A Hardware Implementation of a Coherent SOQPSK-TG Demodulator for FEC Applications

by

Gino Pedro Enrique Rea Zanabria

Submitted to the graduate degree program in Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Thesis Committee:

Dr. Erik Perrins: Chairperson

Dr. Andrew Gill

Dr. Shannon Blunt

Date Defended

The Thesis Committee for Gino P.E. Rea Zanabria certifies
that this is the approved version of the following thesis:

A Hardware Implementation of a Coherent SOQPSK-TG Demodulator for FEC Applications

Committee:

_____

Chairperson

_____

_____

_____

Date Approved

Acknowledgements

First of all, I would like to thank my family for always supporting and encouraging me throughout this incredible journey. They have been there when I most needed them, and I know that without their love and guidance, none of this would have been possible. I would also like to thank Dr. Erik Perrins, my academic advisor, for giving me the opportunity to be part of his research team. His experience and knowledge in the field of wireless communications have been a source of inspiration throughout these years, and without a doubt, he will be a role model to follow in my professional life. Next, I would like to thank Dr. Andrew Gill and Dr. Shannon Blunt for taking the time to serve on my committee. I have a great respect for their work and, I am honored by having them on my committee. And last but not least, I would like to thank all the friends I made at KU for making this journey more fun, less stressful, and surely one I will never forget.

Abstract

This thesis presents a hardware design of a coherent demodulator for *shaped offset quadrature phase shift keying*, telemetry group version (SOQPSK-TG) for use in *forward error correction* (FEC) applications. Implementation details for data sequence detection, symbol timing synchronization, carrier phase synchronization, and block recovery are described. This decision-directed demodulator is based on maximum likelihood principles, and is efficiently implemented by the *soft output Viterbi algorithm* (SOVA). The design is intended for use in a *field-programmable gate array* (FPGA). Simulation results of the demodulator's performance in the additive white Gaussian noise channel are compared with a Matlab reference model that is known to be correct. In addition, hardware-specific parameters are presented. Finally, suggestions for future work and improvements are discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In aeronautical telemetry, vital information about an aeronautical vehicle is remotely measured and sent to a distant location for analysis. The operations that aeronautical telemetry perform are numerous and complex, and some of them include new aircraft testing, systems monitoring, missile tracking and positioning, and area surveillance. The success of an aeronautical telemetry mission is highly dependent on the robustness of the communication link between the aeronautical vehicle and the ground station. Due to the inherent cost of each flight test, the receiver must be able to recover the transmitted information from the noisy received signal, and avoid costly retransmissions.

In an effort to upgrade its current communication methods, the aeronautical telemetry community has taken part in a migration to *forward error correction* (FEC) codes in the recent years. By introducing meaningful redundancy into the stream of data, FEC codes allow the receiver to detect and correct errors, up to some limit, without the need and, more importantly, the cost of data retransmis-

sions. The adoption of FEC codes in aeronautical telemetry is a clear advantage. However, migration to this technology also represents a challenge because existing receivers must be enhanced to be FEC-compatible.

The High-Rate High-Speed Forward Error Correction Architectures for Aeronautical Telemetry (HFEC) project, carried out at The Information and Telecommunication Technology Center (ITTC) at The University of Kansas, is currently investigating modern FEC codes with high-performance iterative decoders. The goal of this research is to develop hardware FEC decoders that are efficient in their use of hardware resources and implementation effort. The project focusses on two FEC codes as design examples. These are low density parity check (LDPC) codes and serially concatenated convolutional codes (SCCC). Both LDPC and SCCC decoders require a demodulator that can provide soft-output, as well as recover the symbol timing and carrier phase from the noisy received signal. The internal components and efficient hardware implementation of this demodulator is the focus of this thesis.

1.2 Objectives

In this thesis, we present a hardware implementation of a fully-synchronized demodulator for *shaped offset quadrature phase shift keying, telemetry group* version (SOQPSK-TG) for use in FEC applications. This demodulator is attractive for its reduced complexity and strong performance, and is efficiently implemented by the *soft output Viterbi algorithm* (SOVA). The main contributions of this work are in the implementation details of data sequence detection, symbol timing synchronization, carrier phase synchronization, and block recovery. This implementation has been written in the widely-used hardware description language known

as VHDL, and is intended for use in a *field-programmable gate array* (FPGA).

1.3 Organization

This thesis is organized into 9 chapters. The information contained in these chapters is listed below (chapters containing the novel contributions of this thesis are marked with a *):

- Chapter 2 gives a description of the signal model for SOQPSK and the most common precoders that are used for this modulation.
- Chapter 3 introduces the two iterative decoders considered as design examples in the HFEC project: SCCC and LDPC.
- Chapter 4 describes a reduced-complexity approach for the detection of SOQPSK via the soft-output Viterbi algorithm.
- Chapter 5 explains how symbol timing synchronization is achieved.
- Chapter 6 explains how carrier phase synchronization is achieved.
- *Chapter 7 gives a highly-detailed look at a hardware design of the fully-synchronized SOQPSK-TG demodulator. This chapter contains the majority of the work of this thesis, and therefore is longer.
- *Chapter 8 reveals the results of the hardware implementation of the SOQPSK-TG demodulator in VHDL.
- *Chapter 9 gives conclusions and suggestions for future improvements.

Chapter 2

Description of SOQPSK

This chapter describes the signal model for SOQPSK and the most common precoders that are used for this modulation.

2.1 CPM Signal Model

The SOQPSK signal is defined as a CPM [1] with the complex baseband representation

$$s(t;\boldsymbol{\alpha}) \triangleq \sqrt{\frac{E}{T}}\, e^{j\phi(t;\boldsymbol{\alpha})} \tag{2.1}$$

where E is the symbol energy, and T is the symbol time. The phase is a pulse train of the form

$$\phi(t;\boldsymbol{\alpha}) \triangleq 2\pi h \sum_{i=-\infty}^{k} \alpha_i q(t-iT), \quad kT \le t < (k+1)T \tag{2.2}$$

where $h = 1/2$ is the modulation index, and $\alpha_i \in \{-1, 0, 1\}$ is a transmitted symbol. We use this notation to be consistent with previous work with SOQPSK; nonetheless, it is in conflict with traditional CPM notation. In strict CPM terms,

we really have $h = 1/4$ and $\alpha_i \in \{-2, 0, 2\}$ when the data alphabet is ternary ($M = 3$). The phase pulse $q(t)$ is defined as

$$q(t) \triangleq \begin{cases} 0, & t < 0 \\ \displaystyle\int_0^t f(\sigma)\, d\sigma, & 0 \le t < LT \\ 1/2, & t \ge LT \end{cases} \tag{2.3}$$

where $f(t)$ is the frequency pulse, which has a duration of L symbol times and an area of $1/2$. When the frequency pulse lasts one symbol time ($L = 1$), it is said to be *full-response*; however, when it lasts more than one symbol time ($L > 1$), it is said to be *partial-response*. Due to the constraints on $f(t)$ and $q(t)$, the phase in (2.2) may be expressed as

$$\phi(t; \boldsymbol{\alpha}) = \underbrace{2\pi h \sum_{i=k-L+1}^{k} \alpha_i q(t - iT)}_{\theta(t; \boldsymbol{c}_k; \alpha_k)} + \underbrace{\pi h \sum_{i=0}^{k-L} \alpha_i}_{\theta_k} \tag{2.4}$$

with support on the interval $kT \le t < (k+1)T$. The first term $\theta(t; \boldsymbol{c}_k; \alpha_k)$ is the *correlative phase* and is a function of the *correlative state vector* $\boldsymbol{c}_k \triangleq [\alpha_{k-L+1}, ..., \alpha_{k-2}, \alpha_{k-1}]$ and the current symbol α_k. The correlative phase contains the L most recent symbols being modulated by the phase pulse. The second term θ_k is the *phase state* and is a function of the remaining symbols. Due to the fact that h is a rational number, the phase state can only assume $p = 4$ distinct values when taken modulo-2π, which are $\theta_k \in \{0, \pi/2, \pi, 3\pi/2\}$. When this result is applied in (2.1), it gives $e^{j\theta_k} \in \{\pm 1, \pm j\}$.

There are multiple versions of SOQPSK, which differ by their respective frequency pulses. In this work, we focus on the version recently adopted in aeronau-

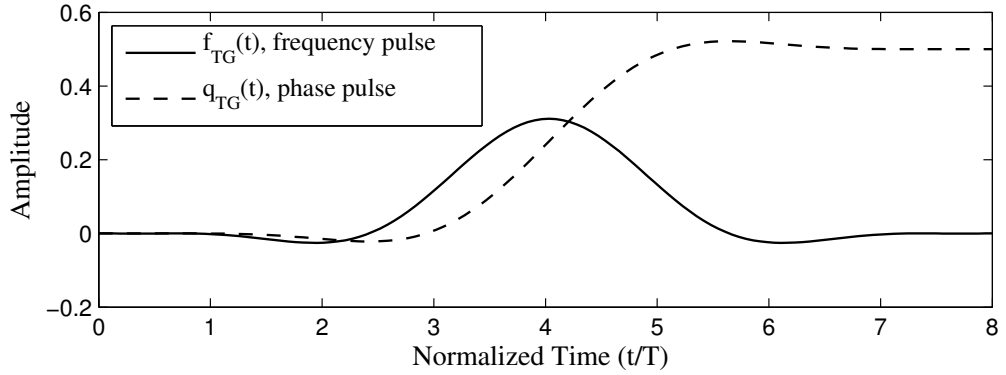

Figure 2.1. Length-$8T$ frequency pulse and corresponding phase pulse for SOQPSK-TG.

tical telemetry, known as "SOQPSK-TG" [2]. It uses a partial-response frequency pulse with $L = 8$, which is given by

$$f_{\mathrm{TG}}(t) \triangleq A \frac{\cos\left(\frac{\pi\rho Bt}{2T}\right)}{1 - 4\left(\frac{\rho Bt}{2T}\right)^2} \times \frac{\sin\left(\frac{\pi Bt}{2T}\right)}{\frac{\pi Bt}{2T}} \times w(t) \tag{2.5}$$

where the window is

$$w(t) \triangleq \begin{cases} 1, & 0 \leq \left|\frac{t}{2T}\right| < T_1 \\ \frac{1}{2} + \frac{1}{2}\cos\left(\frac{\pi}{T_2}\left(\frac{t}{2T} - T_1\right)\right), & T_1 \leq \left|\frac{t}{2T}\right| \leq T_1 + T_2 \\ 0, & T_1 + T_2 < \left|\frac{t}{2T}\right| \end{cases} \tag{2.6}$$

The constant A is chosen to give the pulse an area of $1/2$ and $T_1 = 1.5$, $T_2 = 0.5$, $\rho = 0.7$, and $B = 1.25$. The partial-response frequency pulse shown in Fig. 2.1 results in a more compact spectrum (compared to other frequency pulses) and was selected to meet the bandwidth constraints of the aeronautical telemetry community [2].

2.2 Frequency Pulse Truncation for SOQPSK-TG

The structure of the CPM phase in (2.4) is conveniently described by a phase trellis comprised of pM^{L-1} states. For SOQPSK-TG, this amounts to $pM^{L-1} = 512$ states. An optimal detector for this version of SOQPSK would consequently require a 512-state trellis, which is impractical and highly complex. Due to this reason, we pursue a near-optimum approximation for SOQPSK-TG, known as *pulse truncation* (PT) [3, 4]. This approximation results in a simple detector that is based on a four-state trellis with a loss in performance of only 0.2 dB [5].

The PT approximation for SOQPSK-TG is based on the fact that the frequency pulse $f_{\text{TG}}(t)$ shown in Fig. 2.1 is near-zero for a significant portion of its duration. Using this argument, the frequency pulse can be truncated to only include its smooth time-varying section. In other words, the truncation is centered such that half is applied to the beginning of the pulse and half to the end. After translating these conditions to the phase pulse we obtain the modified phase pulse

$$q_{\text{PT}}(t) = \begin{cases} 0, & t < 0 \\ q(t + (L-1)T/2), & 0 \le t \le T \\ 1/2, & t > T \end{cases} \tag{2.7}$$

It is important to notice that since $q_{\text{PT}}(t)$ has variations only in the time interval $[0, T]$, it behaves like a full-response pulse ($L = 1$). This implies that the correlative state vector $\boldsymbol{c}_k$ in (2.4) is empty; and thus, it will be omitted from the notation used in future chapters. We base the detector presented in this work on this truncated phase pulse.

2.3 SOQPSK Precoders

SOQPSK is different from ordinary CPM in that it uses a precoding operation to convert the binary sequence $\{u_k\}$ into a ternary sequence $\{\alpha_k\}$. The signal model for uncoded SOQPSK is shown in Fig. 2.2. In this section, we describe two of the most commonly used precoders for SOQPSK.

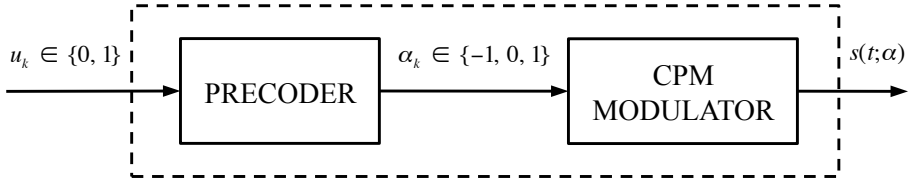

Figure 2.2. Signal model for uncoded SOQPSK.

2.3.1 Standard Precoder

The standard precoder converts the binary input bits $\{u_k\}$ into ternary data $\{\alpha_k\}$ according to the mapping [6]

$$\alpha_k(\boldsymbol{u}) \triangleq (-1)^{k+1}(2u_{k-1} - 1)(u_k - u_{k-2}) \tag{2.8}$$

where $u_k \in \{0, 1\}$ and $\alpha_k \in \{-1, 0, +1\}$. The role of the precoder is to orient the phase of the CPM signal in (2.4), such that it behaves like the phase of an OQPSK signal that is driven by the bit sequence $\boldsymbol{u}$. For convenience, in what follows we refer to $\alpha_k(\boldsymbol{u})$ as α_k, but we stress that $\boldsymbol{u}$ is the underlying bit sequence.

The precoder imposes three important constraints on the ternary data [6]:

1. In any given bit interval, α_k is drawn from one of two binary alphabets, $\{0, +1\}$ or $\{0, -1\}$.

2. When $\alpha_k = 0$, the binary alphabet for α_{k+1} switches from the one used for α_k, but when $\alpha_k \neq 0$ the binary alphabet for α_{k+1} does not change.

3. A value of $\alpha_k = +1$ cannot be followed by $\alpha_{k+1} = -1$, and vice versa.

These constraints imply that not every possible ternary symbol pattern is a valid SOQPSK data pattern. For example, the ternary data sequences $\ldots, 0, -1, +1, 0, \ldots$ and $\ldots, -1, 0, -1, \ldots$ violate the SOQPSK constraints.

2.3.2 Recursive Precoder

Another frequently used precoder that satisfies these constraints can be obtained by differentially encoding the input bits u_k at the transmitter. The differential (recursive) nature of this precoder is essential when SOQPSK is used as the inner code in a serially concatenated system [7]. The differentially encoded bits are

$$d_k = u_k \oplus d_{k-2} \tag{2.9}$$

where $\oplus$ is the XOR operator for binary data in the set $\{0, 1\}$. The precoder in this case is

$$\alpha_k(\boldsymbol{u}) = (-1)^k \, u_k \, d'_{k-1} \, d'_{k-2} \tag{2.10}$$

where $d' \in \{-1, +1\}$ is the antipodal counterpart of d_k and is given by $d'_k = 2d_k - 1$.

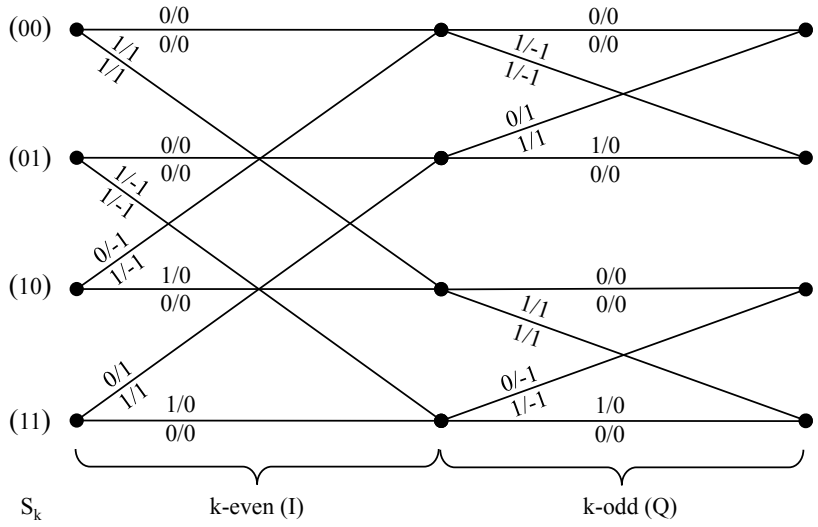

Figure 2.3. Four-state time-varying trellis. The labels above each branch are for the standard precoder in (2.8), while the labels below each branch are for the recursive precoder in (2.10). The branch labels indicate the input-bit/output-symbol pair u_k/α_k.

2.4 Trellis Representation

The precoder/CPM modulator pair shown in Fig. 2.2 can be thought of as having a *state* at any time throughout the encoding process. Using u_{k-1}, u_{k-2}, and k-even/k-odd from the standard precoder (2.8) as state variables, it has been shown that eight states are required to describe the precoder/CPM system [8]. We may reduce the number of states from eight to four if we construct a *time-varying* trellis, with different sections for k-even and k-odd. This four-state time-varying trellis is shown in Fig. 2.3. The labels above each branch show the input-bit/output-symbol pair u_k/α_k for the given branch using the standard precoder. The state variable pairs $S_k \in \{00, 01, 10, 11\}$ shown on the left side of the trellis are ordered (u_{k-2}, u_{k-1}) for k-even and (u_{k-1}, u_{k-2}) for k-odd. When k is even, the input bit u_k replaces the leftmost bit in the pair, and when k is odd, it replaces

the rightmost bit. It is important to note that for any given time interval k, each branch is identified with a unique value of the *branch vector* $[u_k, S_k]$ [5].

Similarly, the recursive precoder (2.10) is also described by the four-state time-varying trellis in Fig. 2.3. The labels below each branch show the input-bit/output-symbol pair u_k/α_k for the recursive precoder. In this case, the state variables are d_{k-1} and d_{k-2}, instead of u_{k-1} and u_{k-2}. The state variable pairs S_k are ordered and updated in the same way as before. Although each precoder imposes a different input-bit/output-symbol mapping, the output-symbols are identical in either case.

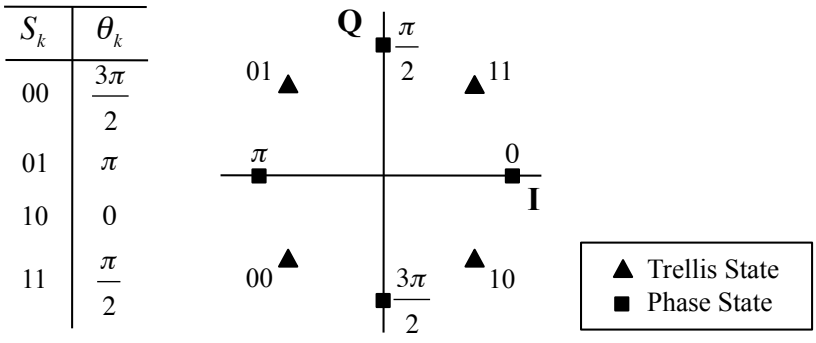

Figure 2.4. Mapping between the trellis state variable pairs S_k and the CPM phase states θ_k.

A key relationship between the SOQPSK precoders and the CPM modulator is that the state variable pairs S_k and the CPM phase state θ_k are interchangeable as state variables [9]. This one-to-one mapping is shown in Fig. 2.4 and is essential to the reduced-complexity characteristic of the detector proposed herein.

Chapter 3

Coded SOQPSK Iterative Decoders

SOQPSK serves as the inner code in the two concatenated coded modulation schemes investigated by the HFEC project. In order to present a framework for the demodulator described in this work, this chapter describes the two iterative decoders considered as design examples.

3.1 Serially Concatenated Convolutional Code Decoder

The SCCC modulation scheme under consideration is shown in Fig. 3.1. The encoder/transmitter portion of the system consists of a convolutional code (CC) encoder, an S-random interleaver (labeled as "Π" in the block diagram), the recursive SOQPSK precoder from (2.10), and a CPM modulator. Therefore, the CC serves as the outer code, and SOQPSK serves as the inner code in a serially concatenated coding scheme. The recursive formulation of the precoder is necessary to yield large coding gains from the concatenation of the outer CC and the interleaver [5].

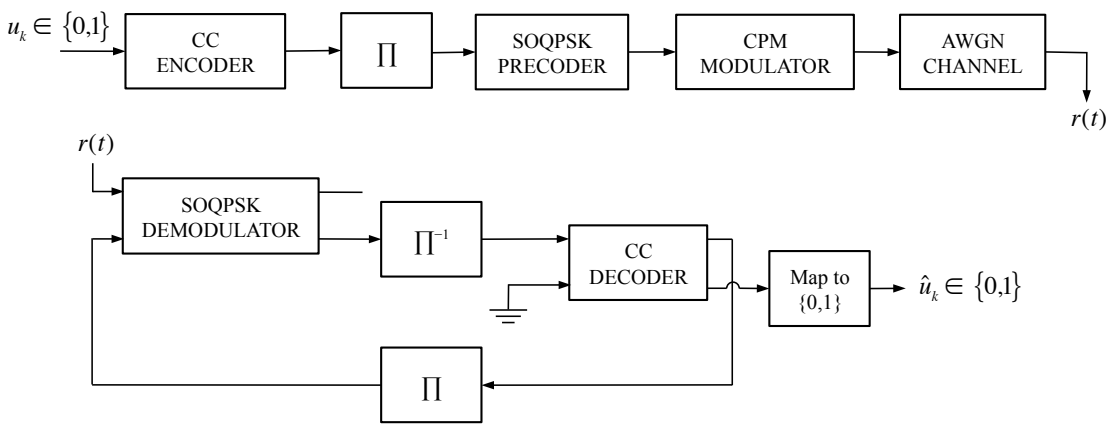

Figure 3.1. Block diagram of a serially concatenated convolutional code decoder.

In the receiver portion of the system, an iterative decoding approach is used. Instead of making one pass over the concatenated decoder, the iterative method performs several. Soft decisions about the inner code are produced from the SOQPSK demodulator, de-interleaved and fed into the CC decoder. Then, soft decisions about the outer code are produced from the CC decoder, re-interleaved and used as prior information in the SOQPSK demodulator. Since there is never any prior information about the outer code, that input in the CC decoder is assumed to be zero (shown with a "ground" symbol). The decoding operation repeats itself for a set number of iterations, after which, a final binary output is generated.

While Fig. 3.1 only shows one version of the SOQPSK demodulator, in reality this iterative decoding scheme requires two versions. For the first iteration, a full-version of the demodulator is required to recover the symbol timing and carrier phase of the received signal, and at the same time, to estimate the transmitted bit sequence. Ordered matched filter outputs from within the demodulator are stored to be used as information inputs to the demodulator for the second and

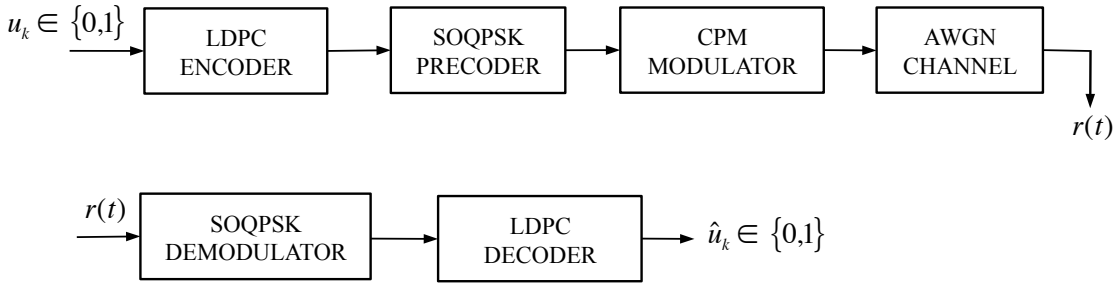

Figure 3.2. Block diagram of a concatenated low density parity check decoder.

following iterations through the decoder. We refer to this ordered matched filter outputs as branch increments in the following chapters. The branch increments are already time-synchronized and phase-corrected; therefore, in order to process these inputs only a simple-version of the demodulator is required.

This iterative decoding method provides a significant increase in performance over a single iteration. In addition, the use of a soft-decision implementation for the SOQPSK demodulator and the CC decoder provides a 1-2 dB gain in BER performance over a hard-decision implementation [10]. Both, the demodulator and the decoder are efficiently implemented by the soft-output Viterbi algorithm. The use of interleavers (Π) helps the system manage bursts of errors, which the Viterbi algorithm is very sensitive to.

3.2 Low Density Parity Check Decoder

The concatenated LDPC modulation scheme under consideration is shown in Fig. 3.2. The encoder/transmitter portion of the system consists of an LDPC encoder, the standard SOQPSK precoder from (2.8), and a CPM modulator. In this case, LDPC serves as the outer code, and SOQPSK serves as the inner code.

In the receiver portion of the system, soft decisions about the inner code are produced by the SOQPSK demodulator and provided as inputs to the LDCP decoder. Unlike the SCCC model, the concatenated LDPC scheme only performs one pass over the decoder; therefore, it only requires the full version of the demodulator. The iterative nature of this concatenated decoder comes from the fact that the LDPC decoder performs a fixed number of attempts on the input stream to try to decode the transmitted information. The LDPC algorithm has the advantage of knowing with certainty if the decoding operation was successful, unlike other decoding methods. Therefore, after a set number of iterations, the LDPC decoder outputs a binary sequence if successful, or a decoding failure message, otherwise.

Chapter 4

Sequence Detection for SOQPSK

Consider a signaling waveform sent through additive white Gaussian noise, the AWGN channel. The received signal model is

$$r(t) = \sqrt{\frac{E}{T}}\, e^{j\phi(t-\tau;\boldsymbol{\alpha})} e^{j\phi_0} + w(t) \tag{4.1}$$

where $w(t)$ is a zero-mean complex-valued AWGN process with one-sided power spectral density N_0. This representation shows that the data symbols $\boldsymbol{\alpha}$, the symbol timing τ, and the carrier phase ϕ_0, are unknown to the receiver and must be handled appropriately. A method to recover τ and ϕ_0, based on *maximum likelihood* (ML) principles, is developed in Chapters 5 and 6. In this chapter, we describe a *maximum likelihood sequence detection* (MLSD) approach used to decode the data symbols $\boldsymbol{\alpha}$. This approach is efficiently implemented via the *soft-output Viterbi algorithm* (SOVA). In what follows, we refer to the estimated and hypothesized values of a generic quantity a as $\hat{a}$ and $\tilde{a}$ respectively. Also, $\hat{a}$ and $\tilde{a}$ can assume the same value of a itself.

4.1 Maximum Likelihood Sequence Detection

CPM signals are optimally demodulated by applying MLSD [1, Ch. 7]. Since SOQPSK is a form of CPM, MLSD can be applied to recover the symbol sequence $\boldsymbol{\alpha}$ (and consequently, the underlying bit sequence $\boldsymbol{u}$).

In order to develop this approach, the detector first assumes that the symbol timing τ and the carrier phase ϕ_0 are known [11]. Using the CPM model for SOQPSK in (2.4), it was shown in [5] that the likelihood function for (4.1), given a hypothetical bit sequence $\tilde{\boldsymbol{u}}$ over the interval $0 \leq t \leq T$ is

$$\Lambda(\boldsymbol{r}|\tilde{\boldsymbol{u}}) = \exp\left\{ \frac{1}{N_0}\sqrt{\frac{E}{T}}\,\mathrm{Re}\left\{ e^{-j\phi_0}\boldsymbol{Z}_k(\tilde{\alpha}_k,\tau)e^{-j\tilde{\theta}_k} \right\} \right\} \tag{4.2}$$

where $\boldsymbol{Z}_k(\cdot)$ are the *matched filter* (MF) outputs. The variables $\tilde{\alpha}_k$ and $\tilde{\theta}_k$ correspond to hypothetical values obtained from $\tilde{\boldsymbol{u}}$. The MF outputs $\boldsymbol{Z}_k(\tilde{\alpha}_k,\tau)$ are sampled at the instant $\tau + (k+1)T$ to produce

$$\boldsymbol{Z}_k(\tilde{\alpha}_k,\tau) \triangleq \int_{\tau+kT}^{\tau+(k+1)T} r(t)e^{-j2\pi h\tilde{\alpha}_k q_{\mathrm{PT}}(t-\tau-kT)}\,dt \tag{4.3}$$

In order to implement (4.2), the output of three complex-valued MFs is needed. Since the SOVA must consider all possible path histories, a MF output for each possible value of the ternary $\tilde{\alpha}_k$ must be computed. The complex-valued MF outputs for $\tilde{\alpha}_k = \pm 1$ can be constructed from the same four real-valued components due to the identities $\sin(-x) = -\sin(x)$ and $\cos(-x) = \cos(x)$. The MF output for $\tilde{\alpha}_k = 0$ has a value of unity for length-T, which is simply an integrate-and-dump operation that requires no multiplications. Therefore, only four real-valued filtering operations are required in total to implement (4.2).

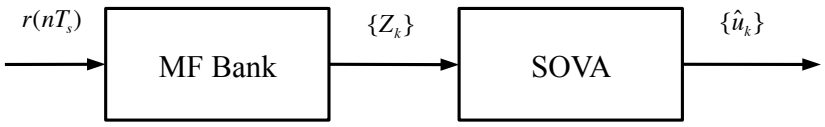

Figure 4.1. Discrete-time approach to MLSD for SOQPSK.

A discrete-time implementation of the sequence detection process is shown in block diagram form in Fig. 4.1. An ADC samples the received signal $r(t)$ at a rate $F_s = \frac{1}{T_s}$ to produce $r(nT_s)$. Then, the samples are fed to the MF bank, whose output forms the values in the set $\{\boldsymbol{Z}_k\}$. The MF outputs are then used to update the branch metrics within the SOVA. The SOVA finds the data symbols sequence $\tilde{\boldsymbol{u}}$ that maximizes (4.2) and outputs the estimated bit sequence $\hat{\boldsymbol{u}}$.

In standard notation, the inputs to the SOVA are real-valued probabilities associated with the hypothetical bit sequence $\tilde{\boldsymbol{u}}$, instead of MF outputs. These probabilities are referred to as *branch increments* and are given by

$$\boldsymbol{B}_k(\tau, \phi_0, [\tilde{u}_k, \tilde{S}_k]) \triangleq \mathrm{Re}\left[e^{-j\phi_0}\boldsymbol{Z}_k(\tilde{\alpha}_k, \tau)e^{-j\tilde{\theta}_k}\right] \tag{4.4}$$

where $\tilde{u}_k$ and $\tilde{S}_k$ are hypothetical values of the branch bit and the state variable, respectively. Each branch increment is identified with a unique value of the branch vector $[\tilde{u}_k, \tilde{S}_k]$. This allows every branch increment to have a one-to-one correspondence with a hypothetical ternary symbol $\tilde{\alpha}_k$ and a hypothetical CPM phase state $\tilde{\theta}_k$, as shown in Figs. 2.3 and 2.4. As a side remark, it is important to note that multiplying by the factor $e^{-j\tilde{\theta}_k} \in \{\pm 1, \pm j\}$ in (4.4) does not require any multiplication resources in the hardware implementation.

Figure 4.2. Block diagram of the soft output Viterbi algorithm.

4.2 SOVA Implementation

The SOVA module under consideration is shown in Fig. 4.2. The module accepts the sequences of a priori probability distributions P($\boldsymbol{c}$; I) and P($\boldsymbol{u}$; I) at the input, and outputs the sequences of probability distributions P($\boldsymbol{c}$; O) and P($\boldsymbol{u}$; O). Here, **c** corresponds to the sequence of coded information, and $\boldsymbol{u}$ corresponds to the sequence of uncoded, underlying information. In this work, we are interested in the two inputs and the $\boldsymbol{u}$ output. The description of the SOVA outlined in this section is based on [12].

To organize the information contained in the trellis shown in Fig. 2.3, and to aid in explaining the operations in the SOVA, we define the following tables. Table 4.1 contains the information for the standard precoder (2.8), while table 4.2 contains the information for the recursive precoder (2.10). The branch index $e \in \{0, 1, 2, 3, ..., 7\}$ is a unique value that identifies each branch in the trellis. This index is ordered from top to bottom, with the branch associated with $u_k = 0$ labeled first than the branch associated with $u_k = 1$ at every trellis state. Also, each branch has an associated starting state $SS(e)$ and an ending state $ES(e)$, which depends on whether k is even or odd. In addition, the branch data $BD(e)$ and branch symbol $BS(e)$ which correspond to the input-bit/output-symbol pair u_k/α_k are also indicated.

Table 4.1. Branch data lookup table for the standard precoder.

e	$SS(e)$	$ES(e)$		$BD(e)$		$BS(e)$	
	S_k			u_k		α_k	
		even	odd	even	odd	even	odd
0	00	00	00	0	0	0	0
1	00	10	01	1	1	1	-1
2	01	01	00	0	0	0	1
3	01	11	01	1	1	-1	0
4	10	00	10	0	0	-1	0
5	10	10	11	1	1	0	1
6	11	01	10	0	0	1	1
7	11	11	11	1	1	0	0

Table 4.2. Branch data lookup table for the recursive precoder.

e	$SS(e)$	$ES(e)$		$BD(e)$		$BS(e)$	
	S_k			u_k		α_k	
		even	odd	even	odd	even	odd
0	00	00	00	0	0	0	0
1	00	10	01	1	1	1	-1
2	01	01	01	0	0	0	0
3	01	11	00	1	1	-1	1
4	10	10	10	0	0	0	0
5	10	00	11	1	1	-1	1
6	11	11	11	0	0	0	0
7	11	01	10	1	1	1	-1

Assume that the SOVA uses K as a time index increasing from 0 to $N-1$, where N is the length of the received sequence. At each decoding step, $\mathrm{P}(\boldsymbol{c};\mathrm{I})$ receives eight real-valued inputs (one for each branch in the trellis) corresponding to the branch increments $\boldsymbol{B}_k(\tau, \phi_0, [\tilde{u}_k, \tilde{S}_k])$ in (4.4). For simplicity, in this section we refer to each branch increment as $B_k(e)$, where $e \in \{0, 1, 2, 3, ..., 7\}$ is a branch index.

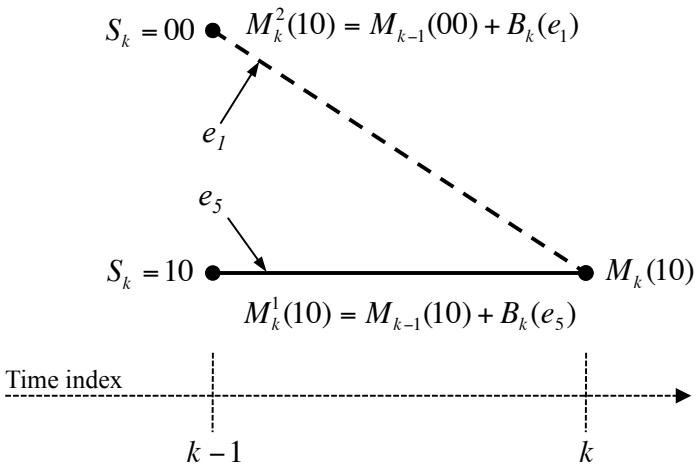

Figure 4.3. Illustration of the metric update process.

With each transition in the binary trellis, two branches enter each trellis state. These are referred to as competing branches, and the SOVA must determine which one is the winning branch. For this purpose, we define the branch metric candidate

$$M_k^{(i)}(ES(e)) = M_{k-1}(SS(e)) + B_k(e) \tag{4.5}$$

where $i \in \{1, 2\}$ is an index to indicate the two competing branches. The value $i = 1$ is typically assigned to the winning candidate, while $i = 2$ is assigned to the losing candidate. The SOVA evaluates the two branch metric candidates terminating at each trellis state S_k, and updates the cumulative metrics according to the following comparison

$$M_k(S_k) = \max \{M_k^{(1)}(S_k),\ M_k^{(2)}(S_k)\} \tag{4.6}$$

Fig. 4.3 shows an illustration of the metric update process. In this example, branch e_1 is considered to be the losing branch, and is marked with a dashed line

to indicate that it will be ignored by the decoder in subsequent operations.

In addition to updating the cumulative metrics, the SOVA must determine the bit $\hat{u}_k$ associated with the winning branch at each trellis state S_k. This is possible by using the one-to-one mapping between branches and the branch vector $[u_k, S_k]$. The decoded bits $\hat{u}_k$ are stored in path decision vectors $\hat{\boldsymbol{u}}(S_k)$, which contain the $(\delta+1)$ most recent decisions $\{\hat{u}_{k-\delta}, ..., \hat{u}_k\}$ at each trellis state S_k. The parameter δ represents the size of the decoding window. It has been shown in, i.e [13], that there is a high probability that the paths at the current stage of the trellis converge to a single surviving path after δ time steps in the decoding process. The use of a decoding window allows the decoder to start generating an output after some number of stages, without the need to traverse the entire received signal.

Next, the SOVA must compute the set of reliabilities $\hat{\boldsymbol{L}}(S_k) = \{\hat{L}_{k-\delta}, ..., \hat{L}_k\}$ associated with the decoded bits in the path decision vectors $\hat{\boldsymbol{u}}(S_k)$ merging at state S_k. To this end, we define

$$\Delta_k(S_k) = |M_k^{(1)}(S_k) - M_k^{(2)}(S_k)| \tag{4.7}$$

and set $\hat{L}_k = \Delta_k(S_k)$ since $\Delta_k(S_k)$ represents the reliability difference between the two most likely code-sequences terminating in state $S_k = ES(e)$ at time step k. Next, the remaining values $\hat{L}_j$, $j = k-\delta, ..., k-1$ of the surviving $\hat{\boldsymbol{L}}(S_k)$ at state S_k have to be updated. The reliabilities update process uses the same notion of competing paths converging at the same trellis state. We refer to these two paths as path-1 and path-2, and without loss of generality assume that path-1 is the surviving path. Therefore, we have the set of reliabilities $\hat{\boldsymbol{L}}^{(1)}(S_k) = \{\hat{L}_{k-\delta}^{(1)}, ..., \hat{L}_{k-1}^{(1)}\}$ for path-1, and $\hat{\boldsymbol{L}}^{(2)}(S_k) = \{\hat{L}_{k-\delta}^{(2)}, ..., \hat{L}_{k-1}^{(2)}\}$ for path-2. Similarly, we have the two path decision vectors $\hat{\boldsymbol{u}}^{(1)}(S_k) = \{\hat{u}_{k-\delta}^{(1)}, ..., \hat{u}_{k-1}^{(1)}\}$ and $\hat{\boldsymbol{u}}^{(2)}(S_k) = \{\hat{u}_{k-\delta}^{(2)}, ..., \hat{u}_{k-1}^{(2)}\}$

corresponding to path-1 and path-2, respectively. First, we consider the case when $\hat{u}_j^{(1)} \neq \hat{u}_j^{(2)}$, for some $j \in \{k-\delta, ..., k-1\}$, and we update as

$$\hat{L}_j(S_k) = \min\ \{\Delta_k(S_k),\ \hat{L}_j^{(1)}\} \tag{4.8}$$

Next, we consider the case when $\hat{u}_j^{(1)} = \hat{u}_j^{(2)}$, for some $j \in \{k-\delta, ..., k-1\}$, and we update as

$$\hat{L}_j(S_k) = \min\ \{\Delta_k(S_k) + \hat{L}_j^{(2)},\ \hat{L}_j^{(1)}\} \tag{4.9}$$

The decoding window of the SOVA applies to the reliabilities in the same way it does to the bits. However, before the reliabilities are sent to the output, they are assigned the sign corresponding to its associated path decision value (positive for $\hat{u}_k = 1$ and negative for $\hat{u}_k = 0$. Next, the input value P($\boldsymbol{u}$; I) associated with decision $\hat{u}_k$ must be subtracted from the newly-computed signed reliabilities. This is due to the fact that the input P($\boldsymbol{u}$; I) is extrinsic information about the code, and hence, it must be removed for the next decoding iteration. The P($\boldsymbol{u}$; I) input is only valid for the SCCC iterative decoder shown in Fig. 3.1, and is non-zero for all the decoding iterations after the first one.

Chapter 5

Symbol Timing Synchronization

Symbol timing synchronization ensures that sampling of the MF outputs is executed at the correct instant. The optimum sampling instant corresponds to the center of the eye diagram, as shown in Fig. 5.1. In general, a clock signal is not transmitted for the purpose of timing synchronization because bandwidth is a limited resource. Therefore, it must be recovered from the noisy received waveforms that carry the data [14, Ch. 8]. In this chapter, we develop a method based on ML principles to recover the symbol timing τ.

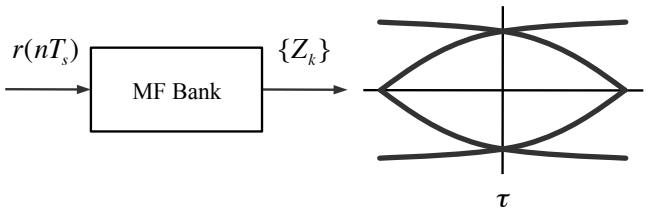

Figure 5.1. Eye diagram showing the optimum sampling instant for the MF outputs.

Since this design is intended for use in digital hardware, the MF bank shown in Fig. 5.1 is implemented as a discrete-time filter. Therefore, an *analog-to-digital converter* (ADC) preceding the MFs is required. The ADC produces T_s-spaced

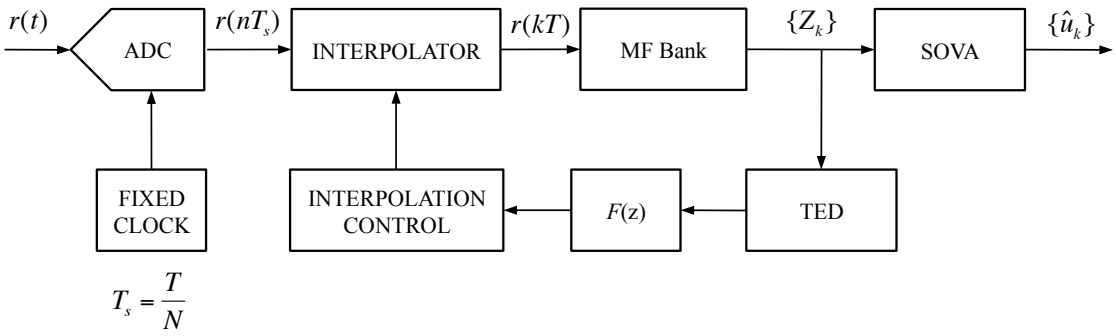

Figure 5.2. A discrete-time approach to symbol timing synchronization for SOQPSK.

samples of the received signal (4.1) at a rate $N = 16$ samples/symbol. Due to the fact that the ADC runs on a fixed clock, the sample rate $1/T_s$ is asynchronous with the symbol rate $1/T$. This timing offset causes the MF bank to produce outputs $\{\boldsymbol{Z}_k\}$ that are not in the optimum sampling instant. The role of the timing synchronizer is to compute samples in the desired time instants using the available samples in $r(nT_s)$, so that the MF outputs are aligned with the center of the eye diagram. This operation is performed by a linear interpolator. A block diagram description of the timing synchronizer is shown in Fig. 5.2. The *timing error detector* (TED) produces a timing error signal based on the MF outputs. This error signal informs the loop filter $F(z)$ about the timing difference, and is used to produce an adjusting signal. The interpolator control block runs a modulo-1 decrementing counter, which is updated using this adjusting signal. When the decrementing counter underflows, it indicates the beginning of a symbol boundary, and provides the fractional interval that the interpolator uses to compute the desired samples.

5.1 Timing Error Detector

The derivation of the TED presented here is based on [11]. In order to recover the symbol timing τ, the ML detector temporarily assumes that the data symbols sequence $\boldsymbol{\alpha}$ and the carrier phase ϕ_0 are known. Using the same definitions from Chapter 4, it was shown in [15] that the likelihood function for (4.1), given a hypothetical timing value $\tilde{\tau}$ over the interval $0 \leq t \leq T$ is

$$\Lambda(\boldsymbol{r}|\tilde{\tau}) = \exp\left\{\frac{1}{N_0}\sqrt{\frac{E}{T}}\,\mathrm{Re}\left\{e^{-j\phi_0}\boldsymbol{Z}_k(\alpha_k,\tilde{\tau})e^{-j\theta_k}\right\}\right\}. \tag{5.1}$$

The ML estimate $\tilde{\tau}$ is the value of τ that maximizes the logarithm of (5.1), the log-likelihood function. In order to find $\tilde{\tau}$, we need to take the partial derivative of the log-likelihood function. Thus, we obtain

$$\frac{\partial}{\partial\tilde{\tau}}\log(\Lambda(\boldsymbol{r}|\tilde{\tau})) = \mathrm{Re}\left\{e^{-j\phi_0}\boldsymbol{Y}_k(\alpha_k,\tilde{\tau})e^{-j\theta_k}\right\} \tag{5.2}$$

where $\boldsymbol{Y}_k(\cdot)$ is the partial derivative of the MF outputs $\boldsymbol{Z}_k(\cdot)$ with respect to $\tilde{\tau}$. The ML estimate $\tilde{\tau}$ is the value of τ that forces (5.2) to zero.

The value $\tilde{\tau}$ is computed in an iterative and adaptive way. Initially, it was assumed that $\boldsymbol{\alpha}$ and ϕ_0 are known, which is not the case. Therefore, two close approximations are used to substitute these values. The true data sequence $\boldsymbol{\alpha}$ is replaced with the estimated decisions $\hat{\boldsymbol{\alpha}}$ within the SOVA, and the true carrier phase ϕ_0 is replaced with the most recent phase estimate $\hat{\phi}_0$ from the phase synchronizer described in Chapter 6. These approximations become more reliable the further we trace back along the trellis. Considering all these factors, the following

timing error signal is obtained as in [15]

$$e_\tau[k-D] \triangleq \mathrm{Re}\left\{e^{-j\hat{\phi}_0[k-D]}\boldsymbol{Y}_{k-D}(\hat{\alpha}_{k-D}, \hat{\tau}[k-D])e^{-j\hat{\theta}_{k-D}}\right\} \tag{5.3}$$

where D represents the delay in computing the error, and $\hat{\alpha}_{k-D}$ and $\hat{\theta}_{k-D}$ are taken from the path history of the best survivor in the SOVA. It is observed in [15] that $D = 1$ produces satisfactory results.

In order to compute the derivative $\boldsymbol{Y}_k(\cdot)$, a discrete-time differentiator would be required. However, it was shown in, e.g. [15], that this value can be approximated with the difference between a late and an early MF output sample. In the implementation of this TED, we use this proposed simplification to calculate $\boldsymbol{Y}_k(\cdot)$.

5.2 Loop Filter

The purpose of the loop filter is to provide an adjusting value to the interpolation control block based on the TED timing error signal. The transfer function for the loop filter in consideration is $F(s) = k$. This is a simple gain and produces a first-order PLL. A block diagram of the loop filter is shown in Fig. 5.3, where $K_p = 1$ and $K_1 = -0.0026$.

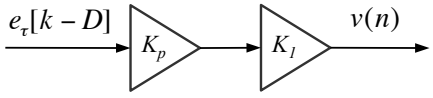

Figure 5.3. A block diagram of the simple gain loop filter $F(s)$.

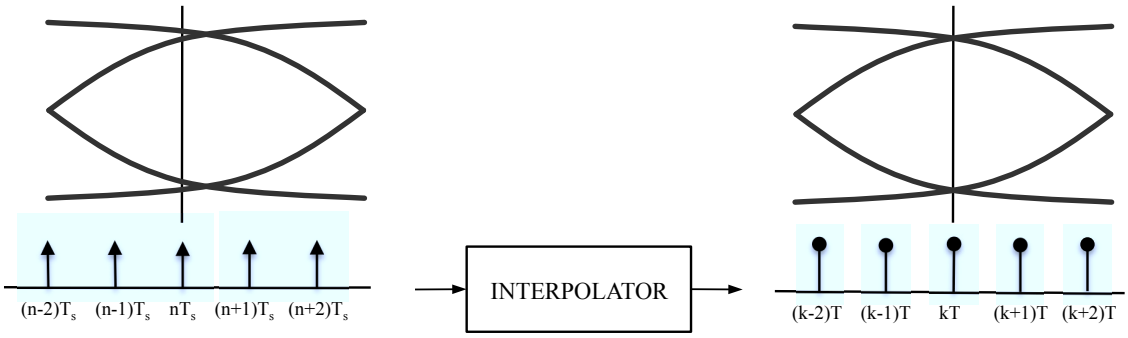

Figure 5.4. Illustration of the interpolation operation to achieve optimum sampling instants. Available samples before interpolation are represented with a triangle, while available samples after interpolation are represented with a circle.

5.3 Interpolation

The continuous-time received signal $r(t)$ in (4.1) is sampled by the ADC at a rate $1/T_s$. This produces T_s-spaced samples, represented with a triangle in Fig. 5.4. Because the sample clock is independent of the data clock used by the transmitter, the sampling instants are not synchronized to the symbol periods. This is illustrated in Fig. 5.4 by showing samples not aligned with the maximum aperture of the eye-diagram. The interpolator uses these available samples to compute desired samples of $r(t)$ at the optimum sampling instances. A desired sample at $t = kT$ is called the k-th interpolant. When the k-th interpolant is between samples $r(nT_s)$ and $r((n+1)T_s)$, the sample index n is called the k-th basepoint index and is denoted $m(k)$. The time instant kT is some fraction of a sample greater than $m(k)T_s$. This fraction is called the k-th fractional interval and is denoted by $\mu(k)$ [14, Ch. 8].

The equation for interpolation may be expressed as

$$r(kT) = r(nT_s) + \mu(k)[r((n+1)T_s) - r(nT_s)] \tag{5.4}$$

for a desired sample at $t = kT$. This sample corresponds to the on-time interpolated sequence that will produce the aligned MF outputs $\{\boldsymbol{Z}_k\}$. It was mentioned earlier that an early and a late MF outputs are also required to approximate the derivative $\boldsymbol{Y}_k(\cdot)$. The early interpolated samples are computed by

$$r((k-1)T) = r((n-1)T_s) + \mu(k)[r(nT_s) - r((n-1)T_s)] \tag{5.5}$$

and the late interpolated samples are found by

$$r((k+1)T) = r((n+1)T_s) + \mu(k)[r((n+2)T_s) - r((n+1)T_s)] \tag{5.6}$$

5.4 Interpolation Control

The purpose of the interpolation control block is to provide the interpolator with the k-th basepoint index $m(k)$ and the k-th fractional interval $\mu(k)$. For the case of this detector, we base the interpolation control on a modulo-1 decrementing counter. This counter is designed to underflow every $N = 16$ samples on average, where the underflows are aligned with the sample times of the desired interpolant. A block diagram of this approach is shown in Fig. 5.5.

The discrete-time samples generated by the ADC are clocked into the interpolator with the same clock used to update the counter. With every clock period, the counter decrements by $1/N$ on average. The loop filter output $v(n)$ adjusts the amount by which the counter decrements. In general, the counter value satisfies

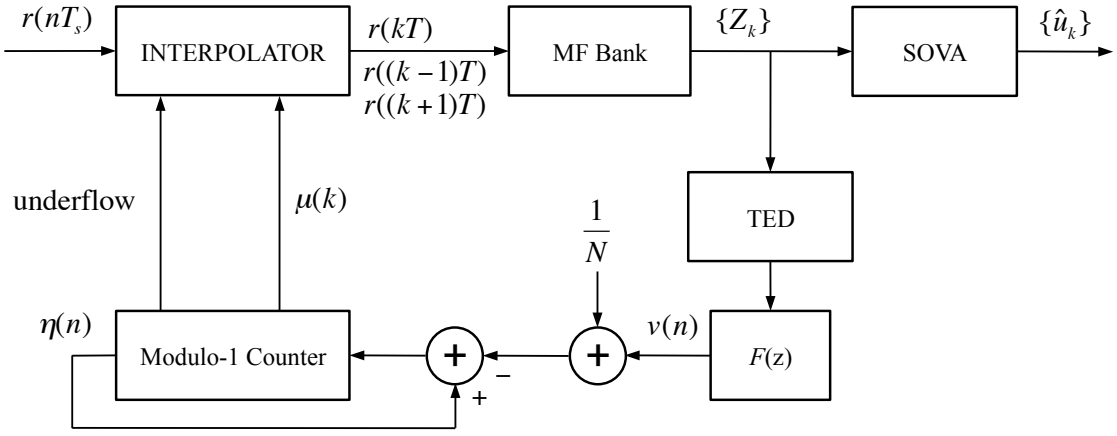

Figure 5.5. A block diagram of the timing synchronizer with the modulo-1 decrementing counter used for interpolation control.

the recursion

$$\eta(n+1) = (\eta(n) - 1/N - v(n)) \mod 1 \tag{5.7}$$

When the decrementing counter underflows, the index n is the basepoint index $m(k)$, as illustrated in Fig. 5.6, and the value of the counter becomes

$$\eta(m(k)+1) = 1 + \eta(m(k)) - 1/N - v(n) \tag{5.8}$$

We notice that when the counter underflows, the values $\eta(m(k))$ and $1-\eta(m(k)+1)$ form similar triangles, which leads to the relationship

$$\frac{\mu(m(k))}{\eta(m(k))} = \frac{1-\mu(m(k))}{1-\eta(m(k)+1)} \tag{5.9}$$

Solving for $\mu(k)$, we obtain

$$\mu(m(k)) = \frac{\eta(m(k))}{\frac{1}{N} + v(n)} \tag{5.10}$$

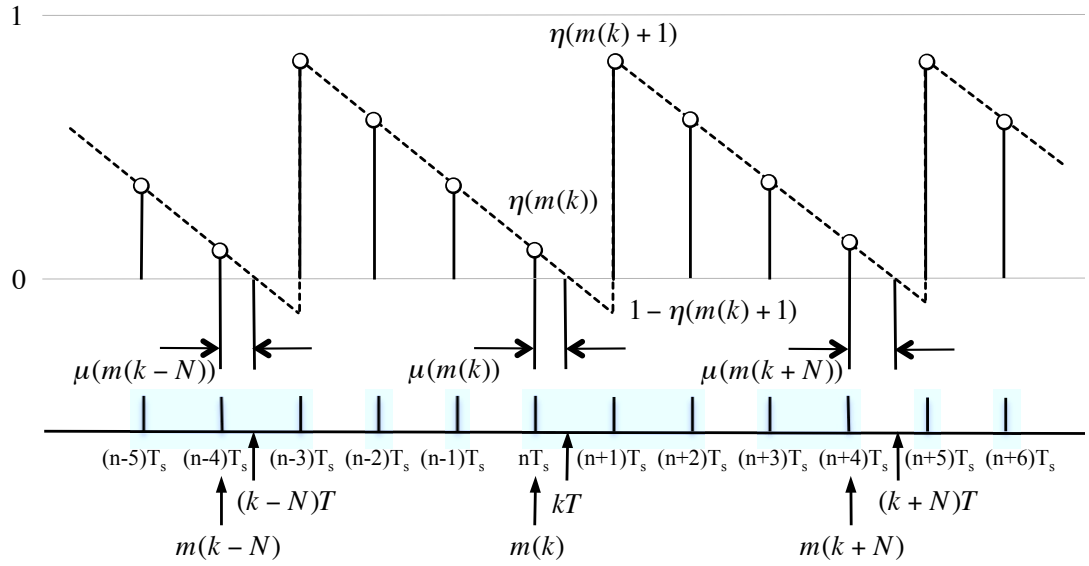

Figure 5.6. Illustration of the modulo-1 decrementing counter underflowing every N samples. In this example, N assumes the value of 4.

When in lock, $v(n)$ is zero on average. Incorporating this consideration into (5.10) produces the final expression for the fractional interval

$$\mu(m(k)) = N\eta(m(k)) \tag{5.11}$$

Chapter 6

Carrier Phase Synchronization

Carrier phase synchronization is the process of forcing the local oscillators in the detector to oscillate in both phase and frequency with the carrier oscillator used at the transmitter. A carrier phase error causes a rotation in the signal space projections. If the rotation is large enough, the signal space projections for each possible symbol lie in the wrong decision region. Consequently, decision errors occur even with perfect symbol timing synchronization and in the absence of additive noise [14, Ch. 7].

The role of the phase synchronizer is to track any residual phase error remaining in the phase after the phase shifts due to the data are removed by a PLL. A block diagram representation of the phase synchronizer is shown in Fig. 6.1. Here, we assume that the discrete-time sequence $r(kT)$ contains the time-synchronized interpolated samples of the discrete-time signal $r(nT_s)$. The complex multiplier rotates these samples in phase by the amount of the most recent carrier phase estimate $\tilde{\phi}_0$. Then, the time and phase-synchronized samples are fed to the MF bank, whose output is used within the SOVA, the TED and the *phase error detector* (PED). The PED produces a phase error signal based on the MF outputs.

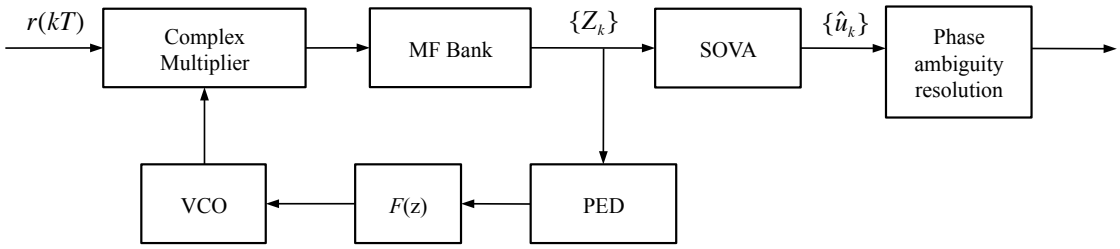

Figure 6.1. A discrete-time approach to phase synchronization for SOQPSK.

This error signal is the input to the loop filter $F(z)$ which drives the discrete-time *voltage-controlled oscillator* (VCO). The VCO outputs an angle that represents the next carrier phase estimate $\tilde{\phi}_0$. At the output of the SOVA, the detector must resolve any phase ambiguity associated with the four possible phase shifts that the PLL can lock on to due to the data. This is discussed at the end of the chapter.

6.1 Phase Error Detector

The implementation of the PED is similar to that of the TED. In order to recover the carrier phase ϕ_0, the ML detector temporarily assumes that the symbol timing τ and the data symbols sequence $\boldsymbol{\alpha}$ are known. Using the same definitions from Chapter 4, the likelihood function for (4.1) given a hypothetical phase value $\tilde{\phi}_0$ over the interval $0 \leq t \leq T$ is

$$\Lambda(\boldsymbol{r}|\tilde{\phi}_0) = \exp\left\{\frac{1}{N_0}\sqrt{\frac{E}{T}}\,\mathrm{Re}\left\{e^{-j\tilde{\phi}_0}\boldsymbol{Z}_k(\alpha_k,\tau)e^{-j\theta_k}\right\}\right\}. \tag{6.1}$$

The ML estimate $\tilde{\phi}_0$ is the value of ϕ_0 that maximizes the logarithm of (6.1), the log-likelihood function. In order to find $\tilde{\phi}_0$, we first need to take the partial

derivative of the log-likelihood function. Thus, we obtain

$$\frac{\partial}{\partial \tilde{\phi}_0} \log(\Lambda(\boldsymbol{r}|\tilde{\phi}_0)) = \mathrm{Im}\left\{-je^{-j\tilde{\phi}_0}\boldsymbol{Z}_k(\alpha_k,\tau)e^{-j\theta_k}\right\} \tag{6.2}$$

where the ML estimate $\tilde{\phi}_0$ is the value of ϕ_0 that forces (6.2) to zero.

Contrary to timing synchronization, in this case, the imaginary part of the MF outputs is forced to zero. This is because of the multiplication of the $-j$ term, which results from the derivative of $e^{-j\tilde{\phi}_0}$, with the real and imaginary arguments of $\boldsymbol{Z}_k(\cdot)$.

Similarly to timing synchronization, the value $\tilde{\phi}_0$ is computed in an iterative and adaptive way. Initially, it was assumed that $\boldsymbol{\alpha}$ and τ are known, which is not the case. Therefore, two close approximations are used to substitute these values. The true data sequence $\boldsymbol{\alpha}$ is replaced with the estimated decisions $\hat{\boldsymbol{\alpha}}$ within the SOVA, and the true symbol timing τ is replaced with the most recent symbol timing estimate $\hat{\tau}$ from the timing synchronizer described in Chapter 5. These approximations become more reliable the further we trace back along the trellis. Considering all these factors, the following phase error signal is obtained

$$e_{\phi_0}[k-D] \triangleq \mathrm{Im}\left\{-je^{-j\hat{\phi}_0[k-D]}\boldsymbol{Z}_{k-D}(\hat{\alpha}_{k-D},\hat{\tau}[k-D])e^{-j\hat{\theta}_{k-D}}\right\} \tag{6.3}$$

where the delay in computing the error is assumed to be $D = 1$ to be consistent with Chapter 5.

6.2 Loop Filter

The transfer function for the loop filter in consideration is $F(s) = k$. This is a simple gain and produces a first-order PLL. A block diagram of the loop filter is shown in Fig. 6.2, where $K_p = 1$ and $K_1 = 0.0026$.

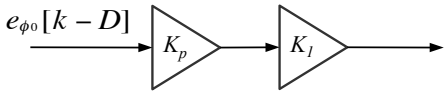

Figure 6.2. A block diagram of the simple gain loop filter $F(s)$.

6.3 Voltage-Controlled Oscillator

The transfer function of the VCO in consideration is $F(s) = K_0/s$, where $K_0 = 1$ is the VCO gain. This is a discrete-time accumulator that stores the running sum of its input. The sum that is stored within the VCO corresponds to the instantaneous phase of the phase error signal produced by the PED. The output of the VCO is the angle corresponding to the next phase error estimate $\hat{\phi}_0[k-D]$. A block diagram representation of the VCO is shown in Fig. 6.3.

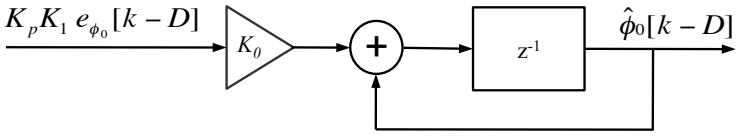

Figure 6.3. A block diagram representation of the voltage-controlled oscillator (VCO).

6.4 Phase Ambiguity Resolution

Similarly to QPSK modulation, SOQPSK exhibits a 90° phase ambiguity. Consequently, the PLL in the phase synchronizer can lock in four different ways with the carrier. It can lock in phase with the carrier, 90° out of phase with the carrier, 180° out of phase with the carrier, or 270° out of phase with the carrier [14, Ch. 6]. If the phase ambiguity is not resolved, decision errors will occur because the symbols constellation will be rotated.

One way of resolving phase ambiguity is by inserting a unique pattern of known symbols (or "attached synch marker" - ASM) in front of the binary sequence $\{u_k\}$. In the receiver, after the carrier phase has been locked, the detector searches for the four possible ASM rotations using a correlation operation, and corrects the phase ambiguity by inverting the appropriate bits according to the detected ASM rotation. A block diagram representation of the phase ambiguity resolution process is shown in Fig. 6.4.

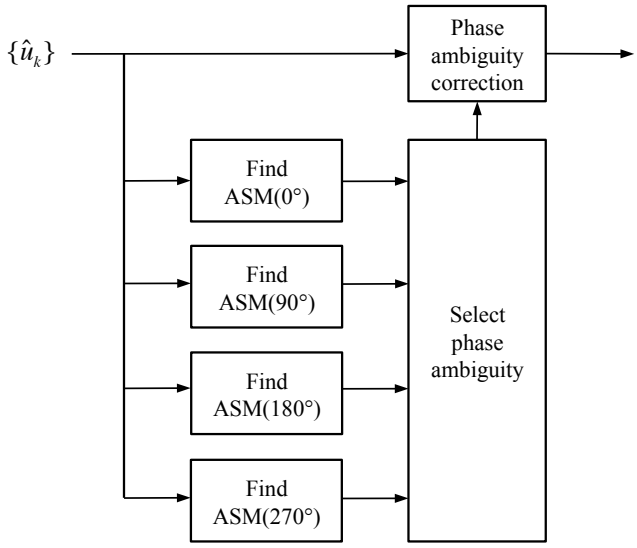

Figure 6.4. Block diagram representation of phase ambiguity resolution for SOQPSK.

Chapter 7

Hardware Implementation

This chapter outlines a detailed hardware implementation of the coherent SOQPSK-TG demodulator described in Chapters 4, 5 and 6. An overview of the proposed design is provided first, followed by a comprehensive description of each hardware component.

7.1 Design Overview

7.1.1 Inputs and Outputs

The description of the design begins with a look at the inputs and outputs to the demodulator. As mentioned in Chapter 3, the iterative decoding schemes targeted by the HFEC project require the implementation of two versions of the demodulator. The full version, which can handle timing and phase synchronization, as well as sequence estimation, is the focus of this chapter. On the other hand, the simple version, which only performs sequence estimation, is not described here as it can be easily deducted from the design of the full version. An illustration of the inputs and outputs of the full demodulator is shown in Fig. 7.1.

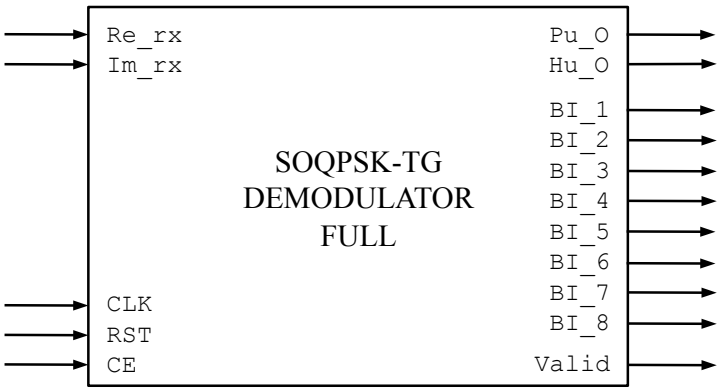

Figure 7.1. A black box view of the full version of the SOQPSK-TG demodulator.

The inputs to the full version of the demodulator are:

- *The information inputs*: `Re_rx`, `Im_rx`. These are the real and imaginary components of the received signal. They are obtained through the processes of sampling and downconversion explained below, and are quantized using eight bits with four bits being fractional.

- *The clock signal*: `CLK`. This signal provides a common time reference to all the components in the design, and it is detected on its rising edge.

- *The reset signal*: `RST`. This signal sets all registers to zero when it is activated, unless noted otherwise in the description. It is asynchronous and active-high.

- *The clock-enable signal*: `CE`. This signal controls the flow of information from external components as it only enables the writing operation of all registers when it is activated.

In the hardware descriptions presented below, the group of control signals: `CLK`, `RST` and `CE`, is collectively referred to as `CTRL`.

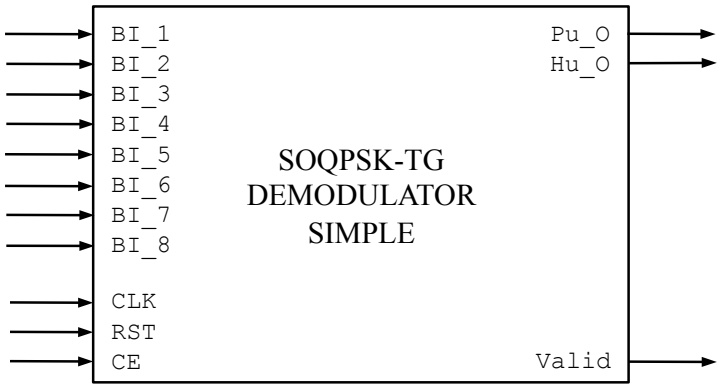

Figure 7.2. A black box view of the simple version of the SOQPSK-TG demodulator.

The outputs of the full version of the demodulator are:

- *The information outputs*: `Pu_O`, `Hu_O`. These are the soft-decisions (reliabilities) and hard-decisions (bits) about the inner code in the concatenated coding schemes described in Chapter 3. The reliabilities are fed to a second decoder in order to estimate the transmitted bit sequence. The hard-decisions are only used for testing purposes.

- *The branch increment outputs*: `BI_1`, ..., `BI_8`. These are the re-ordered time-synchronized and phase-corrected branch increments that are computed at the output of the matched-filters. They serve as information inputs to the simple version of the demodulator in the second and subsequent iterations of the SCCC decoding scheme. An illustration of the inputs and outputs of the simple demodulator is shown in Fig. 7.2.

- *The output valid signal*: `Valid`. This signal indicates the output of the demodulator is valid when it is set to one.

7.1.2 Sampling and Downconversion

The processes of sampling and downconversion are key to understanding the way we extract the information inputs from the received signal $r(t)$. This is a continuous-time band-pass signal centered at the intermediate frequency $f_0 = 70$ MHz. Along with the desired information, noise is also embedded in the signal, so a band-pass filter is first applied to avoid any aliasing effects of noise outside of the bandwidth region. The sample rate is selected in a way that has advantages in the subsequent I/Q downconversion operation. In a process known as band-pass subsampling, the sample rate is selected so as to force the intermediate frequency to alias to the quarter-sample-rate frequency [14, Ch. 8]. There are multiple sample frequencies that achieve this effect, but for the purposes of this design, a sample rate of $F_s = 93\frac{1}{3}$ Msamples/s was selected. This sample rate allows for a maximum usable bandwidth of 46.6667 MHz, which is well above the system's requirement.

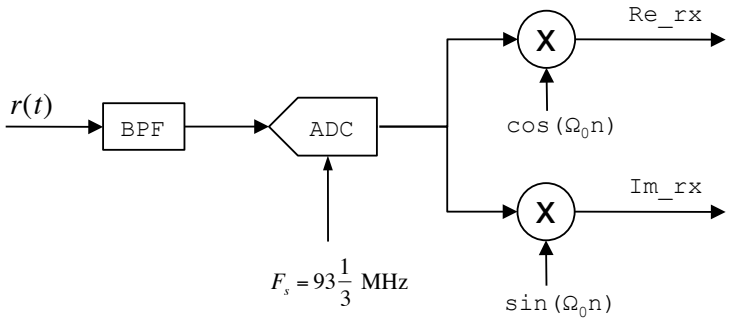

Figure 7.3. Block diagram representation of signal sampling and I/Q downconversion.

As it was mentioned above, the selected sample rate of $F_s = 93\frac{1}{3}$ Msamples/s, has the effect of aliasing the intermediate frequency spectrum of $r(t)$ down to the quarter-sample-rate frequency $f_0' = \frac{1}{4}F_s$. When this is the case, we obtain

$$\Omega_0 = \frac{2\pi f_0'}{F_s} = \frac{\pi}{2} \tag{7.1}$$

so that the I/Q downconversion mixers: $\cos(\Omega_0 n)$ and $\sin(\Omega_0 n)$, assume only three trivial values:

Table 7.1. I/Q downconversion mixers.

n	0	1	2	3	4	5	...
$\cos(n\pi/2)$	1	0	-1	0	1	0	...
$\sin(n\pi/2)$	0	1	0	-1	0	1	...

The fact that the I/Q downconversion mixers only assume the 0, ± 1 values, represents a considerable simplification in the hardware implementation. This is, instead of requiring real multiplications to implement the two frequency translations in Fig. 7.3, they only require simple sign-alterations. The result of the mixing operation is that $r(t)$ is frequency shifted down to baseband.

In this way, the first ADC sample becomes the real input with zero being the imaginary input. Then, the second ADC sample becomes the imaginary input with zero being the real input. After this, the negative of the third ADC sample becomes the real input with zero being the imaginary input. And finally, the negative of the fourth ADC sample becomes the imaginary input with zero being the real input. This pattern is repeated for the remainder of the ADC samples. Any additional phase rotation introduced in the received signal as a product of the downconversion process is measured and corrected by the phase synchronizer.

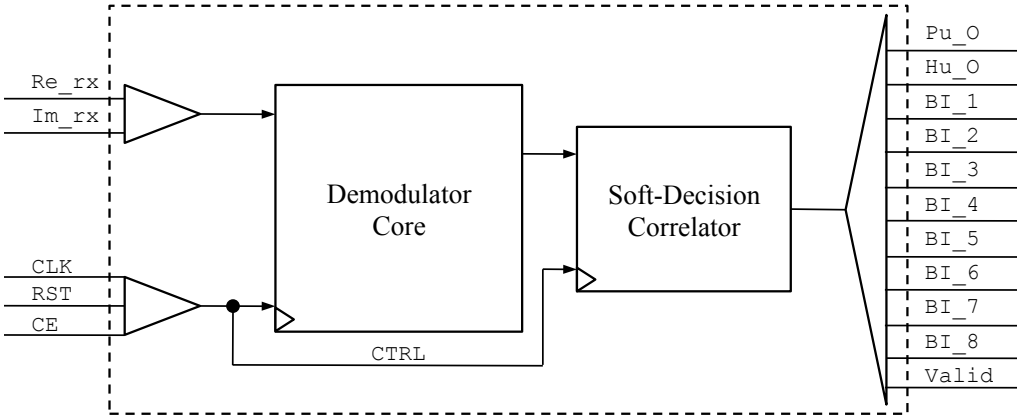

Figure 7.4. Internal structure of the demodulator.

7.1.3 Demodulator Structure

A first-level view of the demodulator structure reveals two major components, as illustrated in Fig. 7.4. These are the *demodulator core* and the *soft-decision correlator*. As the name suggests, the demodulator core is the most extensive and important component in the design. It encompasses all the modules responsible for timing and phase synchronization, as well as, sequence estimation. A detailed view of the demodulator core is given below. The soft-decision correlator serves two essential purposes. The first one is finding the beginning of a frame in the decoded data stream, and the second one is resolving any phase ambiguity in the output data. It does so by performing a correlation of the soft-decisions generated by the demodulator core and a known sequence of bits attached at the beginning of each frame. A detailed hardware description of this module is given in Section 7.10.

A second-level view of the demodulator reveals the internal structure of the demodulator core, as illustrated in Fig. 7.5. Notice how the `CTRL` signal is not shown directly connected to every module; instead, it is represented with a triangle

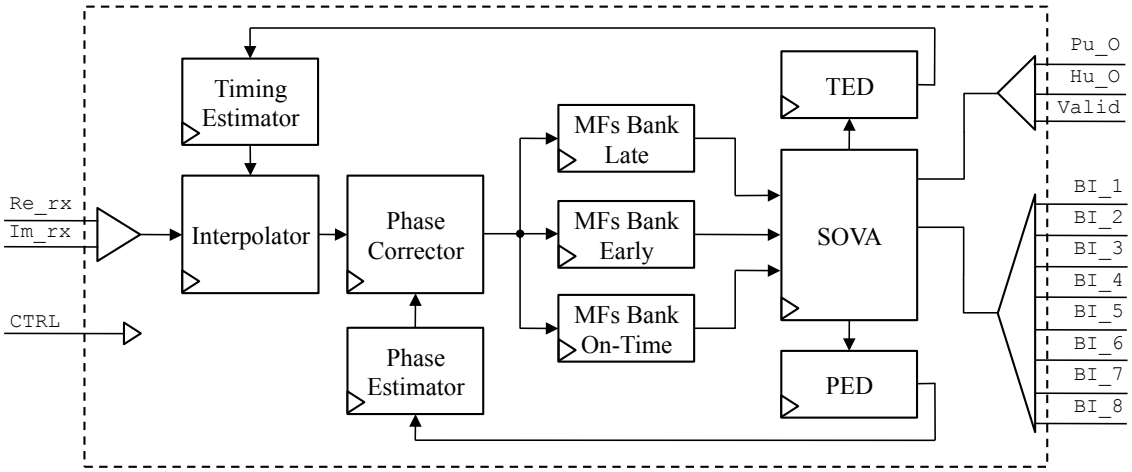

Figure 7.5. Internal structure of the demodulator core.

on their lower left corner. This is done with the purpose to make hardware diagrams easier to read. The internal structure of the demodulator core has been broken down into individual pieces, each one responsible for a separate task. The components that comprise the timing synchronizer are the timing error detector (TED), the timing estimator, and the interpolator. On the other hand, the ones that comprise the phase synchronizer are the phase error detector (PED), the phase estimator, and the phase corrector. And finally, the ones that make up the sequence detector are the matched-filters (MFs) bank, and the soft-output Viterbi algorithm (SOVA).

The information inputs are first processed by the interpolator, which generates a sequence of samples that are aligned with the optimum sampling instances. Also, it produces two additional sequences which represent samples at the early and late sampling moments. The timing estimator, which is comprised by the timing loop filter and the modulo-1 decrementing counter, uses the latest timing error signal from the TED to generate two pieces of information. The first one is

the fractional interval `mu`, which indicates the optimum sampling instances to the interpolator. And the second one is the `underflow` strobe, which signals a new symbol boundary.

The three time-synchronized sequences produced by the interpolator are then processed by the phase corrector, which removes any phase error according to the phase estimator. The phase estimator, which is comprised by the phase loop filter and the VCO, uses the latest phase error signal from the PED to generate an instantaneous phase estimate. The phase corrector rotates the phase of its input by this amount to produce phase-synchronized samples.

The resulting time and phase-synchronized sequences are each passed through their corresponding MFs bank, which are triggered by the `underflow` strobe. This guarantees that all samples corresponding to one symbol are filtered together. Each MFs bank generates three complex-valued outputs, one for each possible transmitted symbol. They also produce a signal that alternates between zero and one with each new output, that is used as a trellis indicator. The outputs of the on-time MFs bank are converted into branch increments within the SOVA, and are then used to compute branch metrics, delta values and winning branch indexes. The outputs of the other MFs banks and the winning branch indexes are used within the TED and the PED to produce the next timing and phase error signals, respectively. The SOVA computes the bits and reliabilities associated with the maximum likelihood path, and outputs them after a decoding window of 16 time steps. The branch increments are delayed throughout the decoding process to be aligned with their corresponding hard-decisions and soft-decisions at the output.

7.2 Interpolator

The interpolator is the point of entry of the information inputs to the demodulator. It produces a sequence of samples that are aligned with the optimum sampling instances, as well as two sequences of samples that correspond to the early and late sampling moments. The inputs to the interpolator are:

- `Re_rx`, `Im_rx`
- `underflow`
- `mu`
- `CTRL`.

And the outputs of the interpolator are:

- `Re_OnTime_rx`, `Im_OnTime_rx`
- `Re_Early_rx`, `Im_Early_rx`
- `Re_Late_rx`, `Im_Late_rx`
- `underflow_out`

A hardware representation of the interpolator is shown in Fig. 7.6. The interpolator receives new values of `Re_rx` and `Im_rx` on the rising edge of every clock cycle. Along with these inputs, it also receives the interpolation control signals `underflow` and `mu`, provided by the modulo-1 decrementing counter. Since we are only interested in the value of `mu` when a new symbol boundary is detected, this input has to be registered. On average, a new symbol boundary is detected every $N = 16$ clock cycles, and it is indicated by the `underflow` strobe.

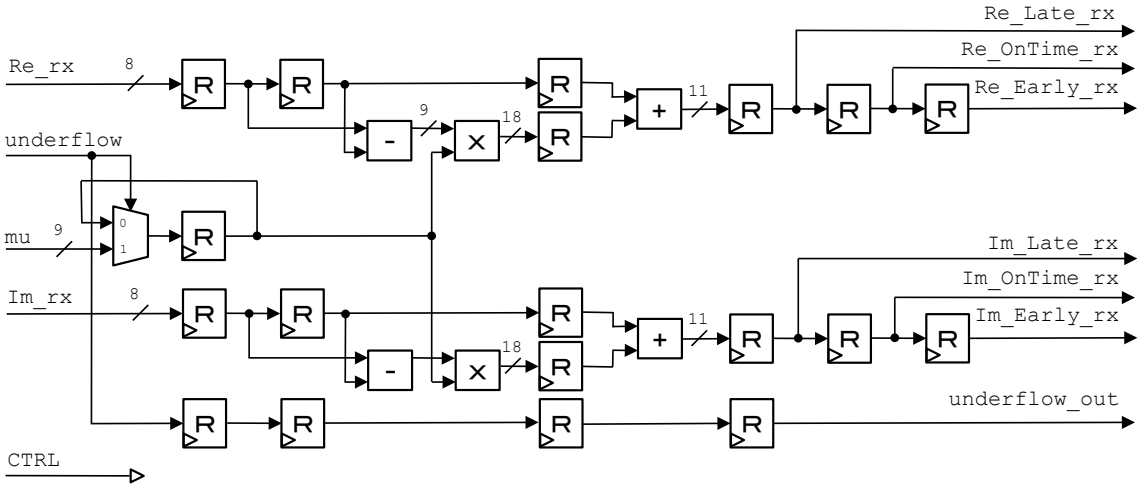

Figure 7.6. Hardware representation of the interpolator.

The information inputs `Re_rx` and `Im_rx` are interpolated as described in Eq. (5.4). In the case of the real input channel, the interpolation begins by computing the difference between the current input and the previous input. The result is multiplied by the registered value of `mu`, and then added with the previous input. The resulting value corresponds to the late interpolated sample `Re_Late_rx`, while the one-time and two-times delayed versions of this value correspond to the on-time sample `Re_OnTime_rx`, and the early sample `Re_Early_rx`, respectively. The interpolation process for samples in the imaginary input `Im_rx` is identical. The `underflow` strobe is propagated through the interpolator to later be used in the matched-filters bank.

7.3 Timing Estimator

The timing estimator is comprised by the timing loop filter and the modulo-1 decrementing counter. It takes in the latest timing error signal from the TED, and produces the interpolation control signals `mu` and `underflow`.

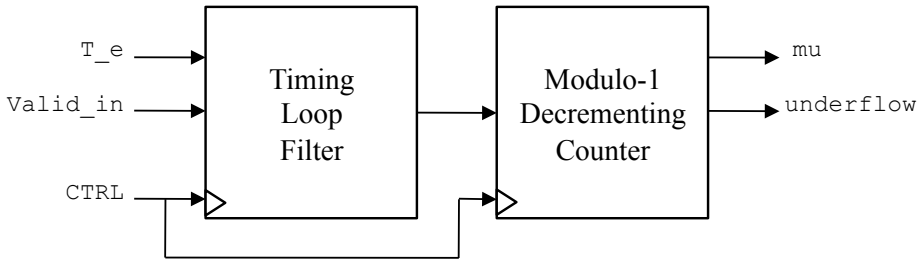

Figure 7.7. Block diagram of the timing estimator.

7.3.1 Timing Loop Filter

The timing loop filter (TLF) is responsible for adjusting the timing error signal `T_e` that is produced by the TED. Since the timing synchronizer is based on a first-order phase-locked loop (PLL), the adjustment corresponds to a simple gain. The inputs to the TLF are:

- `T_e`
- `Valid_in`
- `CTRL`.

And the outputs of the TLF are:

- `TLF_out`
- `Valid_out`

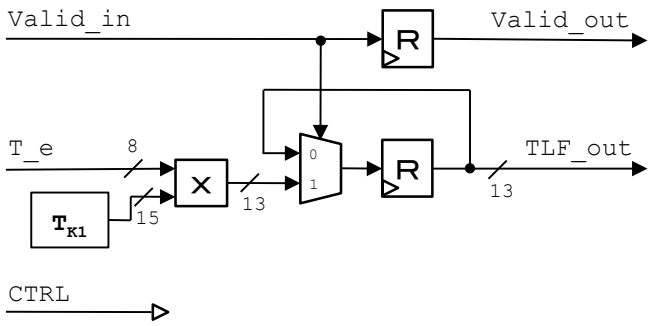

Figure 7.8. Hardware representation of the timing loop filter.

A hardware representation of the TLF is shown in Fig. 7.8. The gain operation performed by the TLF is implemented with a multiplication between the timing error signal `T_e` and the PLL constant $\mathtt{T_{K1}}$. The TLF output `TLF_out` is updated only when the input valid signal `Valid_in` is set to one. `Valid_in` is propagated through the TLF to later be used in the modulo-1 decrementing counter.

The value of the PLL constant $\mathtt{T_{K1}}$ is given by

$$T_{K1} = -0.0026/\pi \tag{7.2}$$

where the division by π is a normalizing factor required by the `Sine\Cosine` block in the phase corrector. The numerator is the same one described in Section 5.2.

7.3.2 Modulo-1 Decrementing Counter

The modulo-1 decrementing counter is responsible for providing the interpolator with the control signals necessary to compute samples at the optimum instances. These interpolation control signals are the fractional interval `mu`, and the

symbol boundary indicator underflow. The inputs to the decrementing counter are:

- TLF_in
- Valid_in
- CTRL.

And the outputs of the decrementing counter are:

- underflow
- mu

A hardware representation of the modulo-1 decrementing counter is shown in Fig. 7.9. According to Eq. (5.7), the counter's value decrements on every clock cycle by the net amount $\frac{1}{16}$ + TLF_in. The majority of the time, TLF_in is equal to zero, so, on average, the counter's value decrements by $\frac{1}{16}$. This is represented by the subtraction on the top part of the diagram. Parallel to this, the two subtractions below represent when the counter is also decremented by TLF_in. The second result is selected only when Valid_in is set to one. Although this design repeats the same subtraction operation twice, it allows us to produce outputs on every clock cycle. This is required by the interpolator as new data samples are clocked in at the same rate. The underflow strobe assumes the same value as the counter's sign bit. Therefore, when the counter's value becomes negative, or 'underflows', this signal is set to one.

The modulo-1 operation is implemented using a multiplexer indexed by the counter's only integer bit. When this bit is zero, it means one of two things: the counter's value is positive and less than one, or it is negative and smaller than -1.

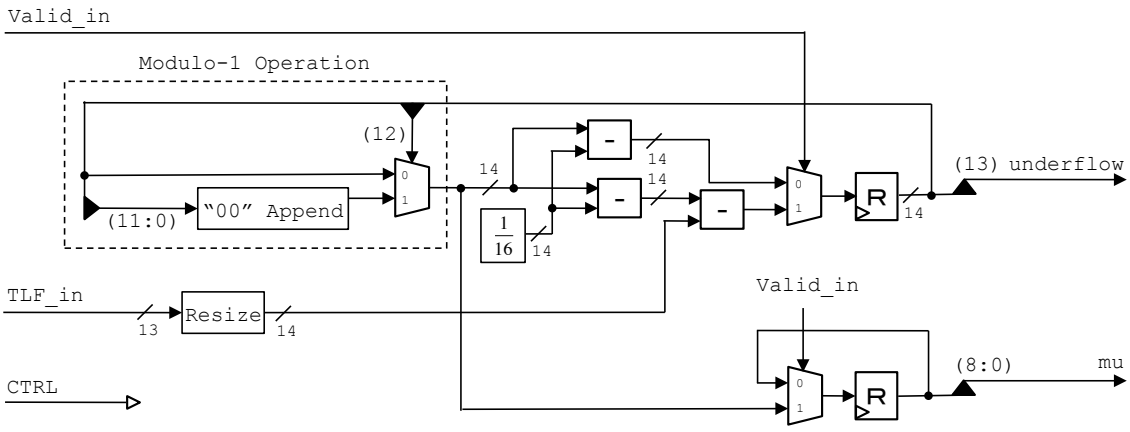

Figure 7.9. Hardware representation of the mod-1 decrementing counter.

The first case is possible, but the second one is not, because the counter is always made positive as soon as it goes negative. Therefore, when this bit is zero, the value of the counter is unchanged. However, when this bit is set to one, it means that the counter's value is greater than one, or that it has just become negative. In either case, the counter is updated to be the complement of the previous value, and so it becomes positive again. This complement operation is given by Eq. (5.8), and is achieved by appending two zeros at the beginning of the fractional part of the counter's value. The fractional interval `mu` corresponds to the value of the counter just before it becomes negative. This is represented with the lower nine bits to take into account the multiplication by $N = 16$, as indicated in Eq. (5.11).

7.4 Phase Corrector

The phase corrector is responsible for removing any residual phase error in the input data according to the information provided by the phase estimator. The inputs to the phase corrector are:

- `Re_OnTime_rx`, `Im_OnTime_rx`
- `Re_Early_rx`, `Im_Early_rx`
- `Re_Late_rx`, `Im_Late_rx`
- `VCO_in`
- `VCO_Valid_in`
- `underflow`
- `CTRL`.

And the outputs of the phase corrector are:

- `Re_OnTime_rx_out`, `Im_OnTime_rx_out`
- `Re_Early_rx_out`, `Im_Early_rx_out`
- `Re_Late_rx_out`, `Im_Late_rx_out`
- `underflow_out`

A hardware representation of the phase corrector is shown in Fig. 7.10. The phase corrector receives three sets of complex-valued samples corresponding to the on-time, early and late sampling instants from the interpolator. At this point, the on-time samples are assumed to be time-synchronized. However, all three

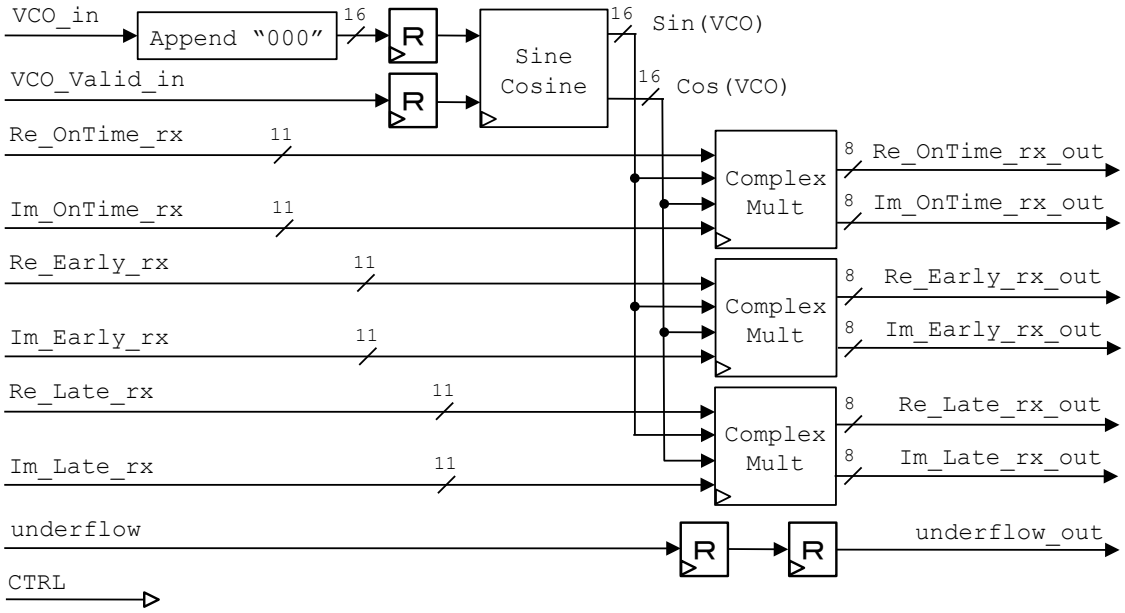

Figure 7.10. Hardware representation of the phase corrector.

sets of samples are still out of phase with the one used at the transmitter. An instantaneous estimate of this phase error is provided by the VCO, along with an input valid signal. The phase corrector rotates the phase of the input samples by the amount of `VCO_in` in order to remove the phase error. It does so by means of a `Sine\Cosine` block and three complex multipliers.

The `Sine\Cosine` block is used to compute the values of `Sin(VCO_in)` and `Cos(VCO_in)` to be provided to the complex multipliers. The implementation of this block is based on the work presented in [16], which takes advantage of the symmetric and periodic behavior of the two functions. The two function are approximated using piecewise polynomials, whose coefficients are used to index two look-up tables. Although complex, this approach results in a more precise output than one with two LUTs being directly indexed by the input angle.

A hardware representation of the complex multipliers is shown in Fig. 7.11. Three identical copies of the complex multiplier are needed, one for each set of

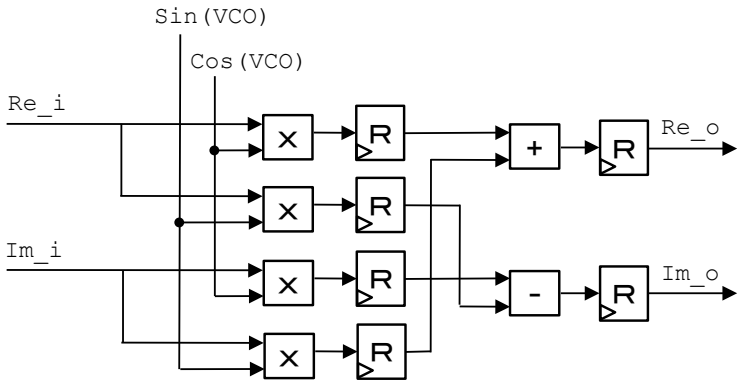

Figure 7.11. Hardware representation of the complex multiplier.

input samples. Each complex multiplier takes in four inputs to compute the phase rotation given by

$$(\texttt{Re\_i} + j\texttt{Im\_i}) * (\cos(\texttt{VCO\_in}) + j\sin(\texttt{VCO\_in})) \tag{7.3}$$

where `Re_i` and `Im_i` are the real and imaginary data samples for each sequence. This produces the three sets of complex-valued phase-corrected outputs shown in Fig 7.10. The `underflow` signal is only propagated through the design to be aligned with the data outputs, and later be used in the matched-filters.

7.5 Phase Estimator

The phase estimator is comprised by the phase loop filter and the voltage-controlled oscillator. It takes in the latest phase error signal from the PED, and produces an instantaneous estimate of the phase error.

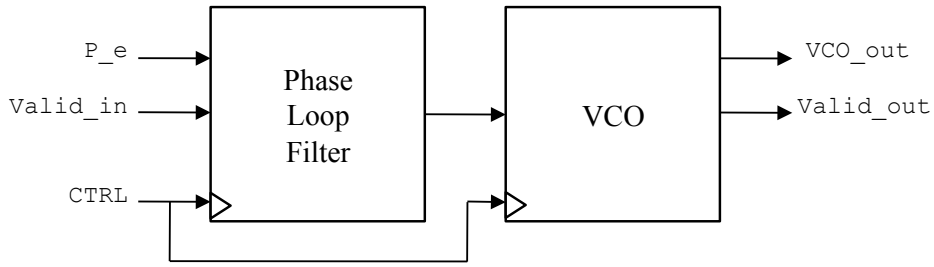

Figure 7.12. Block diagram of the phase estimator.

7.5.1 Phase Loop Filter

The phase loop filter (PLF) is responsible for adjusting the phase error signal `P_e` that is produced by the PED. Since the phase synchronizer is based on a first-order phase-locked loop (PLL), the adjustment corresponds to a simple gain. The inputs to the PLF are:

- `P_e`
- `Valid_in`
- `CTRL`.

And the outputs of the PLF are:

- `PLF_out`
- `Valid_out`

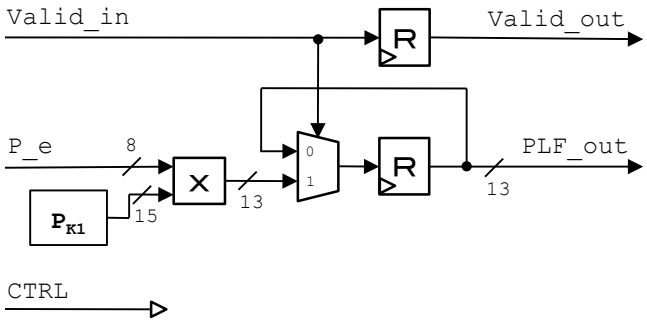

Figure 7.13. Hardware representation of the phase loop filter.

A hardware representation of the PLF is shown in Fig. 7.13. The gain operation performed by the PLF is implemented with a multiplication between the phase error signal `P_e` and the PLL constant $\mathtt{P_{K1}}$. The PLF output `PLF_out` is updated only when the input valid signal `Valid_in` is set to one. `Valid_in` is propagated through the PLF to later be used in the voltage-controlled oscillator.

The value of the PLL constant $\mathtt{P_{K1}}$ is given by

$$P_{K1} = 0.0026/\pi \tag{7.4}$$

where the division by π is a normalizing factor required by the `Sine\Cosine` block in the phase corrector. The numerator is the same one described in Section 6.2.

7.5.2 Voltage Controlled Oscillator

The voltage-controlled oscillator (VCO) is responsible for computing and storing the running sum of the adjusted phase error signal provided by the PLF. The inputs to the VCO are:

- `PLF_in`

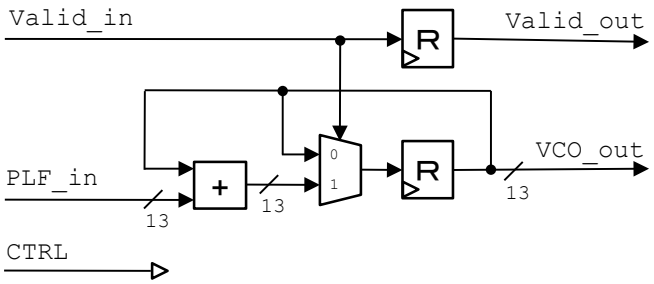

Figure 7.14. Hardware representation of the voltage-controlled oscillator.

- `Valid_in`
- `CTRL`.

And the outputs of the VCO are:

- `VCO_out`
- `Valid_out`

A hardware representation of the VCO is shown in Fig. 7.14. The VCO receives the PLF output signal `PLF_in`, and its associated input valid signal `Valid_in`, to compute the running sum `VCO_out`. Since `PLF_in` is relatively small and alternates between positive and negative values, there is no risk of the accumulator overflowing. Finally, `Valid_in` is propagated to the output to indicate when `VCO_out` has changed.

7.6 MFs Bank

The matched-filters bank is responsible for implementing the two matched-filters required for values of $\tilde{\alpha}_k = \pm 1$, and the accumulator for $\tilde{\alpha}_k = 0$. It does so by employing two parallel multiply-and-accumulate systems, which reduce hardware utilization, but as a tradeoff, increase the design's complexity. The inputs to the MFs bank are:

- `Re_rx`, `Im_rx`
- `underflow`
- `CTRL`.

And the outputs of the MFs bank are:

- `Re_+1_MFo`, `Im_+1_MFo`
- `Re_-1_MFo`, `Im_-1_MFo`
- `Re_0_MFo`, `Im_0_MFo`
- `TI_out`
- `Valid_out`

A hardware representation of the MFs bank is shown in Fig. 7.15. The MFs bank receives each set of complex-valued data samples from the phase corrector, as well as the propagated `underflow` strobe, and generates three complex-valued matched-filter outputs corresponding to the three possible transmitted symbols -1, 0, and +1. There are three sets of complex-valued data samples produced by the phase corrector: on-time, early and late; therefore, three identical copies of

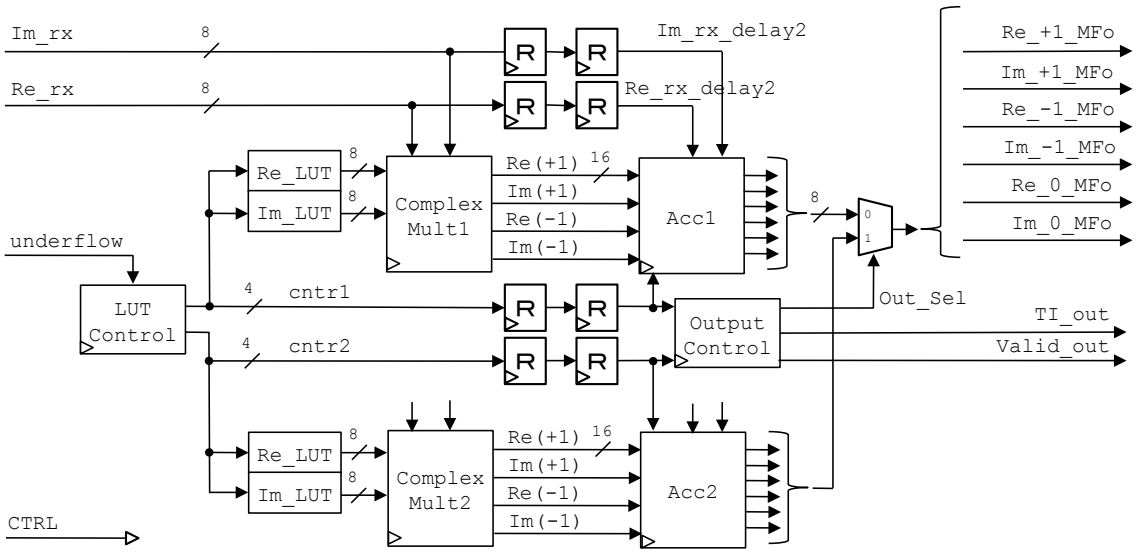

Figure 7.15. Hardware representation of the matched-filters bank.

the MFs bank are used in the demodulator. It is important to note that `Re_rx` and `Im_rx` are not the information inputs even though they are referred to with the same name here.

The matched-filtering operation given by Eq. (4.3) is implemented in a sample-by-sample basis using two parallel multiply-and-accumulate systems. The sample-by-sample approach means that each complex-valued sample of the 16 corresponding to every symbol is filtered individually. A counter generated in the `LUT Control` block keeps track of the number of samples being processed. When this counter reaches a value of 16, the filtering operation is completed for the current symbol and restarted for the next one. However, because of small timing shifts introduced by the timing synchronizer, sometimes two consecutive symbols overlap and share up to one sample. This is when the second multiply-and-accumulate system comes into play so that the shared sample can be used in the computation of both symbols.

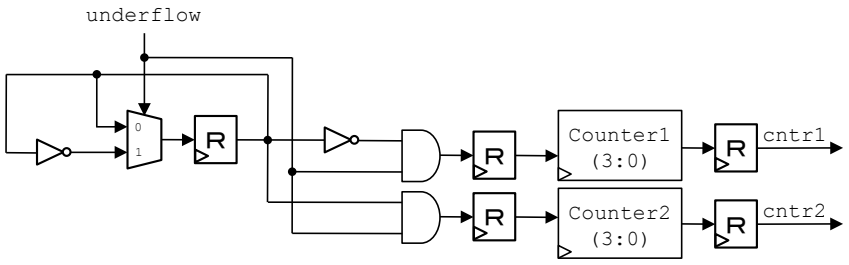

Figure 7.16. Hardware representation of the MFs LUT control system.

A hardware representation of the LUT control system is shown in Fig. 7.16. The LUT control system is triggered by the `underflow` strobe. When the first underflow occurs, it triggers `Counter2`, which starts counting. Even though the trigger signal only lasts for one clock cycle, the counter is programmed to keep increasing its value by one on every clock cycle until it overflows and becomes 0 again. At this point, the next underflow should occur and the counting operation is now switched to `Counter1`. Therefore, if an overlap of two consecutive symbols occurs, the two alternating counters can handle the situation.

Each multiply-and-accumulate system in the design consists of two look-up tables (LUTs), a complex-multiplier, and an accumulator. The LUTs are used to store the matched-filter coefficients $e^{-j2\pi h\tilde{\alpha}_k q_{\mathrm{PT}}(t)}$ for $\tilde{\alpha}_k = \pm 1$. Because of the trigonometric identities $\sin(-x) = -\sin(x)$ and $\cos(-x) = \cos(x)$, the imaginary coefficients for both values of $\tilde{\alpha}_k$ only vary by the sign, while the real coefficients are the same for both cases. Without loss of generality, we store the imaginary coefficients of $\tilde{\alpha}_k = -1$ in `Im_LUT`, and the real coefficients in `Re_LUT`. Both LUTs have a depth of 16 coefficients, and are indexed by one of the two counters generated in the `LUT control` block.

A hardware representation of the complex-multiplier in each parallel system is shown in Fig. 7.17. The complex multiplier takes in the complex-valued sam-

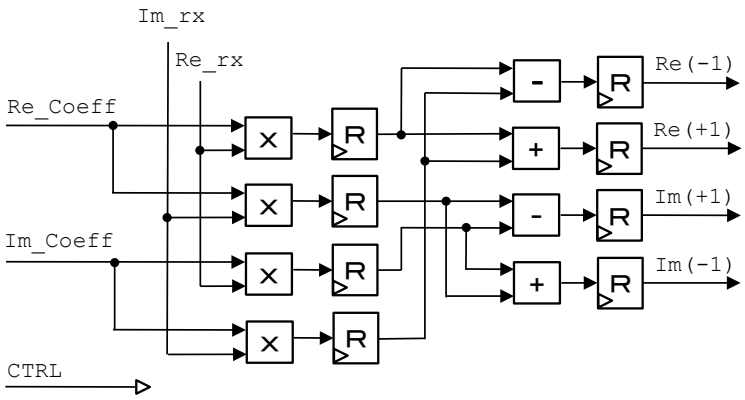

Figure 7.17. Hardware representation of the MFs complex multiplier.

ples `Re_rx` and `Im_rx`, and the real and imaginary coefficients from the LUTs. Because of the similarities in the coefficients described above, it only uses four real-valued multiplications, and appropriate reordering of the real and imaginary components to obtain the results. The outputs of the complex-multiplier and the two-time delayed versions of `Re_rx` and `Im_rx` are then fed to the accumulator.

A hardware representation of the accumulator in each parallel system is shown in Fig. 7.18. The accumulator is responsible for implementing the integral in Eq. (4.3). In discrete-time, the integral becomes a summation, and a period of one symbol time corresponds to 16 clock cycles. Therefore, the accumulator computes the running sum of each of its inputs over 16 clock cycles. The two-times delayed value of the counter indicates whether the sums need to be restarted or continue accumulating. When the counter is zero, the sums are restarted with the current inputs; otherwise, they are accumulated.

Finally, the results from one of the two parallel systems is selected for output. This is done by the `Output Control` block, which is represented in Fig. 7.19. The output control system uses the delayed counter values to make the selection.

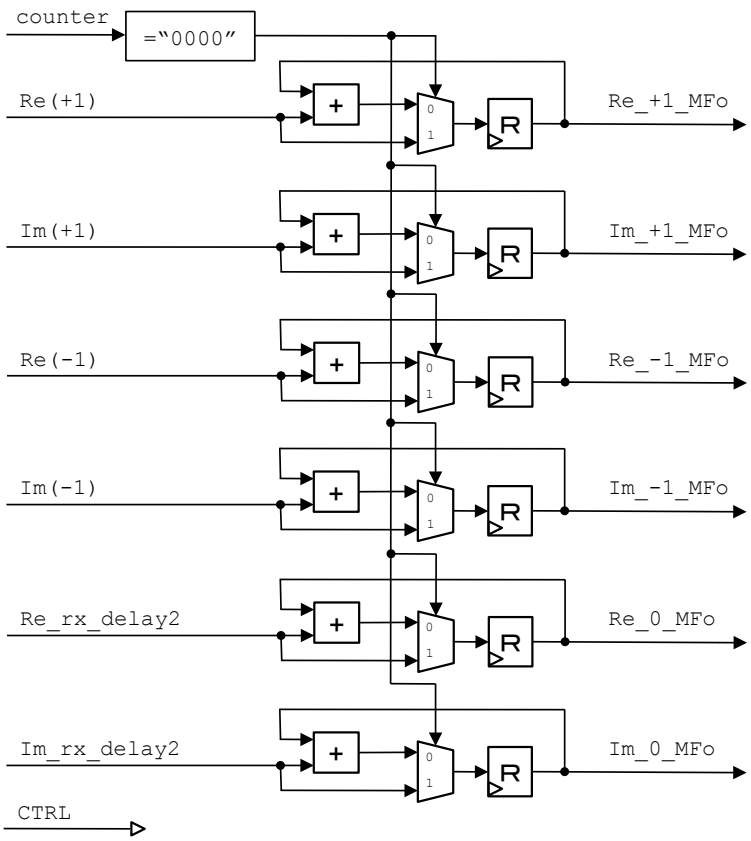

Figure 7.18. Hardware representation of the MFs accumulator.

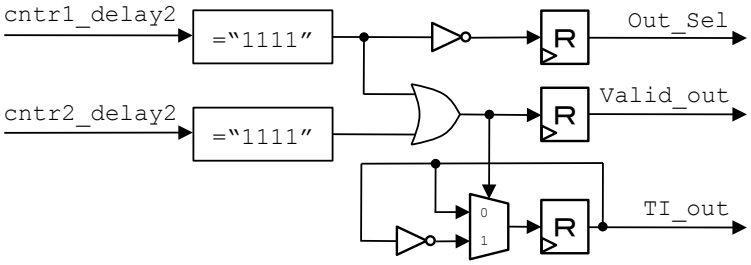

Figure 7.19. Hardware representation of the MFs output control system.

7.7 SOVA

The soft-output Viterbi algorithm (SOVA) decoder is responsible for estimating the hard-decisions and reliabilities associated with the maximum-likelihood path through the trellis. The inputs to the SOVA are:

- Re_+1_OnTime_MF, Im_+1_OnTime_MF
- Re_-1_OnTime_MF, Im_-1_OnTime_MF
- Re_0_OnTime_MF, Im_0_OnTime_MF
- Re_+1_Early_MF, Im_+1_Early_MF
- Re_-1_Early_MF, Im_-1_Early_MF
- Re_0_Early_MF, Im_0_Early_MF
- Re_+1_Late_MF, Im_+1_Late_MF
- Re_-1_Late_MF, Im_-1_Late_MF
- Re_0_Late_MF, Im_0_Late_MF
- TI_in, Valid_in
- CTRL

And the outputs of the SOVA are:

- Pu_O, Hu_O
- BI_1, BI_2, BI_3, BI_4, ..., BI_8
- Valid

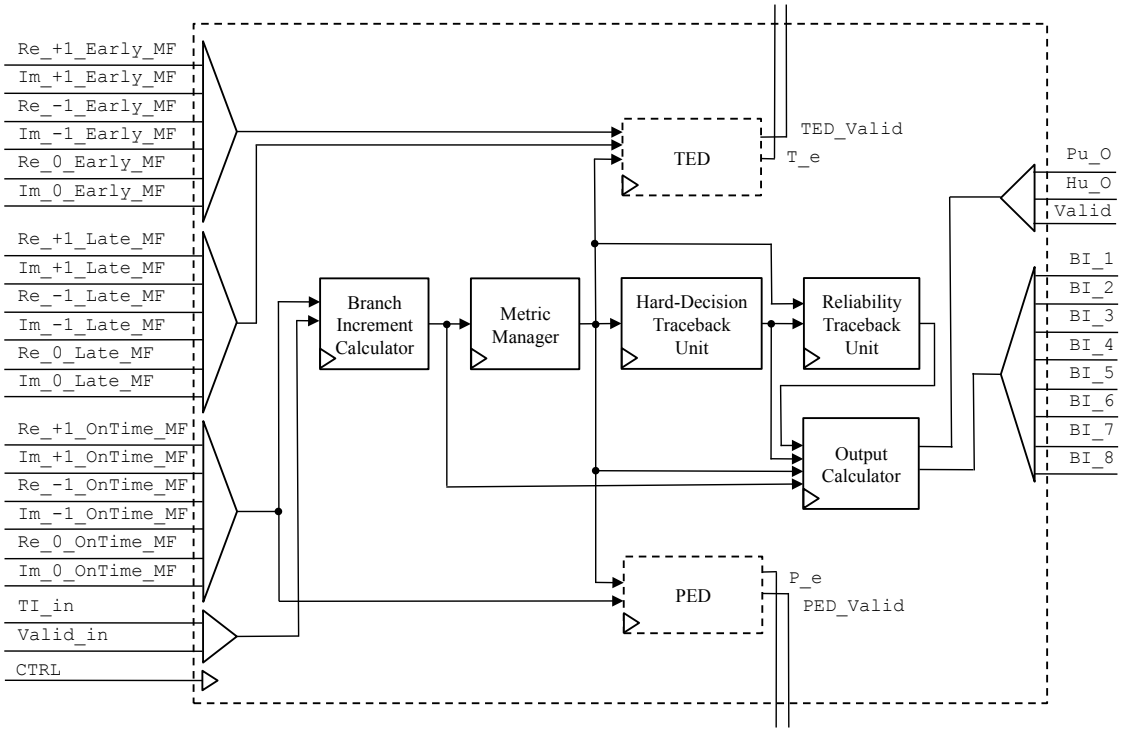

Figure 7.20. Hardware representation of the SOVA decoder.

In order to handle all the needed operations in the decoding process, the SOVA decoder has been broken down into seven individual units, each responsible for a separate task. This structure can be seen in Fig. 7.20. A description of each individual unit is provided below.

- *Branch Increment Calculator.* The branch increment calculator converts the output of the on-time MFs bank into the real-valued probabilities associated with the input bit in each branch of the trellis. The resulting eight branch increments, one for each branch in the trellis, represent the standard input to the SOVA decoder as described in Chapter 4.

- *Metric Manager.* The metric manager computes and updates the cumulative metrics at each trellis state based on the provided branch increments. It also determines the global winning state, the winning branch indexes, and the differences between path metric candidates (Δ) at each decoding step. This information is provided to all other units in the decoder.

- *Timing Error Detector.* The TED is an essential component of the timing synchronizer; however, in hardware, it is shown as part of the SOVA decoder. This is due to the fact that some of its necessary inputs are originated here. The TED uses the early and late MFs outputs, and the winning branch indexes to compute the next timing error signal. A hardware description of the TED is given in Section 7.8.

- *Phase Error Detector.* The PED is an essential component of the phase synchronizer; however, in hardware, it is shown as part of the SOVA decoder. This is due to the fact that some of its necessary inputs are originated here. The PED uses the on-time MFs output, and the winning branch indexes to compute the next phase error signal. A hardware description of the PED is given in Section 7.9.

- *Hard-Decision Traceback Unit.* The hard-decision traceback unit (HTU) updates the path decision vectors according to the indexes of the winning branches and the trellis indicator. It outputs the oldest path decisions in the decoding window, as well as comparisons of path decision candidates for each trellis state.

- *Reliability Traceback Unit.* The reliability traceback unit (RTU) updates the reliability vectors according to the indexes of the winning branches, the Δ values, the comparisons of path decision candidates, and the trellis indicator. Similarly to the HTU, it outputs the oldest reliabilities in the decoding window for each trellis state.

- *Output Calculator.* The output calculator accepts information from all the other pieces, and determines the best choice of hard-decision and reliability for each decoding step. It does so by using the global winning state determined by the metric manager. In addition, it delays the branch increments corresponding to each decoded information so they are aligned at the output. The branch increments are used as information inputs to the simple version of the demodulator in the following iterations of the SCCC decoding scheme.

7.7.1 Branch Increment Calculator

The branch increment calculator is responsible for providing the metric manager with the eight real-valued probabilities associated with the input bit in each branch of the trellis. The inputs to the branch increment calculator are:

- `Re_+1_OnTime_MF`, `Im_+1_OnTime_MF`
- `Re_-1_OnTime_MF`, `Im_-1_OnTime_MF`
- `Re_0_OnTime_MF`, `Im_0_OnTime_MF`
- `TI_in`, `Valid_in`
- `CTRL`

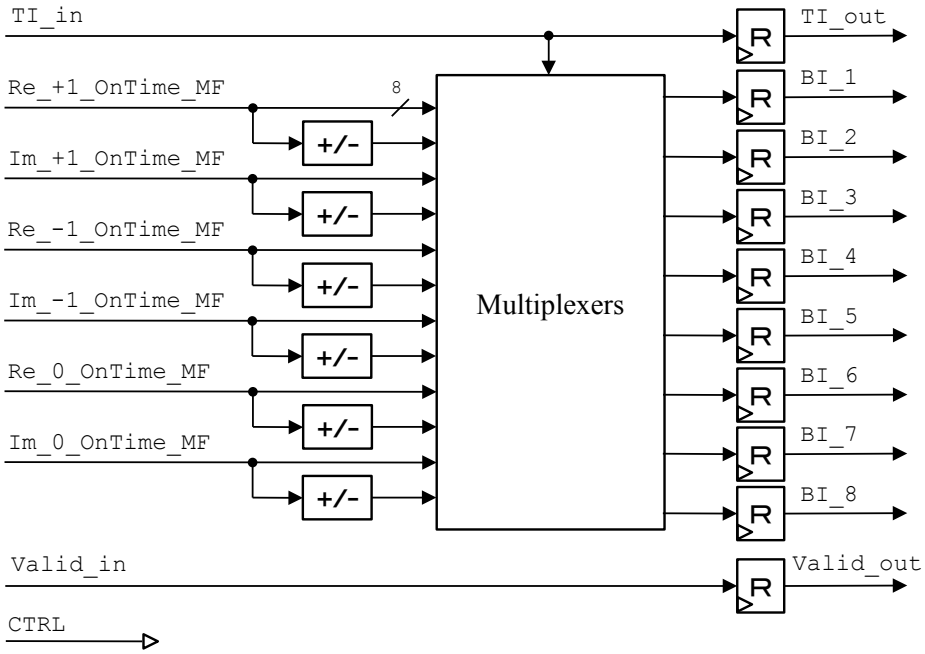

Figure 7.21. Hardware representation of the branch increment calculator.

And the outputs of the branch increment calculator are:

- `BI_1`, `BI_2`, `BI_3`, `BI_4`, ..., `BI_8`
- `TI_out`, `Valid_out`

A hardware representation of the branch increment calculator is shown in Fig. 7.21. The branch increment calculator receives the complex-valued outputs of the on-time MFs bank, and computes their sign-altered counterparts. Both sets of data are fed to a bank of multiplexers, which map the values of the branch increments according to what part of the trellis is being used. The bank of multiplexers contains six 2-by-1 multiplexers, each one indexed by the trellis indicator `TI_in`. The mapping performed by the multiplexers is shown in Table 7.2, and follows Eq. (4.4).

Table 7.2. Mapping of branch increments according to TI.

TI	BI_1
0	-Im_0_OnTime_MF
1	-Im_0_OnTime_MF

TI	BI_2
0	-Im_+1_OnTime_MF
1	-Im_-1_OnTime_MF

TI	BI_3
0	-Re_0_OnTime_MF
1	-Re_+1_OnTime_MF

TI	BI_4
0	-Re_-1_OnTime_MF
1	-Re_0_OnTime_MF

TI	BI_5
0	Re_-1_OnTime_MF
1	Re_0_OnTime_MF

TI	BI_6
0	Re_0_OnTime_MF
1	Re_+1_OnTime_MF

TI	BI_7
0	Im_+1_OnTime_MF
1	Im_-1_OnTime_MF

TI	BI_8
0	Im_0_OnTime_MF
1	Im_0_OnTime_MF

7.7.2 Metric Manager

The metric manager is responsible for updating, comparing, and storing the cumulative metrics at every decoding step. The inputs to the metric manager are:

- BI_1, BI_2, BI_3, BI_4, ..., BI_8
- TI_in, Valid_in
- CTRL

And the outputs of the metric manager are:

- w1, w2, w3, w4
- d1, d2, d3, d4
- gmax
- TI_out, Valid_out

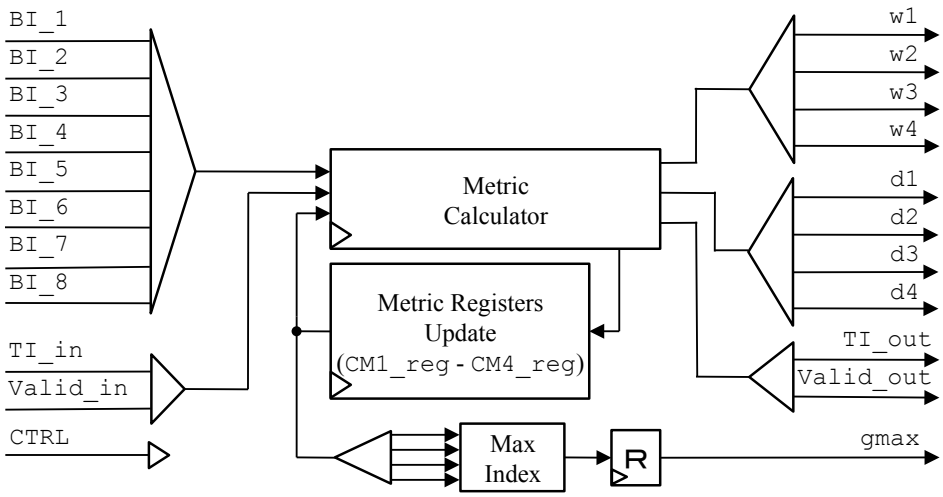

Figure 7.22. Hardware representation of the metric manager.

A hardware representation of the metric manager is shown in Fig. 7.22. The metric manager contains four registers, `CM1_reg-CM4_reg`, which store the cumulative metric values of each trellis state, respectively. On every decoding step, the metric calculator uses the provided eight branch increments BI_1, ..., BI_8 and the registered cumulative metric values CM1_reg, ..., CM4_reg to compute a new set of four cumulative metrics. The newly computed metrics are stored by the metric registers update unit in a way that allows the decoding operation to continue indefinitely without overflowing. The four registered cumulative metric values are fed to a max index unit, which determines the index $(0-3)$ of the current maximum registered metric.

A hardware representation of the metric calculator unit is shown in Fig. 7.23. The metric calculator computes a new set of cumulative metrics based on Eq. (4.5) and Eq. (4.6). According to Eq. (4.5), there are two branch metric candidates entering each of the four trellis states. The eight branch metric candidates are the result of the addition between the previous cumulative metric and the branch

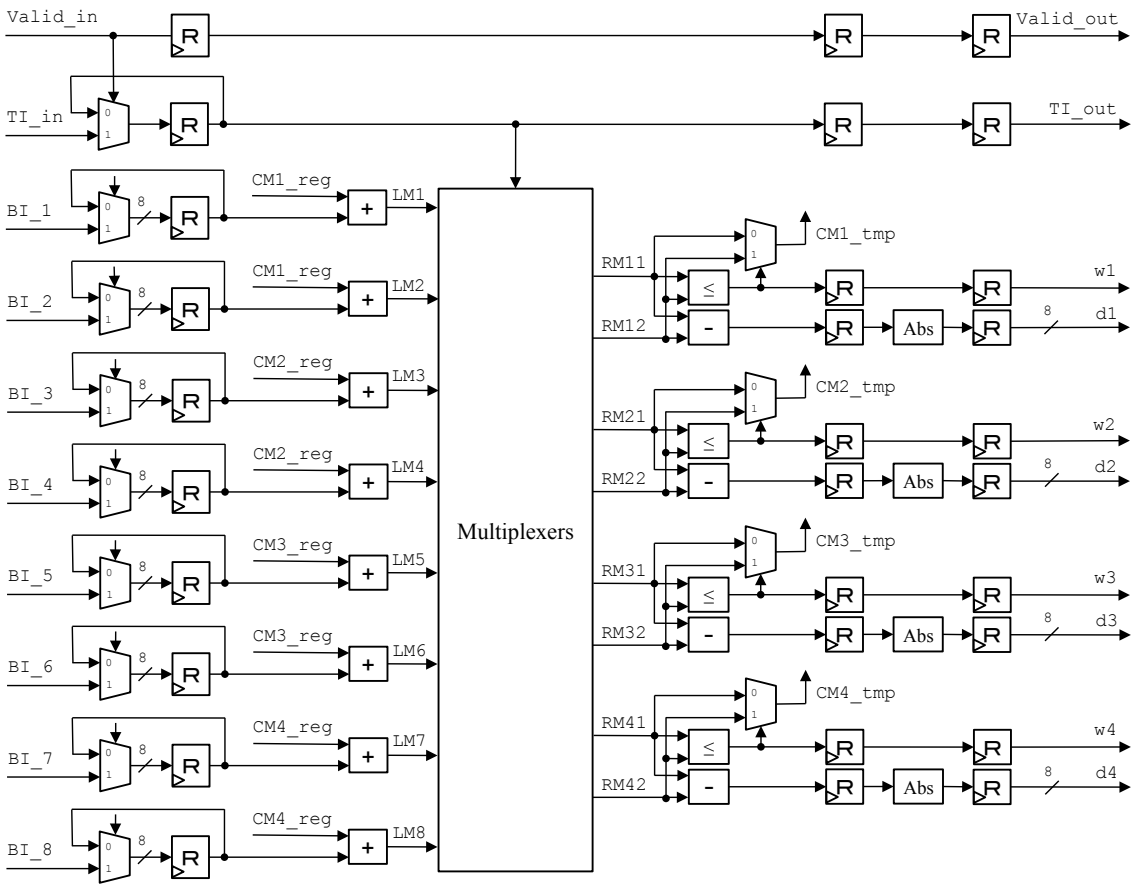

Figure 7.23. Hardware representation of the metric calculator.

increment corresponding to that ending state. In Fig. 7.23, these results are referred to as `LM*`, which represents the branch metric candidate * as seen from the left-hand side of the trellis. In order to determine the two branch metric candidates entering each trellis state, we use a bank of multiplexers. The bank of multiplexers contains six 2-by-1 multiplexers, each one indexed by the trellis indicator. The mapping performed by the multiplexers is shown in Table 7.24.

The two branch metric candidates entering each trellis state are referred to as `RM*1` and `RM*2`, where `RM*1` corresponds to the branch arriving to state * from above, and `RM*2` corresponds to the one arriving from below. The winning

Table 7.3. Mapping of branch metric candidates according to `TI`.

TI	RM11
0	LM1
1	LM1

TI	RM12
0	LM5
1	LM3

TI	RM21
0	LM3
1	LM2

TI	RM22
0	LM7
1	LM4

TI	RM31
0	LM2
1	LM5

TI	RM32
0	LM6
1	LM7

TI	RM41
0	LM4
1	LM6

TI	RM42
0	LM8
1	LM8

candidates are chosen according to Eq. (4.6) using four comparators and four multiplexers. Each comparator's binary output becomes the selector signal for the corresponding multiplexer, which outputs the next cumulative metric value. In addition, the comparator outputs represent the winning branch indexes `(w1-w4)`. Finally, the Δ values `(d1-d4)` are computed according to Eq. (4.7). Each Δ value is the positive difference between the two branch metric candidates entering each state.

A hardware representation of the metric registers update unit is shown in Fig. 7.24. The metric registers update unit receives the four newly computed cumulative registers and applies a mask on them before storing the values. The mask serves the purpose of resetting bit number 16 of the metrics, when all the four registered metrics have already reached this value. This guarantees that the decoding operation can be carried on for an indefinite amount of time without overflowing the cumulative metric registers.

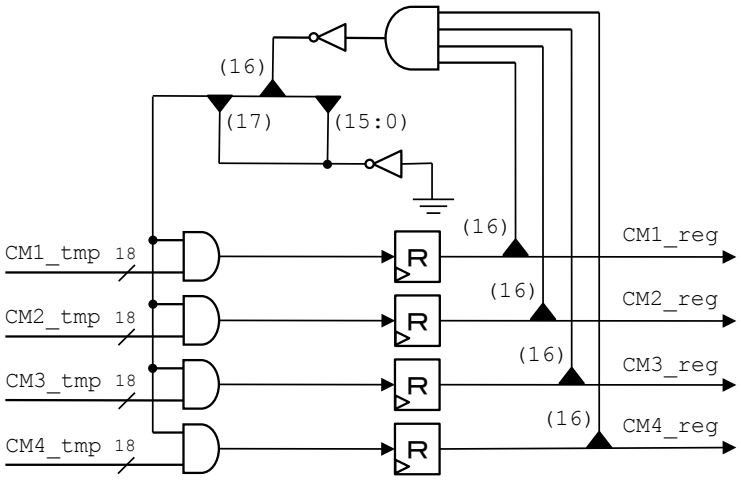

Figure 7.24. Hardware representation of the metric registers update unit.

7.7.3 Hard-Decision Traceback Unit

The hard-decision traceback unit (HTU) is responsible for updating the path-decision vectors on every decoding step. The inputs to the HTU are:

- `w1`, `w2`, `w3`, `w4`
- `TI_in`, `Valid_in`
- `CTRL`

And the outputs of the HTU are:

- `u_hat1`, `u_hat2`, `u_hat3`, `u_hat4`
- `u_vector_xor1`, `u_vector_xor2`, `u_vector_xor3`, `u_vector_xor4`
- `TI_out`, `Valid_out`

A hardware representation of the HTU is shown in Fig. 7.25. The HTU is implemented using a method of traceback called register exchange, with a decoding window length of $T = 16$. In this method, four length-$T - 1$ registers are

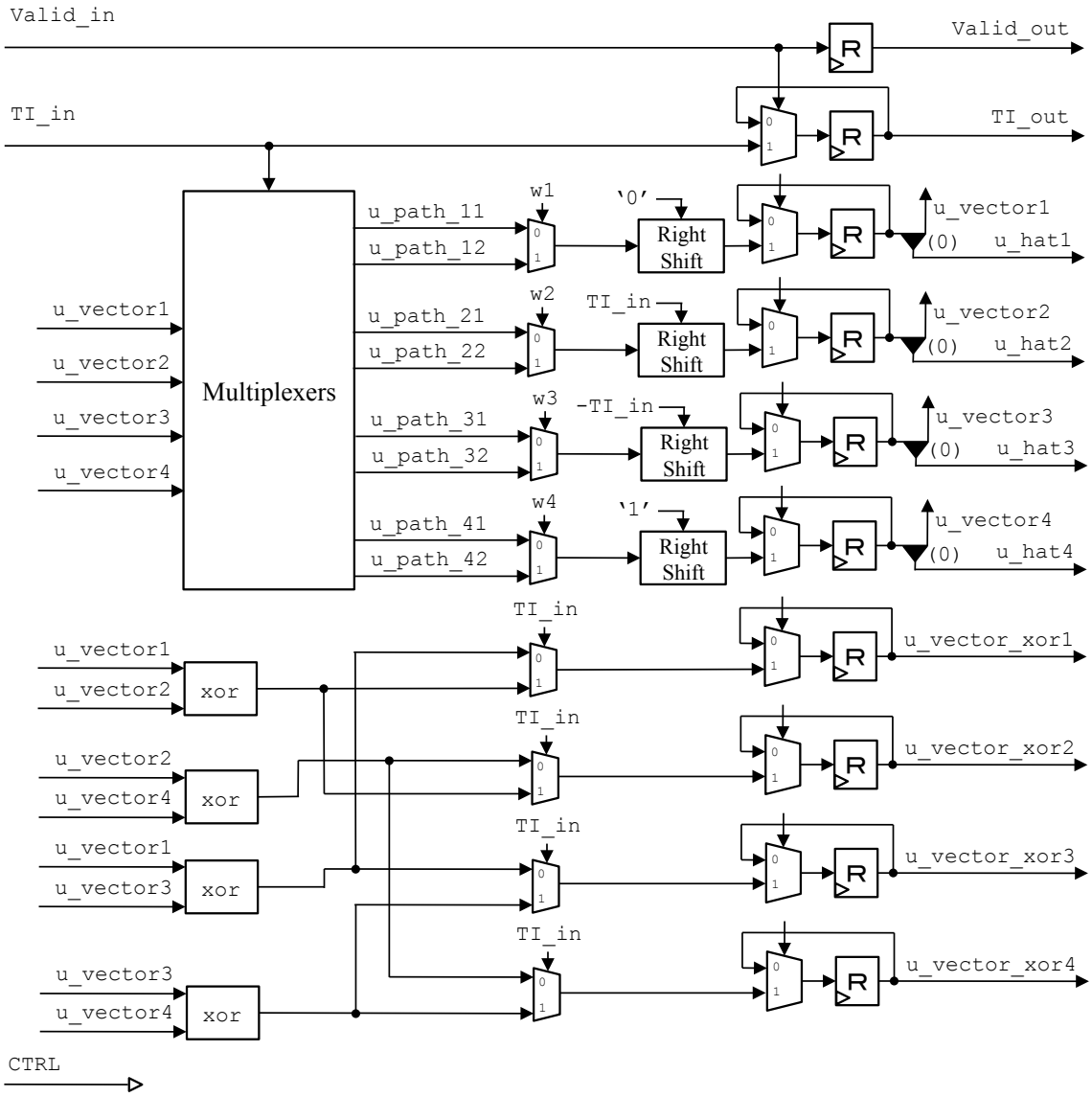

Figure 7.25. Hardware representation of the hard-decision traceback unit.

used to keep track of the newest hard-decisions associated with the maximum-likelihood path ending at each trellis state. In Fig. 7.25, these registers are labeled `u_vector1`, ..., `u_vector4`, and are referred to as path-decision vectors. On every decoding step, two competing paths merge into the same ending state. The path-decision vector corresponding to the winning branch metric candidate

gets copied into the path-decision vector of that ending state. And the $\tilde{u}_k$ value associated with the winning branch gets shifted in as the newest hard-decision of the vector.

Table 7.4. Mapping of merging path-decision vectors according to `TI`.

TI	u_path_11
0	u_vector1
1	u_vector1

TI	u_path_12
0	u_vector3
1	u_vector2

TI	u_path_21
0	u_vector2
1	u_vector1

TI	u_path_22
0	u_vector4
1	u_vector2

TI	u_path_31
0	u_vector1
1	u_vector3

TI	u_path_32
0	u_vector3
1	u_vector4

TI	u_path_41
0	u_vector2
1	u_vector3

TI	u_path_42
0	u_vector4
1	u_vector4

In the HTU, a bank of multiplexers is used to determine the two competing paths merging at each state. The bank of multiplexers contains six 2-by-1 multiplexers, each one indexed by the trellis indicator. The mapping performed by the multiplexers follows each branch in the trellis, and is shown in Table 7.4. The winning branch indexes `(w1-w4)` provided by the metric manager, determine which one of the two competing path-decision vectors to copy for the next set of states. The hard-decision value to be shifted in is a constant for the first and last ending states. However, for the two middle ending states, this value depends on the trellis indicator, as shown in Fig. 7.25. The shift in values are right-shifted into the vectors that were selected by the winning branch indexes. The results of the shifting operations become the path-decision vectors for the next decoding

steps, and the oldest bit of each vector is used as the estimated hard-decision output.

In addition, the HTU computes four registers of comparison bits, `u_vector_xor1`, ..., `u_vector_xor4`, one for each pair of merging path-decision vectors. There are only four possible pairs of merging paths, given by the structure of the trellis. Each merging paths comparison is implemented by means of an xor operation, which returns zero if the inputs are equal, and one if they are different. Four multiplexers indexed by the trellis indicator are used to select the comparison bit registers associated with each decoding step. This comparison bit vectors are necessary to update the reliabilities in the reliability traceback unit.

7.7.4 Reliability Traceback Unit

The reliability traceback unit (RTU) is responsible for updating the set of reliabilities associated with the hard-decisions generated by the HTU. The inputs to the RTU are:

- `w1`, `w2`, `w3`, `w4`
- `d1`, `d2`, `d3`, `d4`
- `u_xor1`, `u_xor2`, `u_xor3`, `u_xor4`
- `TI_in`, `Valid_in`
- `CTRL`

And the outputs of the RTU are:

- `L_hat1`, `L_hat2`, `L_hat3`, `L_hat4`
- `Valid_out`

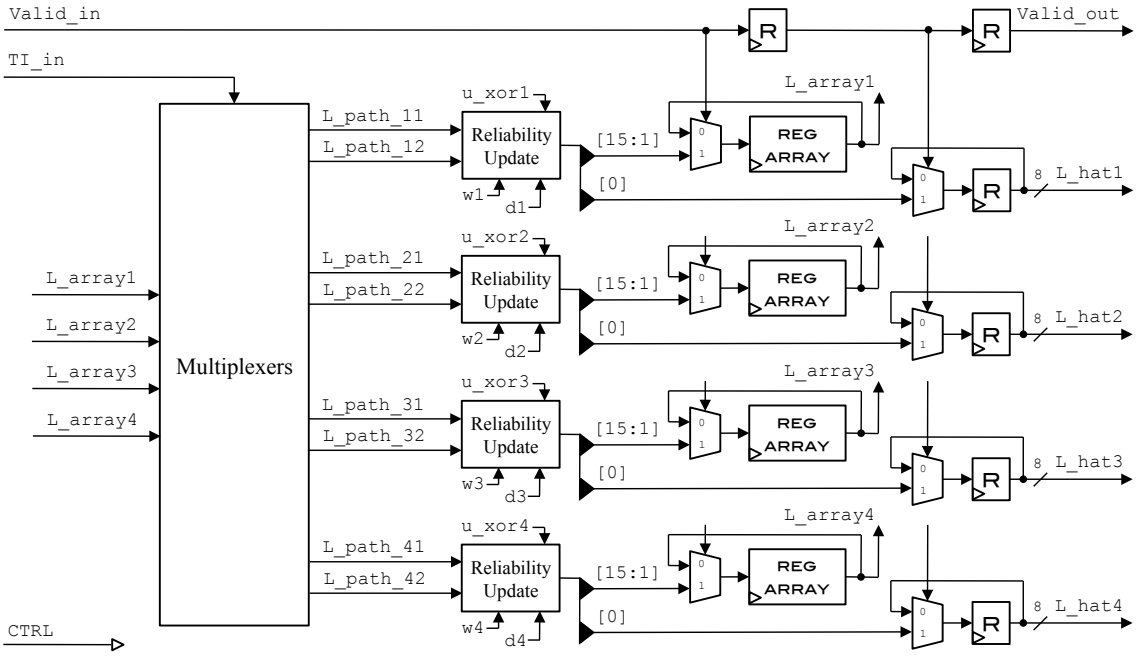

Figure 7.26. Hardware representation of the reliability traceback unit.

A hardware representation of the RTU is shown in Fig. 7.26. The RTU is implemented using the same method of traceback as the HTU. The only difference is that instead of maintaining four lenght-$T-1$ registers, the RTU keeps track of four length-$T-1$ arrays of 8-bit registers. This is because each reliability is represented with eight bits, while in the HTU each hard-decision is represented with just one bit.

In a similar way as before, on every decoding step there are two merging paths ending on the same trellis state. Each merging path has an array of reliabilities associated with the hard-decisions in its path-decision vector. These arrays of reliabilities are labeled as `L_array1`, ..., `L_array4` as shown in Fig. 7.26. In order to determine the two merging sets of reliabilities at each state, a bank of multiplexers is used. The bank of multiplexers contains six 2-by-1 multiplexers, each one indexed by the trellis indicator. The mapping performed by the multi-

plexers is shown in Table 7.5. As it can be noticed, this is the same mapping as the one used for merging path-decision vectors in the HTU.

Table 7.5. Mapping of merging reliability arrays according to `TI`.

TI	L_path_11
0	L_vector1
1	L_vector1

TI	L_path_12
0	L_vector3
1	L_vector2

TI	L_path_21
0	L_vector2
1	L_vector1

TI	L_path_22
0	L_vector4
1	L_vector2

TI	L_path_31
0	L_vector1
1	L_vector3

TI	L_path_32
0	L_vector3
1	L_vector4

TI	L_path_41
0	L_vector2
1	L_vector3

TI	L_path_42
0	L_vector4
1	L_vector4

The two merging reliability arrays for each trellis state are passed into the reliability update units, where the winning branch indexes `(w1-w4)`, Δ values `(d1-d3)`, and bit comparisons `(xor1-xor4)` determine the way the reliabilities should be updated. A hardware representation of the reliability update unit is shown in Fig. 7.27. Using the provided information, the reliabilities are updated following Eq. (4.8) and Eq. (4.9), for the cases with different and equal hard-decision estimates, respectively. The $T-1$ most significant locations of the updated reliability arrays are stored in the reliability vectors for the next decoding steps. And the oldest reliability value from each array is used as the output.

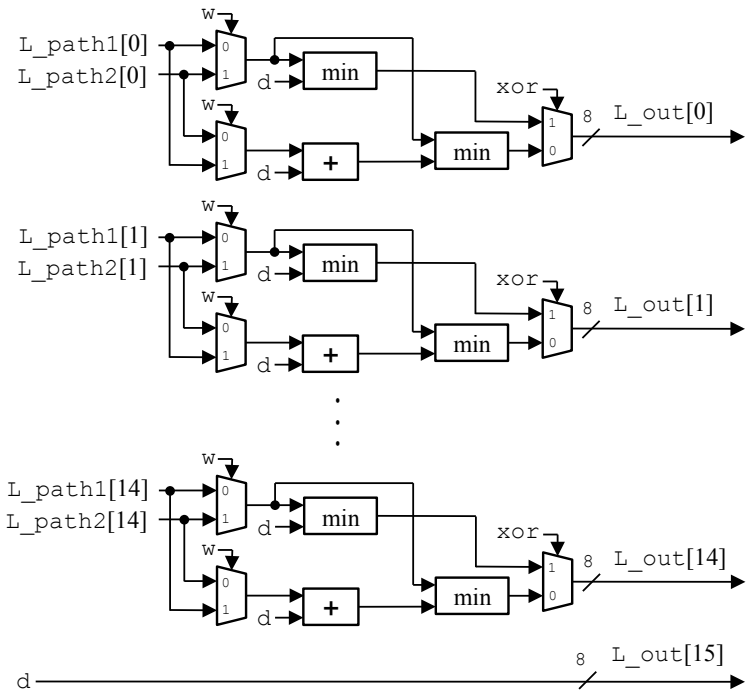

Figure 7.27. Hardware representation of the reliability update unit.

7.7.5 Output Calculator

The output calculator is responsible for rearranging the final decoder output based on the information provided by all other units. The inputs the output calculator are:

- `u_hat1`, `u_hat2`, `u_hat3`, `u_hat4`
- `L_hat1`, `L_hat2`, `L_hat3`, `L_hat4`
- `BI_1_in`, `BI_2_in`, `BI_3_in`, ..., `BI_8_in`
- `gmax`
- `MM_Valid_in`, `RTU_Valid_in`, `BI_Valid_in`
- `CTRL`

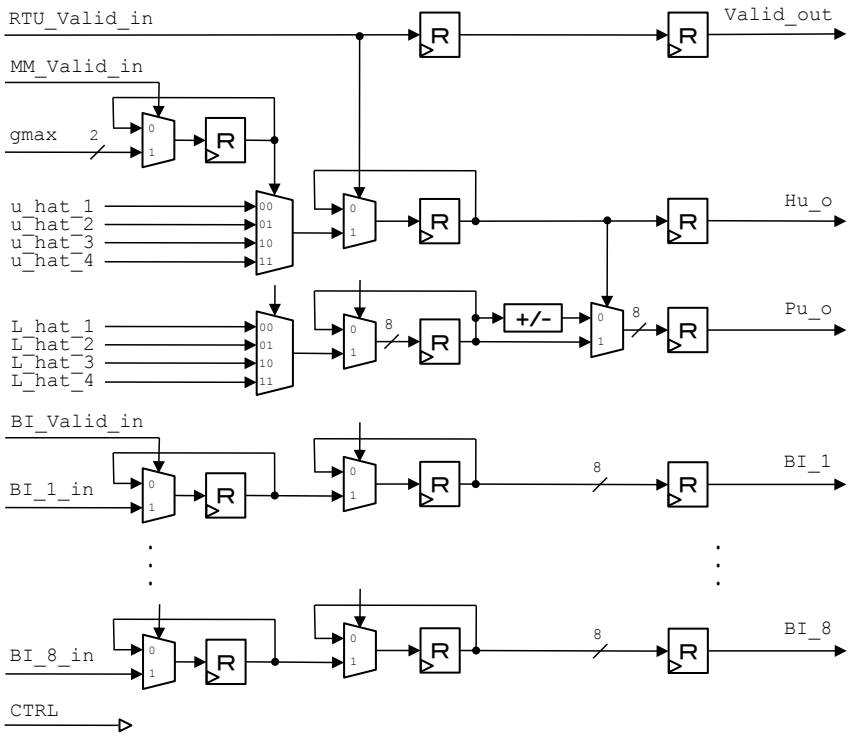

Figure 7.28. Hardware representation of the output calculator.

And the outputs of the output calculator are:

- `Hu_o`, `Pu_o`
- `BI_1`, `BI_2`, `BI_3`, ..., `BI_8`
- `Valid`

A hardware representation of the output calculator is shown in Fig. 7.28. The output calculator selects the best choice of hard-decision and reliability for each decoding step from the four available paths. The best choice is determined by the global maximum winning state `(gmax)`, which represents the path of maximum-likelihood. Then, the selected hard decision is used to apply the proper sign to the selected reliability, as it needs to be in antipodal form upon output. Finally, the

output calculator delays the branch increments that led to the current decoded information so that they are aligned in the output.

7.8 TED

The timing error detector (TED) is responsible for providing the TLF with the next estimate of the timing error signal. The inputs of the TED are:

- `Re_+1_Early_MF`, `Im_+1_Early_MF`
- `Re_-1_Early_MF`, `Im_-1_Early_MF`
- `Re_0_Early_MF`, `Im_0_Early_MF`
- `Re_+1_Late_MF`, `Im_+1_Late_MF`
- `Re_-1_Late_MF`, `Im_-1_Late_MF`
- `Re_0_Late_MF`, `Im_0_Late_MF`
- `w1_in`, `w2_in`, `w3_in`, `w4_in`
- `gmax_in`
- `TI_in`, `Valid_in`
- `CTRL`

And the outputs of the TED are:

- `T_e`
- `Valid_out`

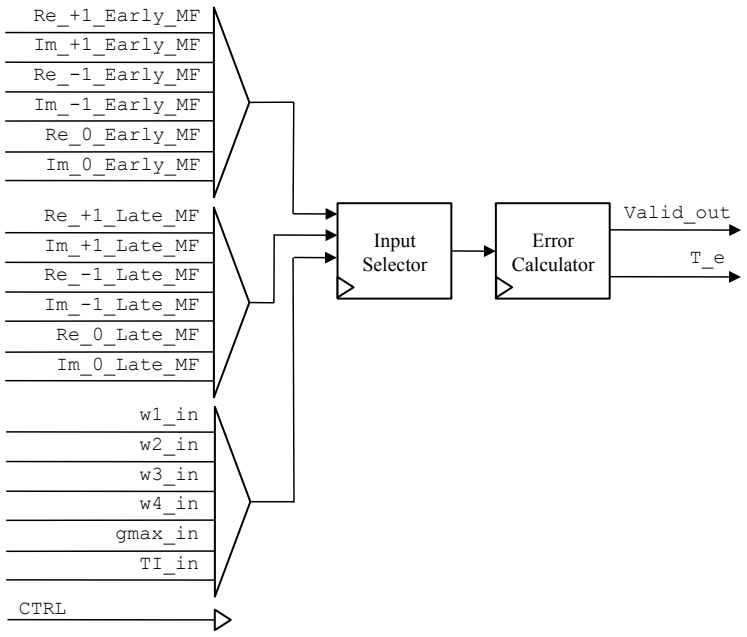

Figure 7.29. Block diagram of the timing error detector.

A block diagram representation of the TED is shown in Fig. 7.29. The internal structure of the TED has been broken down into two individual units. The first unit is the input selector, which reorders the input so that it is as expected by the error calculator. And the second unit is the error calculator, which computes the timing error estimate `T_e` from the path history of the global maximum winning state `gmax`.

A hardware representation of the input selector is shown in Fig. 7.30. The input selector receives the complex-valued output from the early and late MFs, and computes the input to be provided to the error calculator unit. The input expected by the error calculator is described by Eq. (5.3). According to this equation, the input to the error calculator is the real part of the multiplication between the derivative of the on-time MFs $\boldsymbol{Y}_k(\cdot)$, and the phase states $e^{-j\hat{\theta}_{k-D}}$. The derivative of the on-time MFs is obtained by computing the difference between

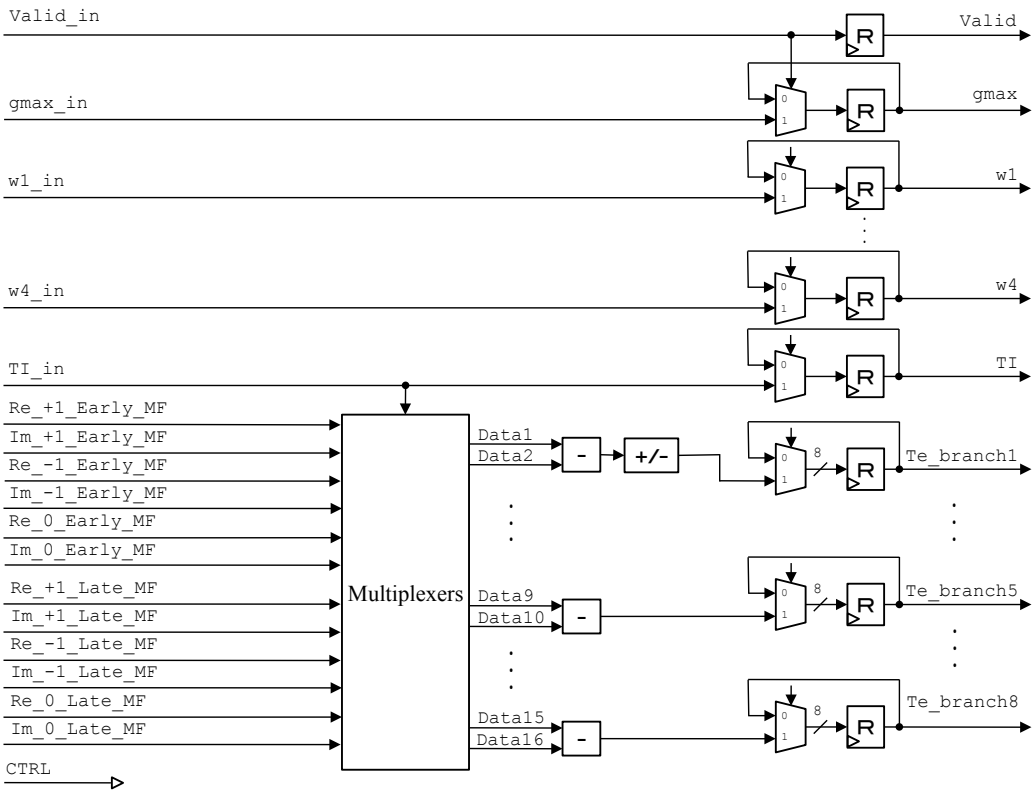

Figure 7.30. Hardware representation of the TED input selector.

every corresponding late and early MF output. And the phase states only assume $\pm 1, \pm j$ values, so the multiplication is implemented with a simple sign-alteration and reordering of the real and imaginary parts. When both operations described above are considered together, the output produced by Eq. (5.3) can be simply obtained with a set of eight subtractions. The operands in each subtraction are one from the late MFs output and the other from the early MFs output. Whether the real or imaginary part of the MFs output is used depends on the trellis indicator and the effects of applying the $Re(\cdot)$ operation and multiplying by $\pm 1, \pm j$. The mapping of operands necessary to obtain the result described by Eq. (5.3) is shown in Table. 7.6. This mapping is performed by a bank of twelve 2-by-1 multiplexers indexed by the trellis indicator.

Table 7.6. Mapping of subtraction operands according to `TI`.

TI	Data1
0	Im_0_Late_MF
1	Im_0_Late_MF

TI	Data2
0	Im_0_Early_MF
1	Im_0_Early_MF

TI	Data3
0	Im_+1_Late_MF
1	Im_−1_Late_MF

TI	Data4
0	Im_+1_Early_MF
1	Im_−1_Early_MF

TI	Data5
0	Re_0_Late_MF
1	Re_+1_Late_MF

TI	Data6
0	Re_0_Early_MF
1	Re_+1_Early_MF

TI	Data7
0	Re_−1_Late_MF
1	Re_0_Late_MF

TI	Data8
0	Re_−1_Early_MF
1	Re_0_Early_MF

TI	Data9
0	Re_−1_Late_MF
1	Re_0_Late_MF

TI	Data10
0	Re_−1_Early_MF
1	Re_0_Early_MF

TI	Data11
0	Re_0_Late_MF
1	Re_+1_Late_MF

TI	Data12
0	Re_0_Early_MF
1	Re_+1_Early_MF

TI	Data13
0	Im_+1_Late_MF
1	Im_−1_Late_MF

TI	Data14
0	Im_+1_Early_MF
1	Im_−1_Early_MF

TI	Data15
0	Im_0_Late_MF
1	Im_0_Late_MF

TI	Data16
0	Im_0_Early_MF
1	Im_0_Early_MF

A hardware representation of the error calculator is shown in Fig. 7.31. The error calculator receives the eight timing error estimates `Te_branch1`, ..., `Te_branch8`, one for each branch in the trellis, and performs the equivalent to two traceback operations to determine the next timing error signal `T_e`. This can also be understood as the TED having a delay of $D = 1$ in computing the timing error signal.

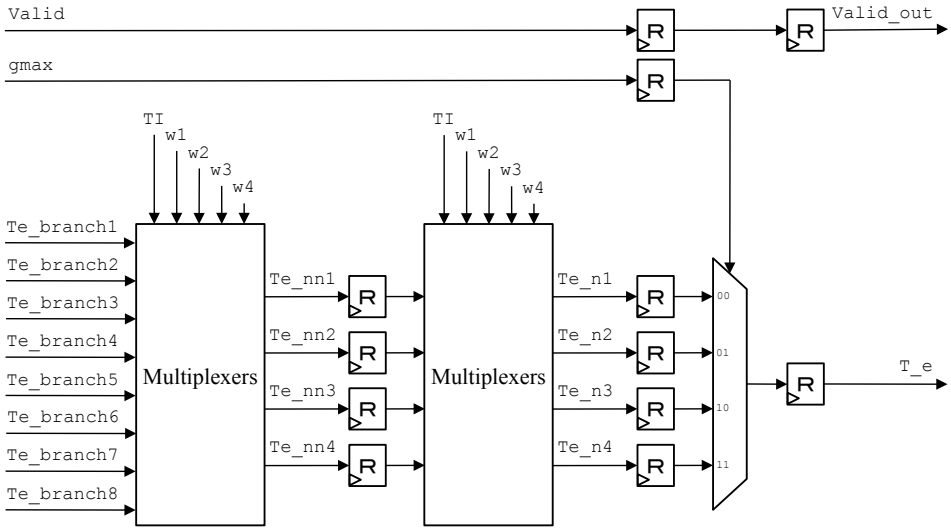

Figure 7.31. Hardware representation of the TED error calculator.

For the first traceback operation, the eight timing error estimates are reordered in pairs merging at the same ending state according to the trellis indicator. Then, using the winner branch indexes `(w1-w4)`, provided by the metric manager, only the timing error estimates corresponding to the winning branches are stored. The mapping performed by the first traceback operation is implemented by a bank of multiplexers. The bank of multiplexers contains four 4-by-1 multiplexers indexed by the winning branch indexes and the trellis indicator as shown in Table 7.7.

In a similar way, for the second traceback operation only the four surviving timing error estimates `Te_nn1`, ..., `Te_nn4` are reordered to their corresponding ending states according to the trellis indicator, and selected based on the winning branch indexes. The mapping performed by the second traceback operation is also implemented with a bank of multiplexers, and is described in Table 7.8. The resulting four timing error estimates are fed to a multiplexer indexed by the global maximum winning state `gmax`, which selects and outputs the next timing error signal `T_e`

Table 7.7. Mapping of first traceback operation according to TI and w1–w4.

w1,TI	Te_nn1
00	Te_branch1
01	Te_branch1
10	Te_branch5
11	Te_branch3

w2,TI	Te_nn2
00	Te_branch3
01	Te_branch2
10	Te_branch7
11	Te_branch4

w3,TI	Te_nn3
00	Te_branch2
01	Te_branch5
10	Te_branch6
11	Te_branch7

w4,TI	Te_nn4
00	Te_branch4
01	Te_branch6
10	Te_branch8
11	Te_branch8

Table 7.8. Mapping of second traceback operation according to TI and w1–w4.

w1,TI	Te_n1
00	Te_nn1
01	Te_nn1
10	Te_nn3
11	Te_nn2

w2,TI	Te_n2
00	Te_nn2
01	Te_nn1
10	Te_nn4
11	Te_nn2

w3,TI	Te_n3
00	Te_nn1
01	Te_nn3
10	Te_nn3
11	Te_nn4

w4,TI	Te_n4
00	Te_nn2
01	Te_nn3
10	Te_nn4
11	Te_nn4

7.9 PED

The phase error detector (PED) is responsible for providing the PLF with the next estimate of the phase error signal. The inputs of the PED are:

- `Re_+1_OnTime_MF`, `Im_+1_OnTime_MF`
- `Re_-1_OnTime_MF`, `Im_-1_OnTime_MF`
- `Re_0_OnTime_MF`, `Im_0_OnTime_MF`
- `w1_in`, `w2_in`, `w3_in`, `w4_in`
- `gmax_in`
- `TI_in`, `Valid_in`
- `CTRL`

And the outputs of the PED are:

- `P_e`
- `Valid_out`

A block diagram representation of the PED is shown in Fig. 7.32. The internal structure of the PED has been broken down into two individual units. The first unit is the input selector, which reorders the input so that it is as expected by the error calculator. And the second unit is the error calculator, which computes the phase error estimate `P_e` from the path history of the global maximum winning state `gmax`.

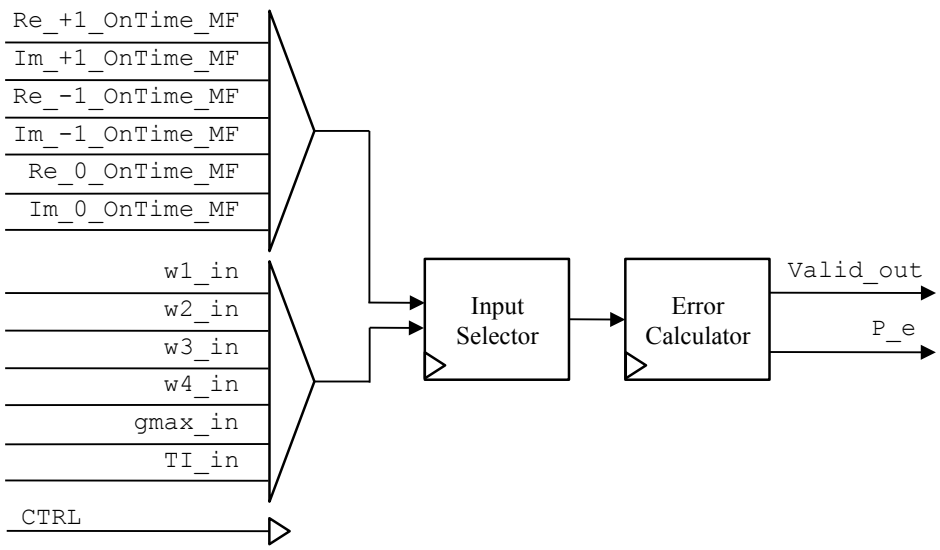

Figure 7.32. Block diagram of the phase error detector.

A hardware representation of the input selector is shown in Fig. 7.33. The input selector receives the complex-valued output from the on-time MFs, and computes the input to be provided to the error calculator unit. The input expected by the error calculator is described by Eq. (6.3). According to this equation, the input to the error calculator is the imaginary part of the multiplication between the on-time MFs, the phase states $e^{-j\hat{\theta}_{k-D}}$, and the term $-j$. Since the phase states only assume $\pm 1, \pm j$ values, the multiplication by $e^{-j\hat{\theta}_{k-D}}$ and the term $-j$ is implemented with a simple sign-alteration and reordering of the real and imaginary parts. When this is taken into consideration, the output produced by Eq. (6.3) can be simply obtained with a bank of six 2-by-1 multiplexers, each one indexed by the trellis indicator. The inputs to the multiplexers are the outputs of the on-time MFs, and their sign-altered counterparts. The mapping necessary to obtain the result described by Eq. (6.3) is shown in Table. 7.9.

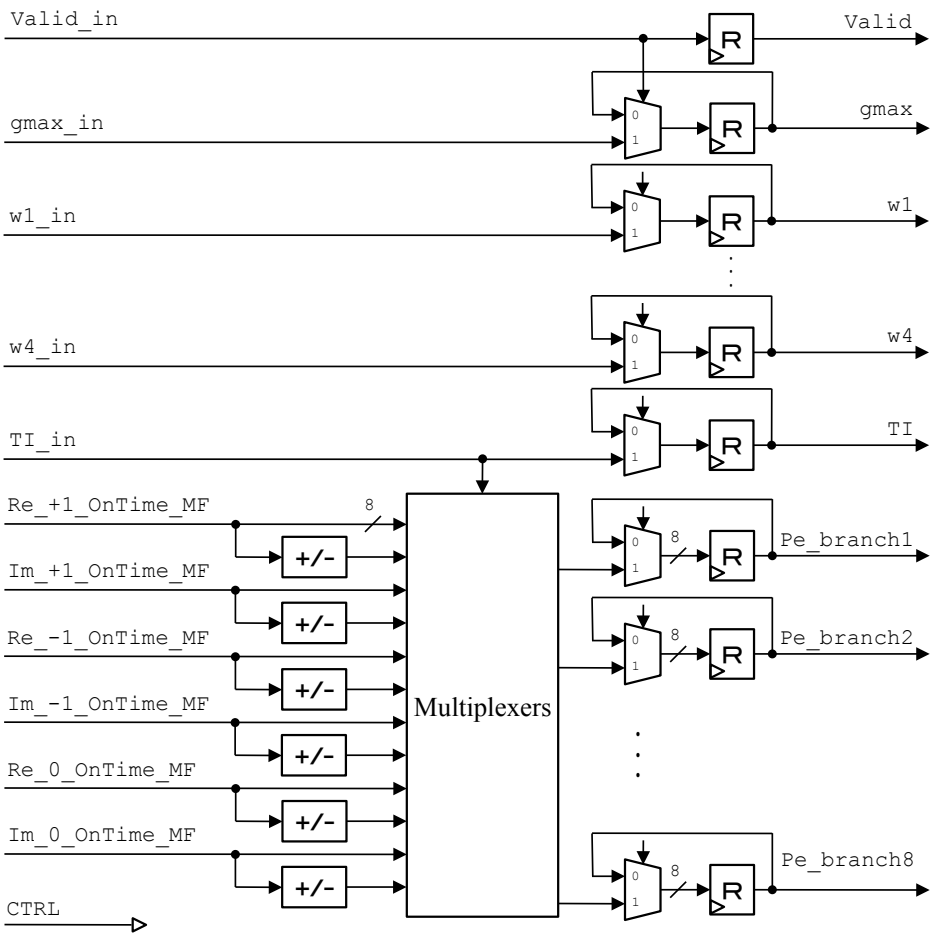

Figure 7.33. Hardware representation of the PED input selector.

A hardware representation of the error calculator is shown in Fig. 7.34. The error calculator receives the eight phase error estimates `Pe_branch1`, ..., `Pe_branch8`, one for each branch in the trellis, and performs the equivalent to two traceback operations to determine the next phase error signal `P_e`. This can also be understood as the PED having a delay of $D = 1$ in computing the phase error signal.

Table 7.9. Mapping of phase-error estimates according to TI.

TI	Pe_branch1
0	Re_0_OnTime_MF
1	Re_0_OnTime_MF

TI	Pe_branch2
0	Re_+1_OnTime_MF
1	Re_−1_OnTime_MF

TI	Pe_branch3
0	−Im_0_OnTime_MF
1	−Im_+1_OnTime_MF

TI	Pe_branch4
0	−Im_−1_OnTime_MF
1	−Im_0_OnTime_MF

TI	Pe_branch5
0	Im_−1_OnTime_MF
1	Im_0_OnTime_MF

TI	Pe_branch6
0	Im_0_OnTime_MF
1	Im_+1_OnTime_MF

TI	Pe_branch7
0	−Re_+1_OnTime_MF
1	−Re_−1_OnTime_MF

TI	Pe_branch8
0	−Re_0_OnTime_MF
1	−Re_0_OnTime_MF

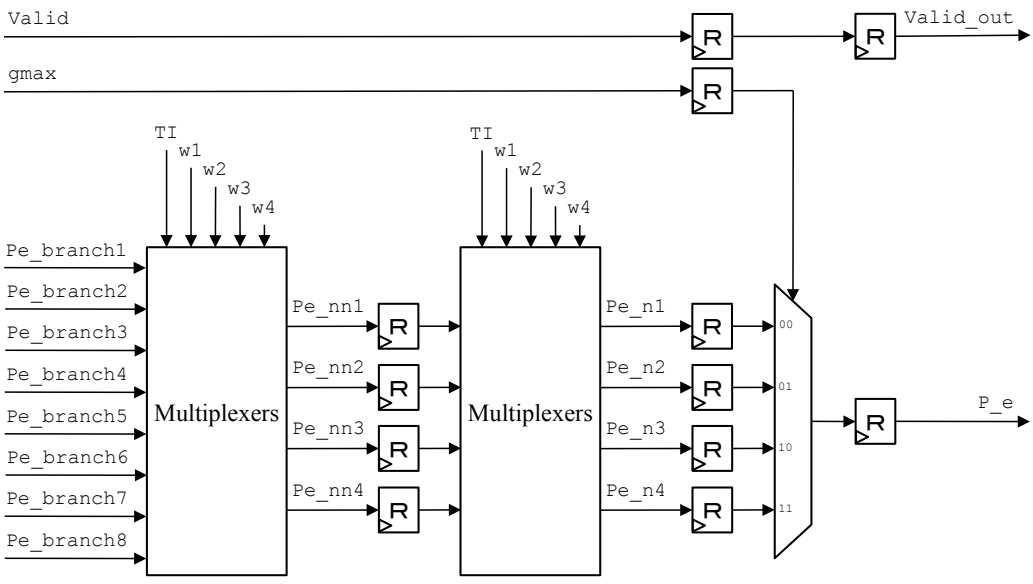

Figure 7.34. Hardware representation of the PED error calculator.

For the first traceback operation, the eight phase error estimates are reordered in pairs merging at the same ending state according to the trellis indicator. Then, using the winner branch indexes `(w1-w4)`, provided by the metric manager, only the phase error estimates corresponding to the winning branches are stored. The mapping performed by the first traceback operation is implemented by a bank of multiplexers. The bank of multiplexers contains four 4-by-1 multiplexers indexed by the winning branch indexes and the trellis indicator as shown in Table 7.10.

In a similar way, for the second traceback operation only the four surviving phase error estimates `Pe_nn1`, ..., `Pe_nn4` are reordered to their corresponding ending states according to the trellis indicator, and selected based on the winning branch indexes. The mapping performed by the second traceback operation is also implemented with a bank of multiplexers, and is described in Table 7.11. The resulting four phase error estimates are fed to a multiplexer indexed by the global maximum winning state `gmax`, which selects and outputs the next phase error signal `P_e`

Table 7.10. Mapping of first traceback operation according to TI and w1–w4.

w1,TI	Pe_nn1
00	Pe_branch1
01	Pe_branch1
10	Pe_branch5
11	Pe_branch3

w2,TI	Pe_nn2
00	Pe_branch3
01	Pe_branch2
10	Pe_branch7
11	Pe_branch4

w3,TI	Pe_nn3
00	Pe_branch2
01	Pe_branch5
10	Pe_branch6
11	Pe_branch7

w4,TI	Pe_nn4
00	Pe_branch4
01	Pe_branch6
10	Pe_branch8
11	Pe_branch8

Table 7.11. Mapping of second traceback operation according to TI and w1–w4.

w1,TI	Pe_n1
00	Pe_nn1
01	Pe_nn1
10	Pe_nn3
11	Pe_nn2

w2,TI	Pe_n2
00	Pe_nn2
01	Pe_nn1
10	Pe_nn4
11	Pe_nn2

w3,TI	Pe_n3
00	Pe_nn1
01	Pe_nn3
10	Pe_nn3
11	Pe_nn4

w4,TI	Pe_n4
00	Pe_nn2
01	Pe_nn3
10	Pe_nn4
11	Pe_nn4

7.10 Soft-Decision Correlator

The soft-decision correlator is responsible for detecting the beginning of a frame in the decoded data stream, as well as, resolving any phase ambiguity resulting from locking with the carrier. The inputs of the soft-decision correlator are:

- `Pu_I`
- `Hu_I`
- `BI_1`, `BI_2`, `BI_3`, ..., `BI_8`
- `CTRL`

And the outputs of the soft-decision correlator are:

- `Pu_O`
- `Hu_O`
- `BI_1_out`, `BI_2_out`, `BI_3_out`, ..., `BI_8_out`
- `Valid_out`

A hardware representation of the soft-decision correlator is shown in Fig. 7.35. The soft-decision correlator receives a new set of inputs about every 16 clock cycles. These inputs correspond to the reliabilities, hard-decisions, and branch-increments generated by the SOVA decoder, and are referred to as the decoded data stream. At this point, the decoded data is only half-way correct, as it still must be processed to find the beginning of a transmitted frame, and to resolve any possible phase ambiguity.

The soft-decision correlator performs both data processing operations by computing two correlation sums between the output reliabilities `Pu_I`, and a known

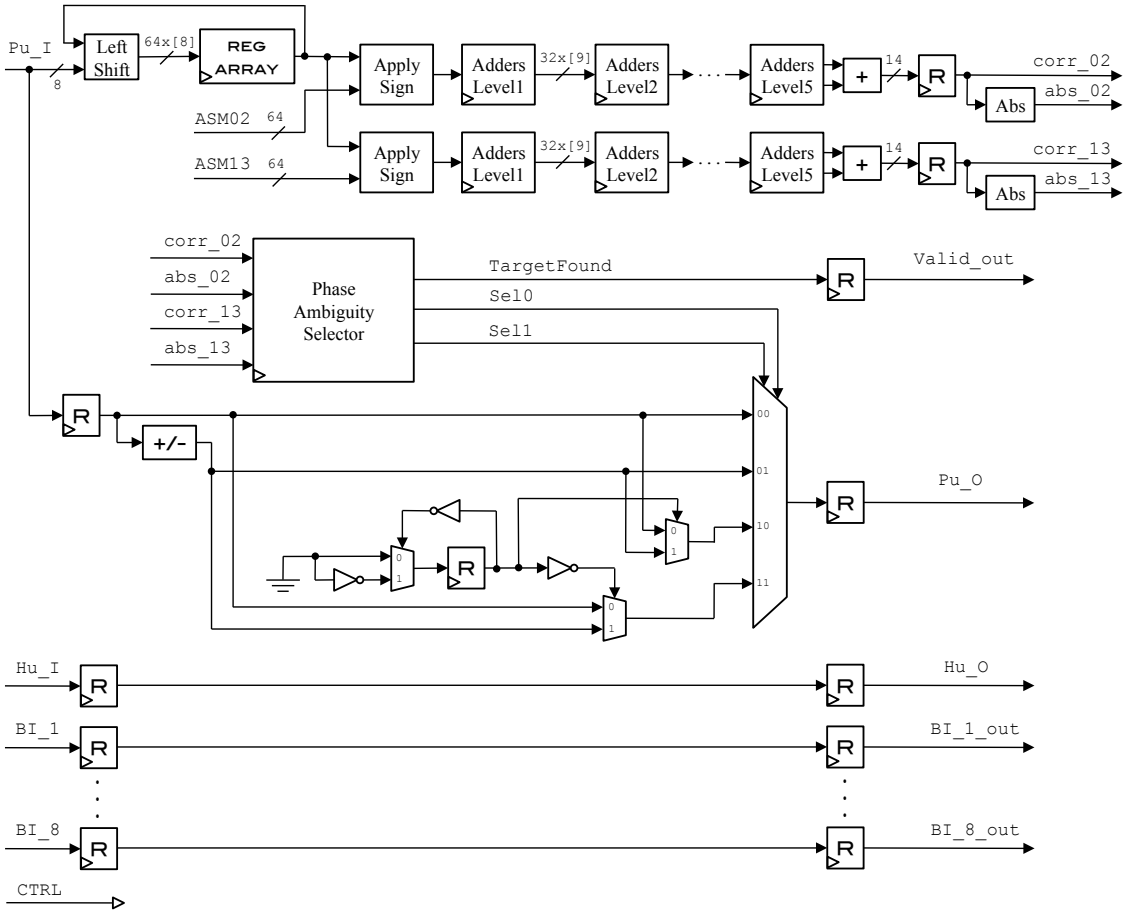

Figure 7.35. Hardware representation of the soft-decision correlator.

sequence of bits attached at the beginning of each frame, called the attached synch marker (ASM). One correlation sum is done using the unmodified ASM, referred to as `ASM02`. And the other one is done using the odd-bit inverted version of `ASM02`, referred to as `ASM13`. In order to perform the one-to-one comparisons required by each correlator, the input reliabilities are left-shifted and stored in a register array with 64 8-bit locations. The length of the array is determined by the 64 bits in the ASM. The correlation is implemented by the apply sign module, which changes the sign of each stored reliability according to the corresponding bit in the ASM. In this way, if a reliability is aligned with a zero, its sign is inverted,

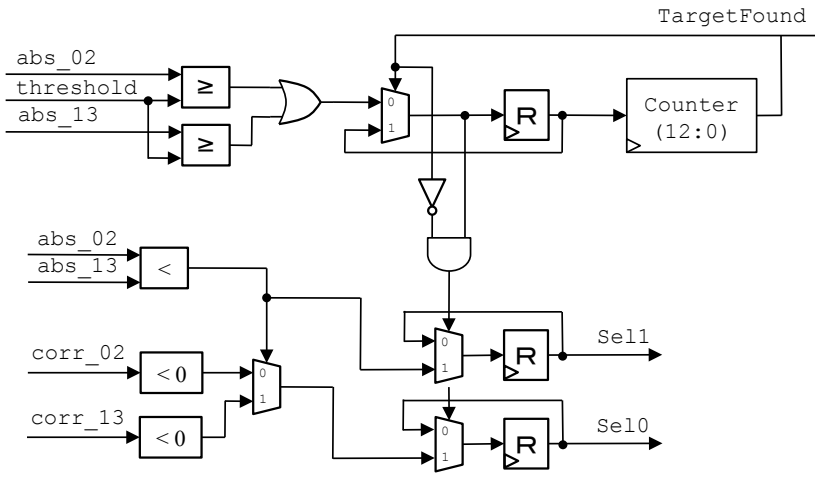

Figure 7.36. Hardware representation of the phase ambiguity selector.

and if its aligned with a one, it remains unchanged. The resulting 64 8-bit comparisons produced by the apply sign module are mutually added in a six-level cascade of adders. The first level contains 32 adders, the second 16, the third 8, the fourth 4, the fifth 2, and the sixth 1. The output of the level six adder corresponds to the correlation sum, from which we also compute its absolute value.

The two correlation sums and their absolute values are used to determine the phase ambiguity in the data, as well as the beginning of a frame. Both operations are performed by the phase ambiguity selector shown in Fig. 7.36. First, the two absolute values are compared against a predefined threshold, set to 915, to determine if a high correlation sum has been obtained. If either comparison returns one, it means that a new frame has been detected, so a counter is started to count for 6240 iterations, which is the length of each frame. The counter's output signal `TargetFound` remains high while the counter's value is between zero and 6240. And then it goes back to zero when the value of 6240 is reached, until a new frame is detected again.

When a new frame is detected, the phase ambiguity selector outputs two signals `Sel0`, and `Sel1`, which index a multiplexer with the four versions of the phase-resolved input reliabilities. The first signal `Sel0` is obtained by comparing the absolute values with themselves. And the second signal `Sel1` is obtained by comparing the correlation sum values with zero, as indicated in Fig. 7.36. The multiplexer then selects one of the four possible phase-resolved reliabilities and sends it to the output. The first input to the multiplexer corresponds to a 0° phase ambiguity. The second input corresponds to a 180° phase ambiguity, so all reliabilities are sign inverted. The third input corresponds to a 90° phase ambiguity, so all odd position reliabilities are sign inverted. And finally, the fourth input corresponds to a 270° phase ambiguity, so all even position reliabilities are sign inverted. The hard-decisions are not modified as they are not used in any other module, but if desired, their phase ambiguity could also be resolved with an additional multiplexer, similar to the one used for the reliabilities.

Chapter 8

Performance Results

This chapter evaluates the performance of the VHDL implementation outlined in Chapter 7 based on two different categories. The first one is *bit error rate* (BER) performance, which compares the decoding results of the proposed demodulator against a software reference model, known to be correct. And the second one is hardware performance, which describes hardware-specific parameters of the chosen design. Throughout this chapter, we refer to the hardware implementation of the demodulator as the "VHDL model", and to the software reference equivalent as the "MATLAB model".

8.1 BER Performance

The VHDL model was first tested in a software simulator, in order to verify the correct functionality of the design. The simulator of choice was ModelSim, because it runs considerably faster than other available software simulators. The input to the VHDL model, which is the noisy received signal, was generated in MATLAB and quantized to have a bit-width of $B = 8$, with four bits being fractional. This

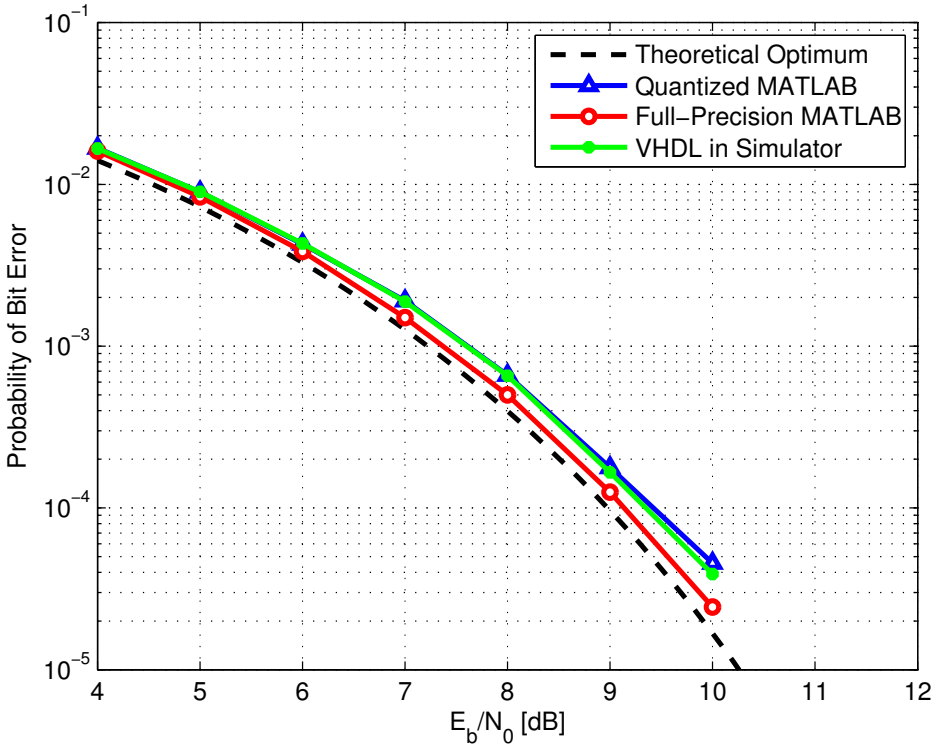

Figure 8.1. BER performance of VHDL model in ModelSim.

bit-width was selected in order to accurately represent the majority of the values in the range of the received signal, while at the same time keeping the complexity of the design at a manageable level. Bit-widths for the remaining registers in the design were estimated from a quantized version of the MATLAB reference model. Also, by means of the quantized MATLAB model, we were able to approximate the performance of the hardware implementation.

The software test of the VHDL model was run over different values of E_b/N_0 in the interval from 4 dB to 10 dB. Each simulation was run over a minimum of 1,000,000 transmitted bits, with a requirement of at least 100 bit errors after decoding. These conditions were sufficient to generate an accurate BER plot. To determine if the VHDL model was operating correctly, we compared its BER

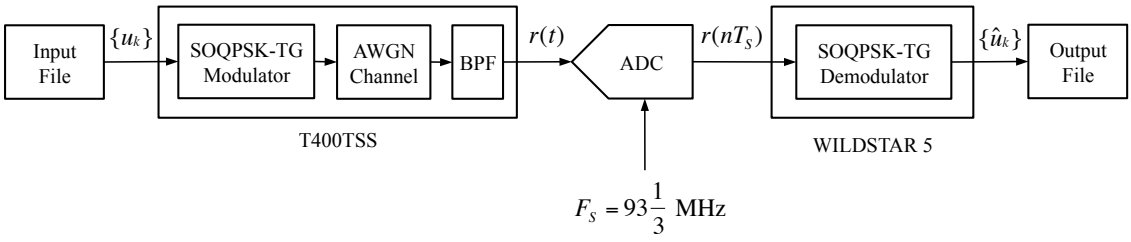

Figure 8.2. Block diagram representation of the hardware test setting.

plot with that of the full-precision MATLAB model, known to be correct. In addition, we also compared it with the BER plot of the quantized MATLAB model. Fig. 8.1 shows the BER performance of the VHDL model in ModelSim compared against the full-precision and quantized MATLAB models. Fig. 8.1 also shows the theoretical optimum performance achievable by the demodulator. We observe that the plot of the VHDL model almost overlaps the plot of the quantized MATLAB model. This means that their BER performance is almost identical, which is something we expected and that verifies the correctness of the design. We also notice that the plot of the VHDL model is very close to that of the full-precision MATLAB model, but with a slight difference. This difference slowly increases with the positive values of E_b/N_0, and can be attributed to the effects of rounding due to the fixed-precision characteristic of the hardware implementation. On average, the loss in performance from the simulation in software is of approximately 0.1$\sim$0.2 dB.

After successfully testing the correct functionality of the VHDL model in software, we proceeded to test it in the actual hardware. For this purpose, we arranged a test setting like the one illustrated in Fig. 8.2. The T400TSS block is a telemetry signal simulator manufactured by the company RT-Logic. This module was used to repeatedly modulate the bit sequence of 6240 bits provided in the input file at

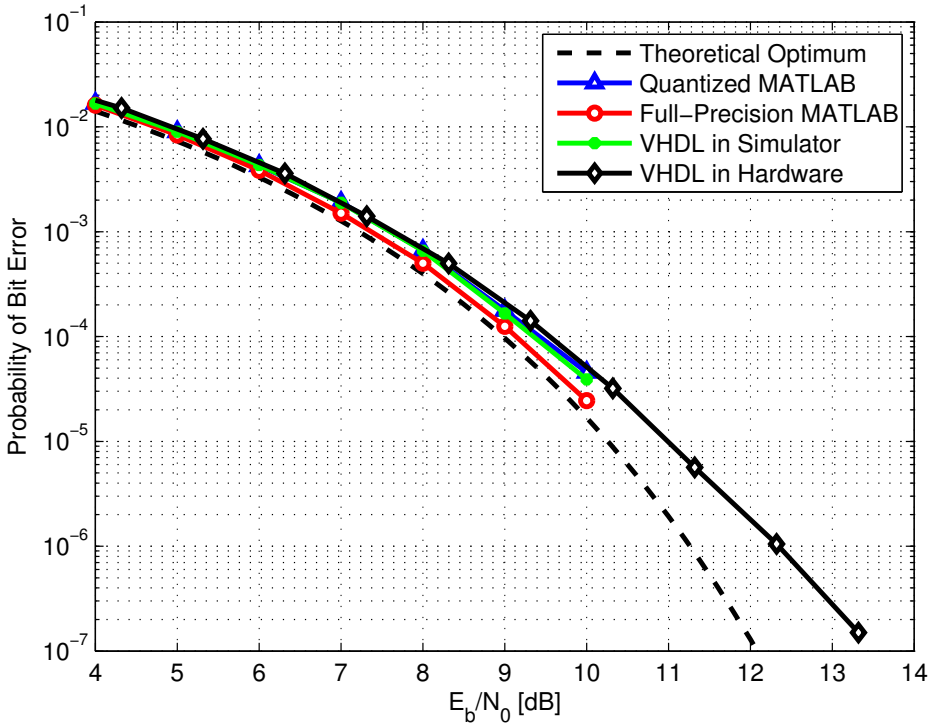

Figure 8.3. BER performance of VHDL model in hardware.

a rate of 5.866 Mbits/s. It was also used to convert the modulated signal to IF, and to add the effects of an AWGN channel. And finally, it was used to apply an anti-aliasing band-pass filter to the resulting signal. The continuous-time received signal was sampled by an ADC at the specific rate of $93\frac{1}{3}$ MHz to produce the input to the demodulator. The WILDSTAR 5 block is a processing board manufactured by the company Annapolis Micro Systems, and contains three Virtex-5 XC5VLX110T FPGAs. After synthesis, mapping and routing, the VHDL design was downloaded into one of the three FPGAs. An output file in the host computer captured the decoded bits to later determine the BER performance of the hardware.

The hardware test of the VHDL model was run over different values of E_b/N_0

in the interval from 4.3 dB to 13.3 dB. The start value of 4.3 dB corresponds to the minimum signal-to-noise power level allowed by the signal simulator. Since the hardware simulation runs considerably faster than the software one, each point was evaluated over 40,000,000 transmitted bits, instead of over just 1,000,000. This allows us to evaluate larger values of E_b/N_0, and to generate a more reliable BER plot. To determine the performance of the VHDL model in hardware, the resulting BER plot was compared with that of the full-precision MATLAB models. Fig. 8.3 shows the BER performance of the VHDL model tested in hardware, as well as all the other plots previously described in the software simulation. In the interval from 4 dB to 10 dB, we can observe that the VHDL performance in hardware has very similar behavior to that of the full-precision MATLAB model, but with a small difference of approximately 0.3∼0.4 dB. This performance difference can be attributed to the effects of hardware error like a small offset in the clock frequency. As it was expected, the performance loss is slightly larger in the hardware simulation than in the software simulation since the former is subject to additional factors. The average BER performance loss for both simulations compared against the full-precision MATLAB model is summarized in Table 8.1. Data points for the three software reference plots at 11 dB and greater are not shown in Fig. 8.3 as they take a very large number of bits, and consequently time, to obtain in software.

Table 8.1. Average BER performance loss.

Simulation Type	Performance Loss
Software	$0.1 \sim 0.2$ dB
Hardware	$0.3 \sim 0.4$ dB

8.2 Hardware Performance

The VHDL model was also evaluated in terms of its hardware-specific performance. The target FPGA for the VHDL model is the Virtex-5 XC5VLX110T. According to [17], this device has $17,280$ available Virtex-5 slices, with each slice containing four lookup tables (LUTs) and four flip-flops (FFs). The overall footprint of the design is of medium size, consuming about 16% (2,790 out of 17,280) of the available slices.

In addition to resource utilization, we are also interested in the maximum clock frequency achieved by the design. By means of the ISE design suite, a user constraint was defined for the clock signal to have a period of 10ns with a 50% duty cycle. Based on this condition, ISE built the design in several attempts trying to obtain the maximum clock frequency. We found that on average, the maximum clock frequency achieved was of 114.6 MHz. This is a successful result as the minimum requirement was to be greater than the ADC sample frequency $F_s = 93\frac{1}{3}$ MHz. The hardware performance results of the VHDL model are summarized in Table 8.2.

Table 8.2. Hardware performance results of the VHDL model.

Slices Occupied (%)	Maximum Clock Frequency (MHz)
16	114.6

Chapter 9

Conclusion

9.1 Interpretation of Results

The BER plots presented in Chapter 8 indicate that the performance of the proposed hardware implementation of the SOQPSK-TG demodulator is very similar to that of the MATLAB reference model, with only a small loss of $\sim$0.4 dB. Considering that this loss in performance is due to the effects of rounding and apparatus error, it can be concluded that the VHDL implementation is correct. This means that the proposed SOQPSK-TG demodulator can reliably and successfully recover the symbol timing, adjust the carrier phase, and estimate the transmitted sequence of bits from the noisy received signal.

These results also demonstrate that successfully implementing in hardware theoretical principles such as the Viterbi algorithm, maximum-likelihood timing recovery and phase synchronization, early-late timing error detection, etc, is possible. Although ideal case performances might not yet be achievable in hardware due to the fixed-precision nature of the designs, and potential equipment error, a very close approximation in performance is definitely within reach.

In terms of hardware-specific performance, the proposed hardware implementation is desirable because it consumes a relatively low number of FPGA resources, and it exceeds the minimum clock frequency requirement.

9.2 Future Work

One suggestion for future work in the hardware implementation is to evaluate the effect that higher quantization resolution may have in the demodulator's performance. This could be studied by increasing the number of bits used to represent internal signals, and observing whether the changes cause an improvement in performance or not. But increasing the resolution comes with a cost in design complexity. Therefore, finding the optimum balance between signal quantization and performance is a tradeoff worth exploring.

Another suggestion for future work is to implement second-order loop filters for the timing and phase synchronizers. A second-order PLL would allow the system to handle larger frequency deviations, and therefore, bring the performance closer to ideal. However, implementing this more sensitive loop filter imposes the challenge of dealing with very high-precision parameters (>20 bits of quantization) which substantially increases the complexity.

Finally, as part of the efforts of the HFEC project, this demodulator must be connected with the convolutional code decoder and the LDPC decoder, as described in Chapter 3, to obtain more robust decoding schemes.

References

[1] J. B. Anderson, T. Aulin, and C.-E. Sundberg, *Digital Phase Modulation*. New York: Plenum Press, 1986.

[2] Range Commanders Council Telemetry Group, Range Commanders Council, White Sands Missile Range, New Mexico, *IRIG Standard 106-04: Telemetry Standards*, 2004. (Available on-line at http://www.ntia.doc.gov/osmhome/106.pdf).

[3] T. Aulin, C.-E. Sundberg, and A. Svensson, "Viterbi detectors with reduced complexity for partial response continuous phase modulation," in *Proc. National Telecommun. Conf., NTC'81*, (New Orleans, LA), pp. A7.6.1–A7.6.7, Nov./Dec. 1981.

[4] A. Svensson, C.-E. Sundberg, and T. Aulin, "A class of reduced-complexity Viterbi detectors for partial response continuous phase modulation," *IEEE Trans. Commun.*, vol. 32, pp. 1079–1087, Oct. 1984.

[5] E. Perrins and M. Rice, "Reduced-complexity approach to iterative detection of SOQPSK," *IEEE Trans. Commun.*, vol. 55, pp. 1354–1362, Jul. 2007.

[6] M. Simon, *Bandwidth-Efficient Digital Modulation With Application to Deep-Space Communication*. New York: Wiley, 2003.

[7] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding," *IEEE Transactions on Information Theory*, vol. 44, pp. 909 – 926, May 1998.

[8] M. Simon and L. Li, (2003, Aug.), "A cross-correlated trellis-coded quadrature modulation representation of MIL-STD shaped offset quadrature phase-shift keying," *Interplan. Network Prog. Rep.*, vol. [Online]. Available: http://ipnpr.jpl.nasa.gov/tmo/progress_report/42-154/154J.pdf.

[9] E. Perrins and M. Rice, "Simple detectors for shaped-offset QPSK using the PAM decomposition," in *Proc. IEEE Global Telecommun. Conf.*, (St. Louis, Missouri), pp. 408–412, Nov./Dec. 2005.

[10] J. Hagenauer and P. Hoeher, "Concatenated Vitebi Decoding," in *Proceedings of the International Workshop on Information Theory*, 1989.

[11] P. Chandran and E. Perrins, "Decision-directed symbol timing recovery for SOQPSK," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 45, pp. 781–789, Apr. 2009.

[12] M. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the equivalence between SOVA and Max-Log-MAP decodings," *IEEE Trans. Commun.*, vol. 2, pp. 137–139, May. 1998.

[13] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley, 2005.

[14] M. Rice, *Digital Communications: A Discrete-Time Approach*. New York: Prentice Hall, 2009.

[15] M. Morelli, U. Mengali, and G. M. Vitetta, "Joint phase and timing recovery with CPM signals," *IEEE Trans. Commun.*, vol. 45, pp. 867–876, Jul. 1997.

[16] D. Lee, J. Villasenor, W. Luk, and P. Leong, "A hardware Gaussian noise generator using the Box-Muller method and its error analysis.," *IEEE Trans. Computers.*, vol. 55, pp. 659–671, June. 2006.

[17] Xilinx, "Virtex-5 Family Overview: Product Specification." February 2009. DS100 (v5.0).