Module Interface Specification for RwaveDetection

Junwei Lin

April 19, 2025

1 Revision History

Date	Version	Notes
Mar 17, 2025	1.0	Creation
April 18, 2025	1.1	Modify the document according to the received feedback

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/Lychee-acaca/CAS741/blob/main/docs/SRS/SRS.pdf.

Contents

3 Introduction

The following document details the Module Interface Specifications for RwaveDetection.

This project focuses on reimplementing the Pan-Tompkins algorithm for R-wave detection in ECG signal processing.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/Lychee-acaca/CAS741.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by RwaveDetection.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of RwaveDetection uses some derived data types: sequences and strings. Sequences are lists filled with elements of the same data type, they are represented using a type with a superscript, for example, $\mathbb{R}^n$ represents a real number sequence of length n. Strings are sequences of characters. In addition, RwaveDetection uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	General Digital Filter Module Specified IIR Filter Module Math Module Pan Tompkins Algorithm Module Annotated Data RMSE Module
Software Decision Module	Rwave Detect Module Input Output Processing Module Error Handler Module Logger Module

Table 1: Module Hierarchy

6 MIS of Hardware-Hiding Module

This section outlines the formal specification of the Hardware-Hiding Module, which provides an abstraction layer for interacting with hardware devices.

6.1 Module

Hardware-Hiding

6.2 Uses

None

6.3 Syntax

This section describes the syntax used in the Hardware-Hiding Module, including the constants, programs, and routines that can be accessed.

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
open	string, $\mathbb{N}$, $\mathbb{N}$	$\mathbb{Z}$	-
write	$\mathbb{Z}$, $\mathbb{N}^n$, $\mathbb{N}$	$\mathbb{N}$	-
read	$\mathbb{Z}$, $\mathbb{N}^n$, $\mathbb{N}$	$\mathbb{N}$	-

6.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

6.4.1 State Variables

None

6.4.2 Environment Variables

- **pathname**: A string environment variable representing the file's full path or filename to be accessed by the Hardware-Hiding Module.

6.4.3 Assumptions

- The input parameters to open must be valid and correspond to existing hardware components.
- The write operation assumes that the hardware device is available and writable.
- The read operation assumes that the hardware device is readable.

6.4.4 Access Routine Semantics

open(pathname, flags, mode):
    See https://en.wikipedia.org/wiki/Open_(system_call).
write(fd, buf, count):
    See https://en.wikipedia.org/wiki/Write_(system_call).
read(fd, buf, count):
    See https://en.wikipedia.org/wiki/Read_(system_call).

6.4.5 Local Functions

None

7 MIS of General Digital Filter Module

This section outlines the formal specification of the General Digital Filter Module, which provides linear filter functions for filtering in the form of difference equations.

7.1 Module

General Digital Filter

7.2 Uses

- Error Handler Module (Section 14)
- Logger Module (Section 15)

7.3 Syntax

This section describes the syntax used in the General Digital Filter Module, including the constants, programs, and routines that can be accessed.

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
lfilter	$\mathbb{R}^n$, $\mathbb{R}^m$, $\mathbb{R}^t$	$\mathbb{R}^t$	-

7.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

- The filter numerator coefficients $b = [b_0, b_1, \ldots, b_{n-1}]$ and denominator coefficients $a = [a_0, a_1, \ldots, a_{m-1}]$ are valid real-valued sequences.
- The input sequence $x = [x_0, x_1, \ldots, x_{t-1}]$ is a real-valued discrete-time series.
- The first coefficient of a satisfies $a_0 \neq 0$ to ensure the difference equation is well-defined.

7.4.4 Access Routine Semantics

lfilter(b, a, x):

- output: The filtered sequence y.
- exception: None

7.4.5 Local Functions

None

8 MIS of Specified IIR Filter Module

This section outlines the formal specification of the Specified IIR Filter Module, which provides parameter derivation functions for Butterworth and Chebyshev filters.

8.1 Module

Specified IIR Filter

8.2 Uses

- General Digital Filter Module (Section 7)
- Error Handler Module (Section 14)
- Logger Module (Section 15)

8.3 Syntax

This section describes the syntax used in the Specified IIR Filter Module, including the constants, programs, and routines that can be accessed.

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
butter	$\mathbb{N}$, $\mathbb{R}$, string	$\mathbb{R}^n$, $\mathbb{R}^m$	ResultExceededExpectation
cheby1	$\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}$, string	$\mathbb{R}^n$, $\mathbb{R}^m$	ResultExceededExpectation
cheby2	$\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}$, string	$\mathbb{R}^n$, $\mathbb{R}^m$	ResultExceededExpectation

8.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

8.4.1 State Variables

None

8.4.2 Environment Variables

- **system_epsilon**: The floating-point precision threshold of the host system, used to determine acceptable numerical error bounds in filter coefficient calculation.
- **max_order_limit**: The maximum allowable filter order supported by the system to prevent excessive computational cost or numerical instability.

8.4.3 Assumptions

- The system performing the calculations can handle floating-point precision properly and maintains `system_epsilon` within a reasonable range for numerical stability.
- The requested filter order does not exceed `max_order_limit`.

8.4.4 Access Routine Semantics

butter(N, Wn, btype):

- transition:
  - Verify that $N \leq$ `max_order_limit`; if not, log the event and raise `ResultExceededExpectationE`
  - Compute the Butterworth filter poles in the s-domain.
  - Apply a bilinear transformation to map the s-domain poles to the z-domain.
  - Convert the poles to numerator (b) and denominator (a) sequences.
  - Ensure that numerical errors in b and a are within `system_epsilon`; if not, log and raise `ResultExceededExpectationError`.
- output: The parameter sequences b and a of the filter.

- exception:
  - If $N >$ `max_order_limit`.
  - If the resulting coefficients contain numerical errors exceeding `system_epsilon`.

cheby1(N, rp, Wn, btype):

- transition:
  - Validate $N \leq$ `max_order_limit`.
  - Calculate Chebyshev Type I poles with ripple rp.
  - Transform the poles from the s-domain to the z-domain via bilinear transformation.
  - Derive b and a coefficient sequences.
  - Check if computed coefficients meet precision threshold `system_epsilon`; otherwise, raise exception.
- output: The parameter sequences b and a of the filter.
- exception:
  - If $N >$ `max_order_limit`.
  - If numerical errors in results exceed `system_epsilon`.

cheby2(N, rs, Wn, btype):

- transition:
  - Validate $N \leq$ `max_order_limit`.
  - Calculate Chebyshev Type II poles ensuring stopband attenuation rs.
  - Apply bilinear transformation for s-to-z domain mapping.
  - Form coefficient sequences b and a.
  - Check numerical stability and precision against `system_epsilon`.
- output: The parameter sequences b and a of the filter.
- exception:
  - If $N >$ `max_order_limit`.
  - If precision violations occur beyond `system_epsilon`.

8.4.5 Local Functions

None

9 MIS of Math Module

This section outlines the formal specification of the Math Module, which provides three mathematical functions: squaring, thresholding, and RMSE calculation.

9.1 Module

Math

9.2 Uses

- Error Handler Module (Section 14)
- Logger Module (Section 15)

9.3 Syntax

This section describes the syntax used in the Math Module, including the constants, programs, and routines that can be accessed.

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
calSquare	$\mathbb{R}^n$	$\mathbb{R}^n$	-
calThreshold	$\mathbb{R}^n$, $\mathbb{R}^n$	$\mathbb{R}^n$	InvalidInput
calRMSE	$\mathbb{R}^n$, $\mathbb{R}^n$	$\mathbb{R}$	InvalidInput

9.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

9.4.1 State Variables

None

9.4.2 Environment Variables

- **system_epsilon**: The floating-point precision threshold of the host system, used to determine acceptable numerical error in RMSE calculation.

9.4.3 Assumptions

- All input sequences are non-empty.
- Floating-point operations on the host system conform to the system-defined `system_epsilon`.

9.4.4 Access Routine Semantics

calSquare(x):

- transition:
  - For each element x_i in sequence x, compute x_i^2.
  - Store the result in the corresponding position in the output sequence.
- output: A sequence containing the square of each element from x.

calThreshold(x, thres):

- transition:
  - Verify that the lengths of x and `thres` are equal.
  - For each pair of elements (x_i, thres_i):
    - If $x_i \geq \text{thres}_i$, retain x_i.
    - Otherwise, set the result to 0.
- output: A sequence of thresholded values.
- exception:
  - If length of x and `thres` differ, log the event and raise `InvalidInput` exception.

calRMSE(x, y):

- transition:
  - Verify that the lengths of x and y are equal.
  - Compute the mean of the squared differences:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$$

  - Compute the square root of MSE to obtain RMSE.
  - Check if the final RMSE value contains invalid floating-point values (e.g. NaN or infinity) or violates `system_epsilon` tolerance; if so, log and raise `InvalidInput`.

- output: The calculated RMSE value.
- exception:
  - If length of x and y differ, or result precision is invalid, raise `InvalidInput` exception.

9.4.5 Local Functions

None

10 MIS of Pan Tompkins Algorithm Module

This section outlines the formal specification of the Pan Tompkins Algorithm Module, which provides the implementation of the Pan-Tompkins algorithm.

10.1 Module

Pan Tompkins Algorithm

10.2 Uses

- Specified IIR Filter Module (Section 8)
- General Digital Filter Module (Section 7)
- Math Module (Section 9)
- Error Handler Module (Section 14)
- Logger Module (Section 15)

10.3 Syntax

This section describes the syntax used in the Pan Tompkins Algorithm Module, including the constants, programs, and routines that can be accessed.

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
findPeak	$\mathbb{R}^n$, $\mathbb{N}$	$\mathbb{Z}^m$	InvalidInput

10.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

10.4.1 State Variables

None

10.4.2 Environment Variables

- **lowpass_cutoff**: Low-pass filter cutoff frequency (Hz).
- **highpass_cutoff**: High-pass filter cutoff frequency (Hz).
- **integration_window**: Window size for moving average integration (in samples).
- **peak_threshold_factor**: Multiplier for adaptive peak detection threshold.

10.4.3 Assumptions

- The input sequence `rawSignal` is a non-empty array of real numbers.
- The input sequence `rawSignal` is suitable for digital filtering.
- Sampling frequency `fs` is a positive integer.

10.4.4 Access Routine Semantics

findPeak(rawSignal, fs):

- transition:
  - Verify that the length of `rawSignal` is greater than the required minimum based on filter orders and integration window size.
  - Use `Specified IIR Filter Module` to design Butterworth bandpass filters with `lowpass_cutoff` and `highpass_cutoff`, normalized by `fs`.
  - Apply bandpass filtering to `rawSignal` using `General Digital Filter Module`.
  - Differentiate the filtered signal.
  - Square each sample in the differentiated signal using `Math Module: calSquare`.
  - Apply moving average integration using a window of size `integration_window`.
  - Detect peaks by:
    - Computing an adaptive threshold as a fraction (`peak_threshold_factor`) of the maximum value in the integrated signal.
    - Identifying local maxima exceeding this threshold.
  - Log the number of detected peaks and their locations.
- output: An integer sequence representing the indices of R-wave peaks.
- exception:
  - If `rawSignal` is shorter than the required minimum length, log the event and raise `InvalidInput`.

10.4.5 Local Functions

None

11 MIS of Annotated Data RMSE Module

This section outlines the formal specification of the Annotated Data RMSE Module, which provides functionality to compare algorithm results with annotated data to determine the magnitude of algorithmic errors.

11.1 Module

Annotated Data RMSE

11.2 Uses

- Math Module (Section 9)
- Error Handler Module (Section 14)
- Logger Module (Section 15)

11.3 Syntax

This section describes the syntax used in the Annotated Data RMSE Module, including the constants, programs, and routines that can be accessed.

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
signalRMSE	$\mathbb{Z}^n$, $\mathbb{Z}^m$	$\mathbb{R}$	NoMatchingPoint

11.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

11.4.1 State Variables

None

11.4.2 Environment Variables

- **matching_window**: Maximum allowable difference (in samples) between a raw index and an annotation index for them to be considered a match.
- **min_matching_count**: Minimum number of matched point pairs required to compute RMSE.

11.4.3 Assumptions

- The implementation of the Pan-Tompkins algorithm is correct.
- Both `rawIndices` and `annIndices` are sorted in ascending order.

11.4.4 Access Routine Semantics

signalRMSE(rawIndices, annIndices):

- transition:
  - Match each annotated index in `annIndices` with the nearest raw index in `rawIndices`, provided the absolute difference is less than or equal to `matching_window`.
  - Form pairs of matched points.
  - If the number of matched pairs is less than `min_matching_count`, log this event and raise `NoMatchingPoint` exception.
  - Compute the differences between matched pairs.
  - Use `Math Module: calRMSE` to compute the root mean square error (RMSE) from these differences.
  - Log the final RMSE value.
- output: A real number representing the RMSE between matched points.
- exception:
  - If no matching pairs are found or the number of matches is below `min_matching_count`, raise `NoMatchingPoint`.

11.4.5 Local Functions

None

12 MIS of Rwave Detect Module

This section outlines the formal specification of the Rwave Detect Module, which provides initialization and invocation functionalities for other modules, serving as the entry point for the entire program.

12.1 Module

Rwave Detect

12.2 Uses

- Input Output Processing Module (Section 13)
- Pan Tompkins Algorithm Module (Section 10)
- Annotated Data RMSE Module (Section 11)
- Error Handler Module (Section 14)
- Logger Module (Section 15)

12.3 Syntax

This section describes the syntax used in the Rwave Detect Module, including the constants, programs, and routines that can be accessed.

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
mainEntry	-	-	-

12.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

12.4.1 State Variables

- **runningState**: Records the current stage of the program's execution, such as **Initializing**, **Running**, or **ErrorTerminated**.

12.4.2 Environment Variables

None

12.4.3 Assumptions

- The program will first initialize and enter this module during execution.
- All dependent modules are correctly implemented and can be initialized without failure.

12.4.4 Access Routine Semantics

mainEntry:

- transition:
  1. Set `runningState` to `Initializing`.
  2. Log the program start event.
  3. Initialize all dependent modules:
     - Input Output Processing Module
     - Pan Tompkins Algorithm Module
     - Annotated Data RMSE Module
     - Error Handler Module
     - Logger Module
  4. Load raw ECG data using the Input Output Processing Module.
  5. Apply the Pan-Tompkins Algorithm Module to detect R-wave peak indices.
  6. Load annotated (reference) R-wave peak indices.
  7. Call the Annotated Data RMSE Module to calculate the RMSE between detected and annotated indices.
  8. Output the RMSE result using the Input Output Processing Module.
  9. Set `runningState` to `Running`.
  10. Log successful completion.
- exception:
  - If any unexpected error or exception occurs during module initialization or execution, log the error via the Logger Module and set `runningState` to `ErrorTerminated`.
  - Use the Error Handler Module to gracefully handle or propagate exceptions as necessary.

12.4.5 Local Functions

None

13 MIS of Input Output Processing Module

This section outlines the formal specification of the Input Output Processing Module, which provides basic data input and output functionality by calling the Hardware-Hiding Module.

13.1 Module

Input Output Processing

13.2 Uses

- Hardware-Hiding Module (Section 6)
- Error Handler Module (Section 14)
- Logger Module (Section 15)

13.3 Syntax

This section describes the syntax used in the Input Output Processing Module, including the constants, programs, and routines that can be accessed.

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
readFromFile	string	$\mathbb{R}^n$	FileError
writeToFile	string, $\mathbb{Z}^m$	-	FileError

13.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

13.4.1 State Variables

None

13.4.2 Environment Variables

None

13.4.3 Assumptions

- The operating system supports correct file read and write operations.
- The file paths provided are valid and accessible to the program.

13.4.4 Access Routine Semantics

readFromFile(filename):

- transition:
  - Calls the Hardware-Hiding Module to open and read the file located at the specified `filename`.
  - Parses the file content into an array of real numbers representing the ECG signal.
- output: A sequence of real numbers $\mathbb{R}^n$ representing the ECG signal after format conversion.
- exception:
  - If the file cannot be opened.
  - If the file format is invalid or unreadable.

  Then throw a `FileError` exception.

writeToFile(filename, index):

- transition:
  - Calls the Hardware-Hiding Module to open or create the file specified by `filename`.
  - Writes the integer sequence `index` (R-wave peak indices) into the file, one value per line or in a defined format.
- output: None
- exception:
  - If the file cannot be opened for writing.
  - If a write operation fails.

  Then throw a `FileError` exception.

13.4.5 Local Functions

None

14 MIS of Error Handler Module

This section outlines the formal specification of the Error Handler Module, which provides handling functionality for exceptions that occur during the execution of the program.

14.1 Module

Error Handler

14.2 Uses

- Hardware-Hiding Module (Section 6)

14.3 Syntax

This section describes the syntax used in the Error Handler Module, including the constants, programs, and routines that can be accessed.

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
throwException	string	-	-

14.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

14.4.1 State Variables

None

14.4.2 Environment Variables

None

14.4.3 Assumptions

- The exception information can be correctly logged and saved via the Hardware-Hiding Module.
- The program can safely terminate or recover following an exception.

14.4.4 Access Routine Semantics

throwException(ex):

- transition:
  - Immediately log the exception message `ex` using the Hardware-Hiding Module.
  - Optionally notify the user via the standard output or log file.
  - Terminate the program's execution or transfer control to a safe shutdown routine.
- output: None
- exception: None

14.4.5 Local Functions

None

15 MIS of Logger Module

This section outlines the formal specification of the Logger Module, which provides functionality for logging output.

15.1 Module

Logger

15.2 Uses

- Hardware-Hiding Module (Section 6)

15.3 Syntax

This section describes the syntax used in the Logger Module, including the constants, programs, and routines that can be accessed.

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
log	string, $\mathbb{N}$	-	-
setLogLevel	$\mathbb{N}$	-	-
setLogPath	string	-	FileError

15.4 Semantics

This section provides the detailed functionality and behavior of the module's functions.

15.4.1 State Variables

- runningTime: The runtime of the program since initialization.
- logLevel: The current log level threshold. Messages below this level will not be logged.
- logPath: The path to the log file.

15.4.2 Environment Variables

- The default log level in the configuration file.

15.4.3 Assumptions

- Logs can be correctly output and stored via the Hardware-Hiding Module.
- The file system is accessible for writing log files.

15.4.4 Access Routine Semantics

log(msg, level):

- transition:
  - If `level` $\geq$ `logLevel`, format the message with a timestamp and program `runningTime`, and append it to the log file at `logPath`.
  - If `level` $<$ `logLevel`, no action is performed.
- output: None
- exception: None

setLogLevel(level):

- transition: Set `logLevel` to the specified `level`. Suppress all log messages with a level lower than this value.
- output: None
- exception: None

setLogPath(filename):

- transition: Set `logPath` to `filename`. All subsequent log messages will be written to this file.
- exception: If the file cannot be opened for writing, throw a `FileError`.

15.4.5 Log Levels Definition

The following numeric levels are used to categorize log messages:

- **0**: ERROR — Critical issues causing program termination.
- **1**: WARNING — Important events that do not halt the program but require attention.
- **2**: INFO — General information about program execution.
- **3**: DEBUG — Detailed debugging messages for development use.

15.4.6 Local Functions

None

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.