec & 16) != 0);
      bit6 = ((noise_dec & 32) != 0);
      bit7 = ((noise_dec & 64) != 0);
      bit8 = ((noise_dec & 128) != 0);
    }

  printf("FORCe /Noise<0> %d %d.%d -Abs\n", bit1, time, decimal);
  printf("FORCe /Noise<1> %d %d.%d -Abs\n", bit2, time, decimal);
  printf("FORCe /Noise<2> %d %d.%d -Abs\n", bit3, time, decimal);
  printf("FORCe /Noise<3> %d %d.%d -Abs\n", bit4, time, decimal);
  printf("FORCe /Noise<4> %d %d.%d -Abs\n", bit5, time, decimal);
  printf("FORCe /Noise<5> %d %d.%d -Abs\n", bit6, time, decimal);
  printf("FORCe /Noise<6> %d %d.%d -Abs\n", bit7, time, decimal);
  printf("FORCe /Noise<7> %d %d.%d -Abs\n", bit8, time, decimal);
}
```

H.2 Header File Prepended to Output of C Code

```
// SET USer Scale -type Time 1e-09
// SETup FOrce -Charge
FORCe /4xClk 1  0.0 -Abs
FORCe /Hld_lo<0> 1 0.0 -Abs
FORCe /Hld_lo<1> 1 0.0 -Abs
FORCe /Hld_lo<2> 1 0.0 -Abs
FORCe /Hld_lo<3> 1 0.0 -Abs
FORCe /Hld_lo<4> 1 0.0 -Abs
FORCe /Noise<0> 0 0.0 -Abs
FORCe /Noise<1> 0 0.0 -Abs
FORCe /Noise<2> 0 0.0 -Abs
FORCe /Noise<3> 0 0.0 -Abs
FORCe /Noise<4> 0 0.0 -Abs
FORCe /Noise<5> 0 0.0 -Abs
FORCe /Noise<6> 0 0.0 -Abs
FORCe /Noise<7> 0 0.0 -Abs
FORCe /reset_clk_lo 1  0.0 -Abs
FORCe /reset_gen 0  0.0 -Abs
FORCe /reset_regen_lo 1  0.0 -Abs
FORCe /reset_clk_lo 0  50.0 -Abs
FORCe /4xClk 0  62.5 -Abs
FORCe /reset_clk_lo 1  100.0 -Abs
FORCe /4xClk 1  125.0 -Abs
FORCe /4xClk 0  187.5 -Abs
FORCe /reset_gen 1  187.5 -Abs
FORCe /4xClk 1  250.0 -Abs
FORCe /4xClk 0  312.5 -Abs
FORCe /4xClk 1  375.0 -Abs
FORCe /4xClk 0  437.5 -Abs
FORCe /reset_gen 0  437.5 -Abs
FORCe /4xClk 1  500.0 -Abs
FORCe /4xClk 0  562.5 -Abs
FORCe /4xClk 1  625.0 -Abs
FORCe /4xClk 0  687.5 -Abs
FORCe /Hld_lo<0> 0  687.5 -Abs
FORCe /Hld_lo<1> 0  687.5 -Abs
FORCe /Hld_lo<2> 0  687.5 -Abs
FORCe /Hld_lo<3> 0  687.5 -Abs
FORCe /Hld_lo<4> 0  687.5 -Abs
```

H.3 Modified C Code Used to Create Test Vectors for Modified Timing

```
#include <stdio.h>
#include <sys/time.h>

void main(int argc, char *argv[]);
void usage();
void noise(int noise_ceiling, int time, int decimal);

void main(int argc, char *argv[])
{
  int i,j,d7,d11,d15,d19,d23;
  struct timeval time;
  int noise_ceiling, end_time=-1;

  if ((argc > 3) || (argc < 2))
    usage();

  if (sscanf(argv[1], "%d", &noise_ceiling) == -1)
    usage();
  if (argc == 3)
    if (sscanf(argv[2], "%d", &end_time) == -1)
      usage();

  if (end_time < 2400000) end_time = 2400000;

  gettimeofday(&time,(struct timezone *)0);
  srandom(time.tv_usec ^ time.tv_sec ^ getpid());

  d7 = (int) (random() % 7);
  d11 = (int) (random() % 11);
  d15 = (int) (random() % 15);
  d19 = (int) (random() % 19);
  d23 = (int) (random() % 23);

  for (i = 6000 ; (d7+d11+d15+d19+d23) > -5 ; i += 1000)
    {
      if (d7 == 0)
         printf("FORCe /Hld_lo<0> 1 %d.0 -Abs\n",(i-500));
      if (d11 == 0)
         printf("FORCe /Hld_lo<1> 1 %d.0 -Abs\n",(i-500));
      if (d15 == 0)
         printf("FORCe /Hld_lo<2> 1 %d.0 -Abs\n",(i-500));
      if (d19 == 0)
         printf("FORCe /Hld_lo<3> 1 %d.0 -Abs\n",(i-500));
      if (d23 == 0)
         printf("FORCe /Hld_lo<4> 1 %d.0 -Abs\n",(i-500));
```

```
      noise(noise_ceiling, i, 0);
      printf("FORCe /4xClk 1  %d.0 -Abs\n",i);
      noise(noise_ceiling, (i+500), 0);
      printf("FORCe /4xClk 0  %d.0 -Abs\n",(i+500));

      if (d7 >= 0) d7--;
      if (d11 >= 0) d11--;
      if (d15 >= 0) d15--;
      if (d19 >= 0) d19--;
      if (d23 >= 0) d23--;
    }
  for (j = i ; j < i+2000 ; j += 1000)
    {
      noise(noise_ceiling, j, 0);
      printf("FORCe /4xClk 1  %d.0 -Abs\n",j);
      noise(noise_ceiling, (j+500), 0);
      printf("FORCe /4xClk 0  %d.0 -Abs\n",(j+500));
    }

  printf("FORCe /reset_regen_lo 0  %d.0 -Abs\n",(j-500));

  for (i = j ; i < j+2000 ; i += 1000)
    {
      noise(noise_ceiling, i, 0);
      printf("FORCe /4xClk 1  %d.0 -Abs\n",i);
      noise(noise_ceiling, (i+500), 0);
      printf("FORCe /4xClk 0  %d.0 -Abs\n",(i+500));
    }

  printf("FORCe /reset_regen_lo 1  %d.0 -Abs\n",(i-500));

  for (j = i ; j < end_time+1 ; j +=1000)
    {
      noise(noise_ceiling, j, 0);
      printf("FORCe /4xClk 1  %d.0 -Abs\n",j);
      noise(noise_ceiling, (j+500), 0);
      printf("FORCe /4xClk 0  %d.0 -Abs\n",(j+500));
    }
}

void usage()
{
  printf("Usage: force_gen noise_ceiling [end_time]\n");
  printf("  where noise_ceiling and end_time are integers\n");
  exit(2);
}
```

```
void noise(int noise_ceiling, int time, int decimal)
{
  int noise_dec = (int) (random() % noise_ceiling);
  int flip_sign = (int) (random() & 1);
  int bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8;

  if (flip_sign)
    {
      noise_dec--;
      bit1 = ((noise_dec & 1) == 0);
      bit2 = ((noise_dec & 2) == 0);
      bit3 = ((noise_dec & 4) == 0);
      bit4 = ((noise_dec & 8) == 0);
      bit5 = ((noise_dec & 16) == 0);
      bit6 = ((noise_dec & 32) == 0);
      bit7 = ((noise_dec & 64) == 0);
      bit8 = ((noise_dec & 128) == 0);
    }
  else
    {
      bit1 = ((noise_dec & 1) != 0);
      bit2 = ((noise_dec & 2) != 0);
      bit3 = ((noise_dec & 4) != 0);
      bit4 = ((noise_dec & 8) != 0);
      bit5 = ((noise_dec & 16) != 0);
      bit6 = ((noise_dec & 32) != 0);
      bit7 = ((noise_dec & 64) != 0);
      bit8 = ((noise_dec & 128) != 0);
    }

  printf("FORCe /Noise<0> %d %d.%d -Abs\n", bit1, time, decimal);
  printf("FORCe /Noise<1> %d %d.%d -Abs\n", bit2, time, decimal);
  printf("FORCe /Noise<2> %d %d.%d -Abs\n", bit3, time, decimal);
  printf("FORCe /Noise<3> %d %d.%d -Abs\n", bit4, time, decimal);
  printf("FORCe /Noise<4> %d %d.%d -Abs\n", bit5, time, decimal);
  printf("FORCe /Noise<5> %d %d.%d -Abs\n", bit6, time, decimal);
  printf("FORCe /Noise<6> %d %d.%d -Abs\n", bit7, time, decimal);
  printf("FORCe /Noise<7> %d %d.%d -Abs\n", bit8, time, decimal);
}
```

H.4 Modified Header File Prepended to Output of Modified C Code

```
// SET USer Scale -type Time 1e-09
// SETup FOrce -Charge
FORCe /4xClk 1  0.0 -Abs
FORCe /Hld_lo<0> 1 0.0 -Abs
FORCe /Hld_lo<1> 1 0.0 -Abs
FORCe /Hld_lo<2> 1 0.0 -Abs
FORCe /Hld_lo<3> 1 0.0 -Abs
FORCe /Hld_lo<4> 1 0.0 -Abs
FORCe /Noise<0> 0 0.0 -Abs
FORCe /Noise<1> 0 0.0 -Abs
FORCe /Noise<2> 0 0.0 -Abs
FORCe /Noise<3> 0 0.0 -Abs
FORCe /Noise<4> 0 0.0 -Abs
FORCe /Noise<5> 0 0.0 -Abs
FORCe /Noise<6> 0 0.0 -Abs
FORCe /Noise<7> 0 0.0 -Abs
FORCe /reset_clk_lo 1  0.0 -Abs
FORCe /reset_gen 0  0.0 -Abs
FORCe /reset_regen_lo 1  0.0 -Abs
FORCe /reset_clk_lo 0  400.0 -Abs
FORCe /4xClk 0  500.0 -Abs
FORCe /reset_clk_lo 1  800.0 -Abs
FORCe /4xClk 1  1000.0 -Abs
FORCe /4xClk 0  1500.0 -Abs
FORCe /reset_gen 1  1500.0 -Abs
FORCe /4xClk 1  2000.0 -Abs
FORCe /4xClk 0  2500.0 -Abs
FORCe /4xClk 1  3000.0 -Abs
FORCe /4xClk 0  3500.0 -Abs
FORCe /reset_gen 0  3500.0 -Abs
FORCe /4xClk 1  4000.0 -Abs
FORCe /4xClk 0  4500.0 -Abs
FORCe /4xClk 1  5000.0 -Abs
FORCe /4xClk 0  5500.0 -Abs
FORCe /Hld_lo<0> 0  5500.0 -Abs
FORCe /Hld_lo<1> 0  5500.0 -Abs
FORCe /Hld_lo<2> 0  5500.0 -Abs
FORCe /Hld_lo<3> 0  5500.0 -Abs
FORCe /Hld_lo<4> 0  5500.0 -Abs
```

References

[1] Baugh, Harold W., "Sequential Ranging -- How it Works," JPL Publication 93-18, Jet Propulsion Laboratory, Pasadena, CA, June 15, 1993.

[2] Blanchard, Alain, *Phase-Locked Loops*, John Wiley & Sons, New York, 1976.

[3] Davenport, Wilbur B. Jr., *Probability and Random Processes*, McGraw-Hill, New York, 1970.

[4] Davenport, Wilbur B. Jr., and Root, William J., *An Introduction to the Theory of Random Signals and Noise*, McGraw-Hill, New York, 1958.

[5] Goldstein, R. M., "Ranging with Sequential Components," Space Program Summary 37-52, Jet Propulsion Laboratory, July, 1968, Vol. II, pp. 46-49.

[6] Golomb, Solomon W., *Digital Communications with Space Applications*, Prentice-Hall, New Jersey, 1964.

[7] Johnson, Glenn, Personal Communication.

[8] Martin, W. L., "Information Systems: A Binary-Coded Sequential Acquisition Ranging System," Space Program Summary 37-57, Jet Propulsion Laboratory, May, 1969, Vol. II, pp. 72-81.

[9] Martin, W. L., "Information Systems: Performance of the Binary-Coded Sequential Acquisition Ranging System at DSS 14," Space Program Summary 37-62, Jet Propulsion Laboratory, March, 1970, Vol. II, pp. 55-61.

[10] Smith, John, Personal Communication.

[11] Tausworthe, R. C., Personal Communication.

[12] Tausworthe, R. C., "Tau Ranging Revisited," TDA Progress Report 42-91, Jet Propulsion Laboratory, November, 1987, pp. 318-324.

[13] Yuen, Joseph H., *Deep Space Telecommunications Systems Engineering*, Plenum Press, New York, 1983.