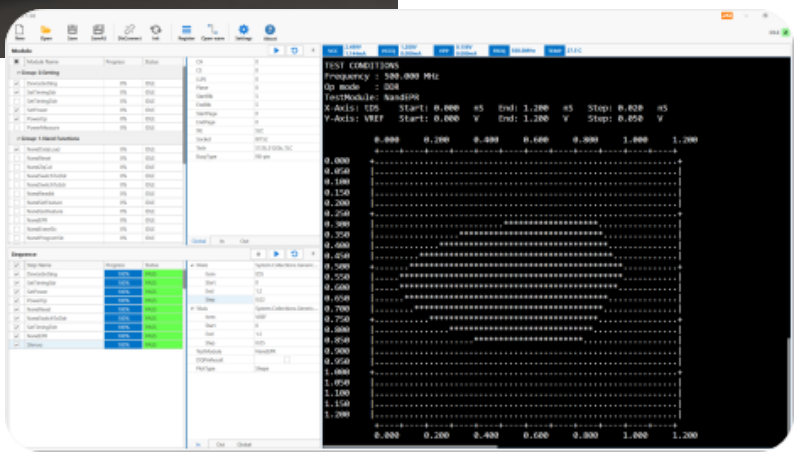
JKI NAND Engineering Tester

Programming Manual

Applicable Product: JTS350F, JTS351F



Version 2.1

JKI Inc.

Version History

Version	Date	Remarks
1.0	2021-10-15	Draft release.
2.0	2022-07-26	Renewal manual.
2.1	2022-11-22	Added API functions

Contents

1. Overview

1.1. Description

JKI's high speed NAND tester delivers high performance with affordable price and small size for testing next generation high speed NAND flash memories. This NAND tester is a portable size engineering test solution developed based on the latest MPSoC + FPGA technology. It supports 2.4Gbps interface using high speed DDR I/O and provides multi-channel programmable precision device power supplies.

<JTS350F>

<JTS351F>

1.2. Features

Fast Control & Easy Development

- Fast MPSoC based test controller (ARM + FPGA)
- Easy test program development environment on Linux
- Easy sequence & parameter control environment on Windows

Software-defined Pattern Generator

- Software-defined command & data pattern generation
- Continuously streaming write and read data vector

Hardware-based Timing Generator

- Support 64 user defined timing sets
- Support source synchronous clocking

Programmable AC/DC Parameter

- Programmable power supplies (8 power sources)
- Programmable test speed up to 2400 Mbps (DDR)
- Programmable AC parameters

Best Signal Integrity

- Best signal integrity for high speed I/Os (HSIO)
- Additional general purpose I/Os (GPIO)

Applications

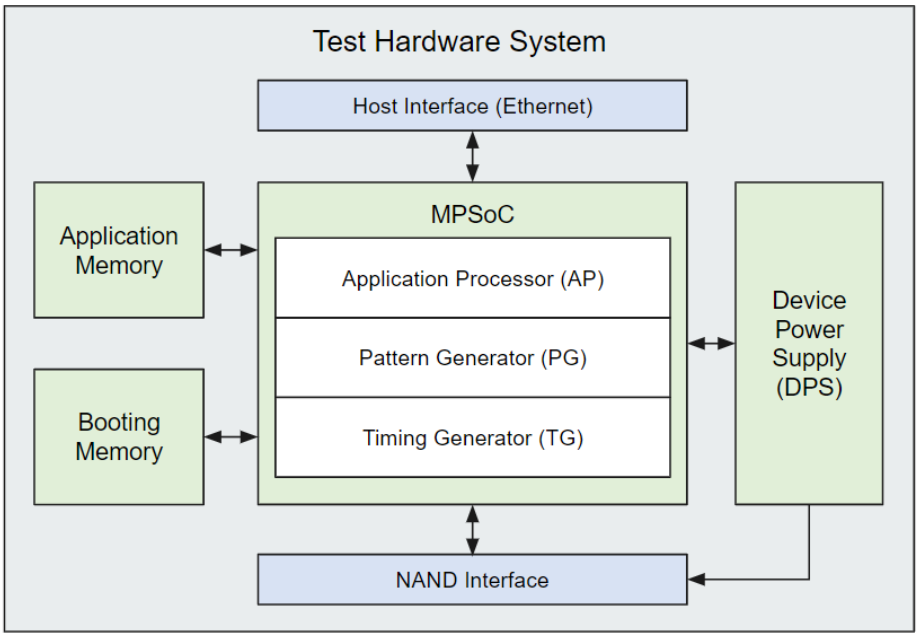
- Engineering test solution for high-speed DDR NAND devices
- NAND flash memory design validation, characterization and failure analysis

2. Hardware Overview

2.1. Test Hardware System

The main tester hardware has the following components.

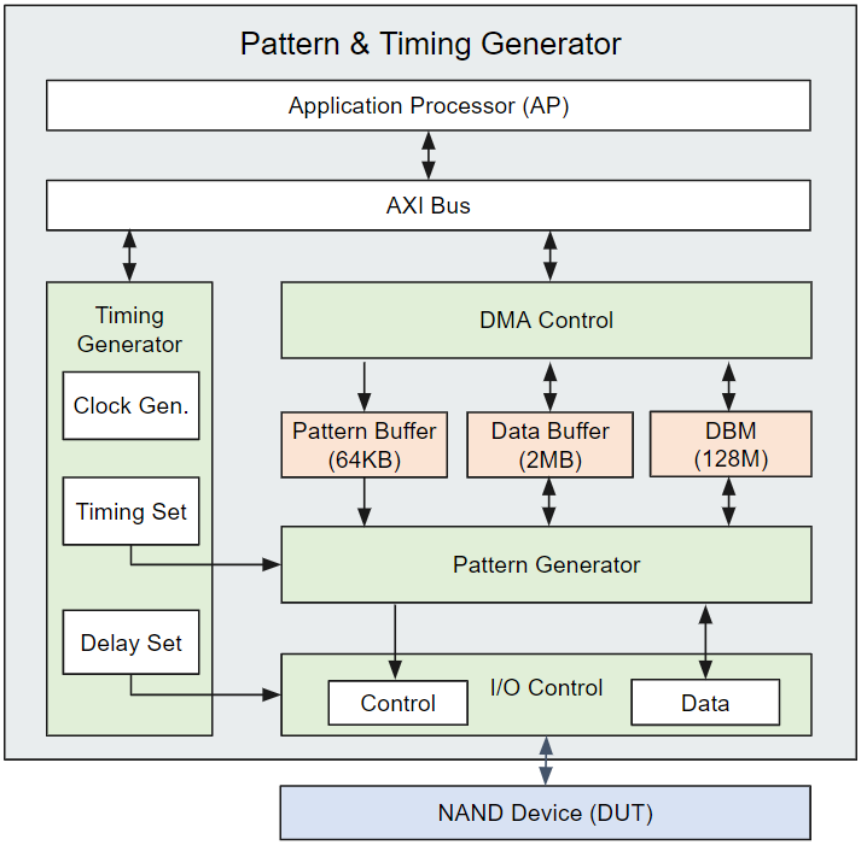
- World's fastest Dual Arm® Cortex®-A53 + Programmable Logic (MPSoC) for the test controller.
- Our patented pattern generator that integrates software and hardware enables high-speed pattern generation and real-time timing control.
- Source synchronous clocking and best signal integrity design technology support 2.4Gbps DDR NAND interface.
- Precision multi-channel programmable device power supply.
- Gigabit Ethernet interface to connect multiple devices to one host.



Hardware System Specifications		
Processor	Application Processor	Dual-core ARM Cortex-A53
	Real-time Processor	Dual-core ARM Cortex-R5
Memory	Application Memory	8GB (DDR4)
	Booting Memory	16GB (eMMC)
Host Interfaces	Ethernet	1G Ethernet
	USB	USB 2.0(Client), 3.0(OTG)
Debug Ports	UART	USB to UART Bridge
	JTAG	14 pin / 2.0 mm pitch
NAND Interfaces	HSIO	29 pin * 2 CH
	GPIO	8 pin
Device Power Supply	DPS for NAND	4 channels
	DPS for ETC	4 channels
Pattern & Timing Generator	PG	1 ea
	TG	1 ea

2.2. Pattern & Timing Generator

Pattern & Timing Generator consists of the following function blocks and specifications.



Pattern & Timing Generator Specifications		
Pattern & Data Memory	Pattern Buffer	64KB
	Data Buffer	1MB(Write), 1MB(Read)
	Data Buffer Memory (DBM)	64MB(Write), 64MB(Read)
Timing Generator	Clock Speed (SDR mode)	10 ~ 100 MHz
	Clock Speed (DDR mode)	100 MHz ~ 1200 MHz (200 Mb/s ~ 2400 Mb/s)
	Clock Resolution	1 MHz
	Timing Sets	64 ea
	Number of Edges for each Timing Set	64 ea
	Edge Control Range	1 ~ 256 UI
	Edge Control Resolution	1 UI (0.4 ~ 1.0 ns)
	Delay Control Range	0 ~ 1 ns
	Delay Control Resolution	3 ~ 4 ps
NAND Interface Pins	Output (CEn[7:0], Controls)	14 pins * 2-ch
	Input (RBn[3:0])	4 pins * 2-ch
	I/O (DQS, DQ[7:0], DBI)	11 pins * 2-ch
	GPIO	8 pins

Application Processor (AP)

The tester has a dual-core ARM Cortex-A53 CPU. The role of this AP is to execute the test program written by the user to control the overall operation of the test, such as power control, pattern sequence control, timing control, and data generation and comparison.

AXI DMA Controller

The six AXI-DMA channels integrated into the pattern generator are used to transfer data between the AP and the pattern generator logic with low latency and wide bandwidth.

Pattern & Data Memory

The pattern program written by the user is automatically allocated to the following three types of memory areas, so that the pattern generator creates a test vector in real time without timing gaps.

- **Pattern Buffer**
  Basically, all vectors used in the test pattern program are allocated to the pattern buffer and passed to the pattern generator. The pattern generator generates timing by referring to the TSET address, and generates data from the data buffer or DBM by referring to the DATA address.
- **Data Buffer**
  Data Buffer is useful when generating small length data. All temporary data that does not refer to DBM during vector registration is passed through here.
- **Data Buffer Memory (DBM)**
  DBM is suitable for reading or writing large amounts of preloaded data patterns. Users must load the data they want to use into DBM in advance. And a pattern is generated by referring to the offset address of this memory designated by the user when registering the test vector.

Timing Generator

The timing generator consists of a clock generator that generates the reference clock required for testing, a TSET memory that stores the user-defined timing set in units of half cycle (1 UI = TCK / 2) of the reference clock frequency, and a Delay memory for setting the IO delay.

- **Clock Generator**
  Clock Generator is made through high-speed precision PLL and can be programmed up to 1200MHz in 1MHz resolution.
- **TSET Memory**
  Up to 64 user-defined **timing sets** can be stored in the TSET memory. Up to 64 **events** for each pin can be described in one timing set. This event is composed of the **state** and the **edge**, which is the basic dataset of timing.
- **Delay Memory**
  The I/O delay of all pins can be fine-tuned in 3-4ps resolution. (static delay)

Pattern Generator

The pattern generator creates the final output pattern in real time without timing gaps by referring to the TSET address and DATA address from the vector registered in the pattern buffer by the user.

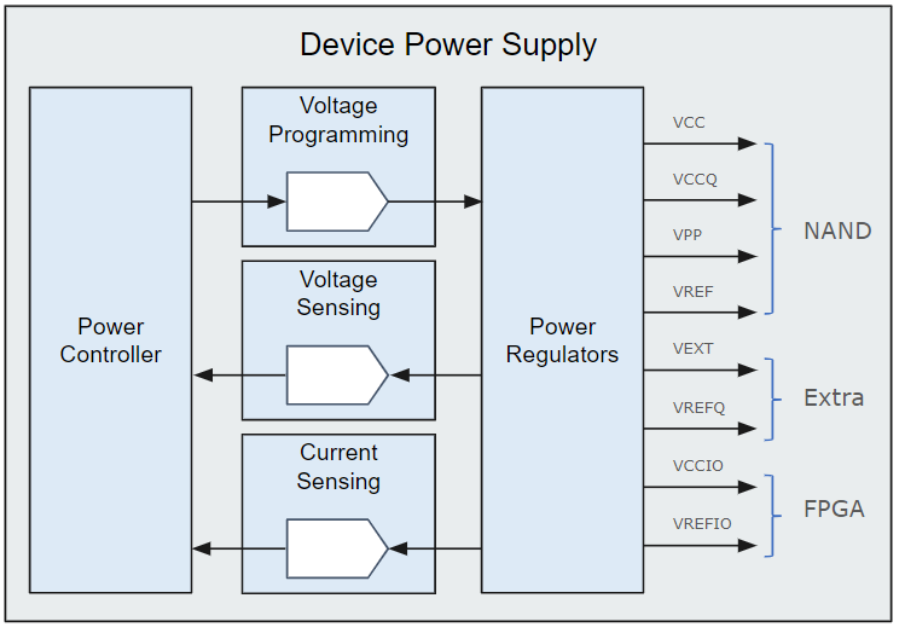
I/O Control

The I/O Control part consists of direction control, delay control, high-speed serialization & deserialization logic for high-speed DDR interface upto 2.4Gbps.

2.3. Device Power Supply

Device Power Supply consists of the following programmable power sources and specifications.
All power supplies have high precision accuracy and program resolution. It is possible to set or measure the voltage level within the setting range and power up/down sequence through the provided API function.

- VCC - NAND Core Power Source
- VCCQ - NAND I/O Power Source
- VPP - NAND High Power Source
- VREF - NAND Reference Voltage Source
- VCCIO - FPGA I/O Power Source
- VREFIO - FPGA Reference Voltage Source
- VREFQ - Extra Power Source
- VEXT - Extra Power Source



Device Power Supply Specifications				
Power Sources	**Setting Range**	**Resolution**	**DC Accuracy**	**Max. Current**
VCC (NAND)	0.8 to 4.0V	<1mV	+/- (1% + 2mV)	3A
VCCQ (NAND)	0.8 to 2.0V	<1mV	+/- (1% + 2mV)	400mA
VPP (NAND)	5.0 to 24V	<1mV	+/- (1% + 20mV)	80mA
VREF (NAND)	0.2 to 4.0V	<1mV	+/- (1% + 2mV)	10mA
VCCIO (FPGA)	0.8 to 2.0V	<1mV	+/- (1% + 2mV)	400mA
VREFIO (FPGA)	0.2 to 2.0V	<1mV	+/- (1% + 2mV)	10mA
VREFQ (Extra)	0.2 to 5.0V	<1mV	+/- (1% + 2mV)	15mA
VEXT (Extra)	1.2 to 30V	<1mV	+/- (1% + 20mV)	10mA

3. Software Overview

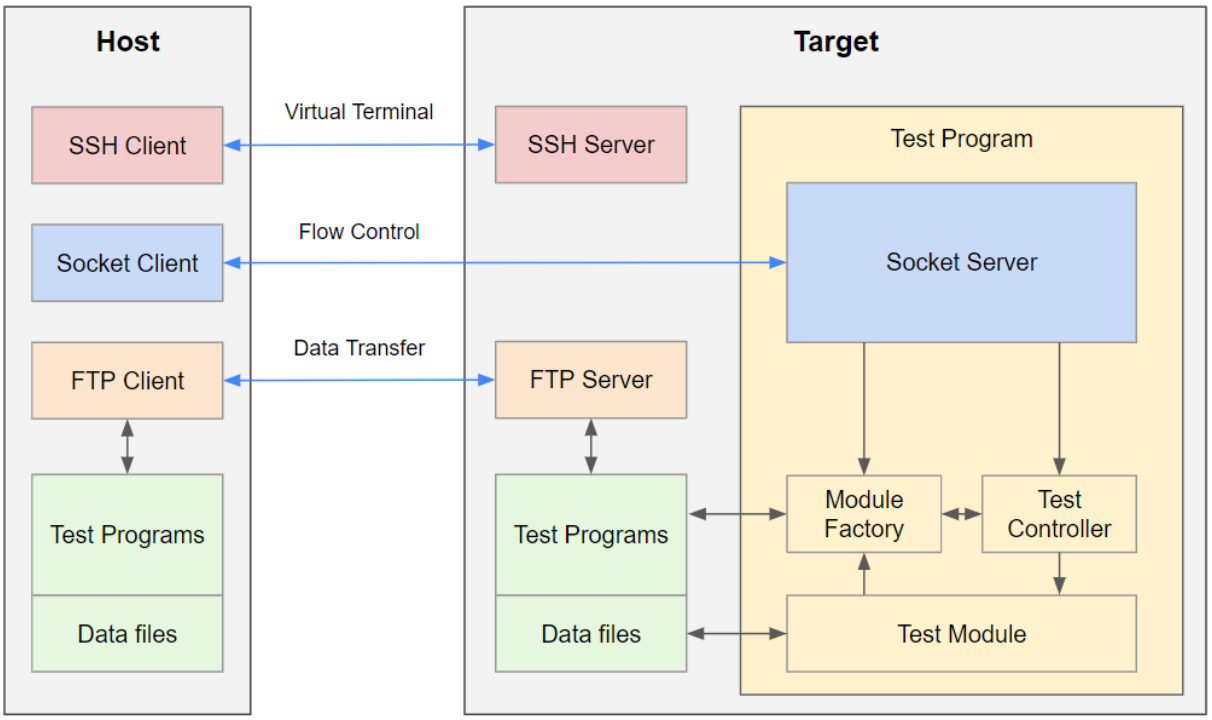
3.1. Software Architecture

The overall software structure for operating the test equipment is as follows.

Host program is a Windows GUI program called **PAS** and provides an easy user interface for operating the tester. Host program is responsible for setting devices, managing test modules, variables and test sequences, displaying execution results, and saving logs.

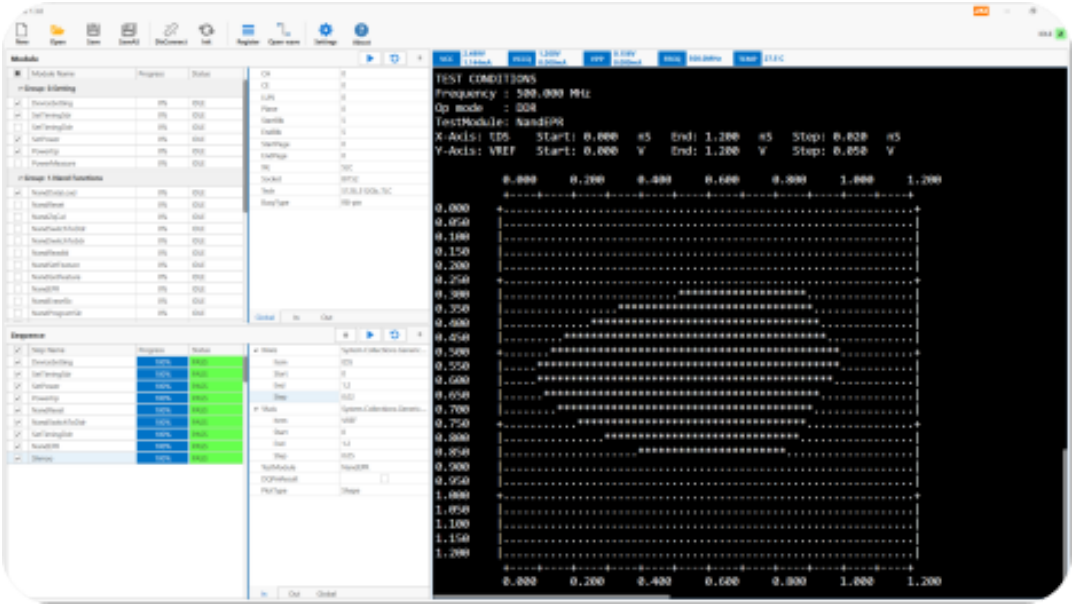
Target program is an executable file built in an Linux-based development environment called **SDK**. The user creates and manages test modules in this program environment, and can select test modules and variables to be linked to the host program according to the desired device or socket type.

Host program and target program are flexibly linked in real time using Ethernet communication protocols (SSH, FTP, TCP/IP).



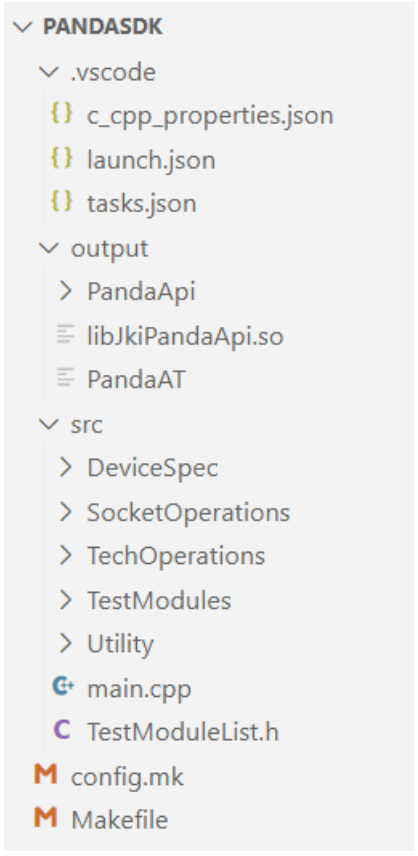
3.2. Host Test Automation Software (PAS)

The PAS software is a test automation software running on Windows OS.



3.3. Target Software Development Kit (SDK)

The SDK is a C++ project provided by JKI to develop device test programs in Embedded Linux.



PandaSDK: top folder in VS code.
vscode: VS code setup files
- + c_cpp_properties.json: compiler path setup
- + launch.json: line by line debugging setup.
- + tasks.json: build short key setup

output: Equipment library and header files
- + libJkiPandaApi.so: JKI equipment API library.
- + PandaApi: API header files
- + PandaAT: final executable file

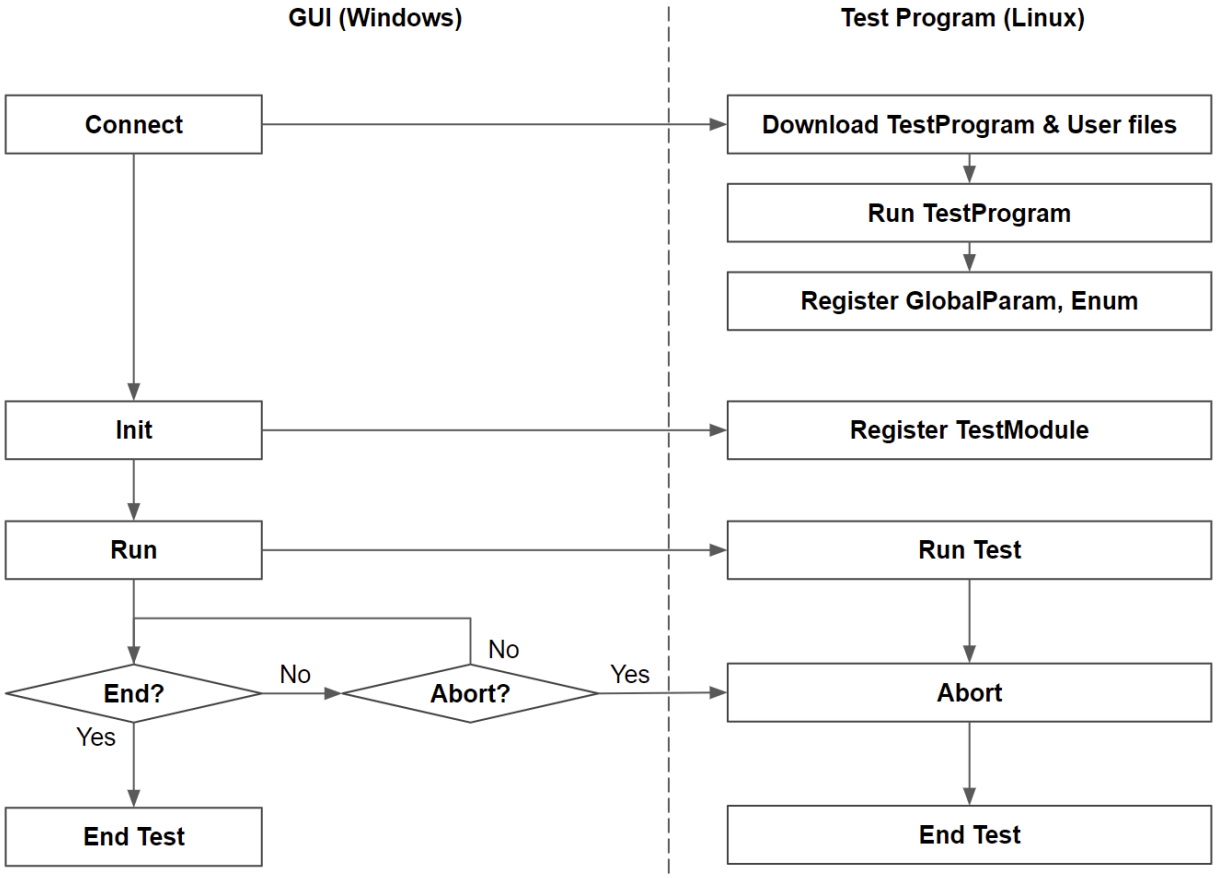
src: User source codes
- + DeviceSpec: define AC, DC spec
- + SocketOperations: define socket specific functions
- + TechOperations: define device specific functions
- + TestModules: define user test modules
- + Utility: Includes many useful functions
- + main.cpp: entry point (do not modify)
- + TestModuleList.h: class that registers test modules

config.mk: Make setup file.
Makefile: Makefile

3.4. Software Flow Chart

The PAS program (Windows GUI) and the test program (Linux SDK) are closely connected, and the message call flow between them is as follows.

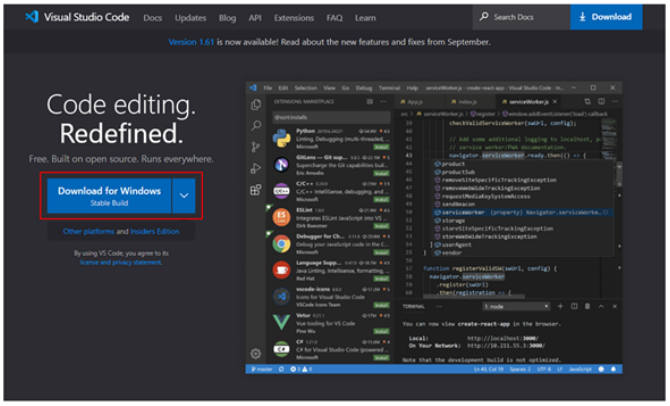
- During the **Connect** process, compiled test programs and custom data files are automatically downloaded from the host to the target.
- In the **Init** process, the test modules of the test program written by the user are uploaded to the host GUI's module list according to the selected device type.
- The module list goes to the Sequence panel, where you can change the execution order and parameters, and you can **Run** and **Abort** any item you want.



3.5. SDK Programming Environment

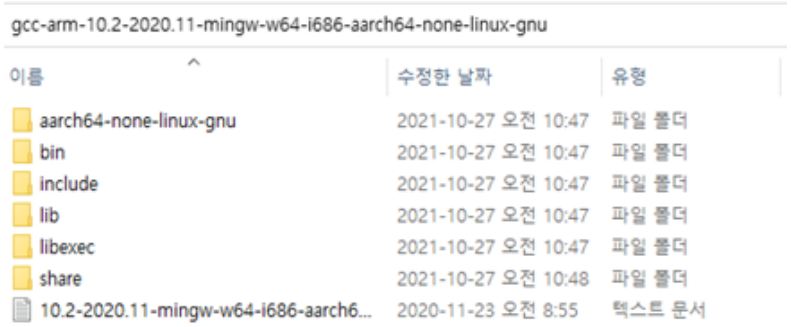
Installing VS Code

Visual Studio Code is a code editor redefined and optimized for building and debugging linux applications.
https://code.visualstudio.com/



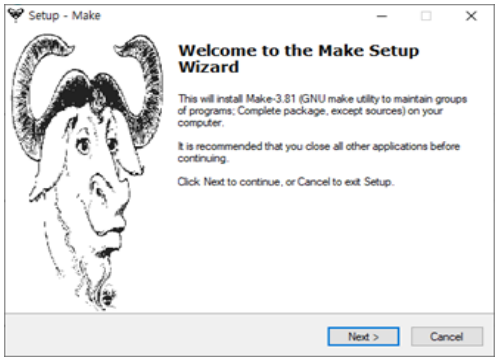
Installing Tool Chain

The GNU Arm Embedded Toolchain is a ready-to-use, open-source suite of tools for C, C++ and assembly programming. Follow the links on this page to download the right version for your development environment. (ARM Cortex-A53 and aarch64-linux-gnu file)
https://developer.arm.com/-/media/Files/downloads/gnu-a/10.2-2020.11/binrel/gcc-arm-10.2-2020.11-mingw-w64-i686-aarch64-none-linux-gnu.tar.xz?revision=86d8332c-f843-476e-a465-debe391a2a73&hash=0A40FAB276CEC6125E5DF48C452C42BAC9DF22CB



Installing Gnu Make

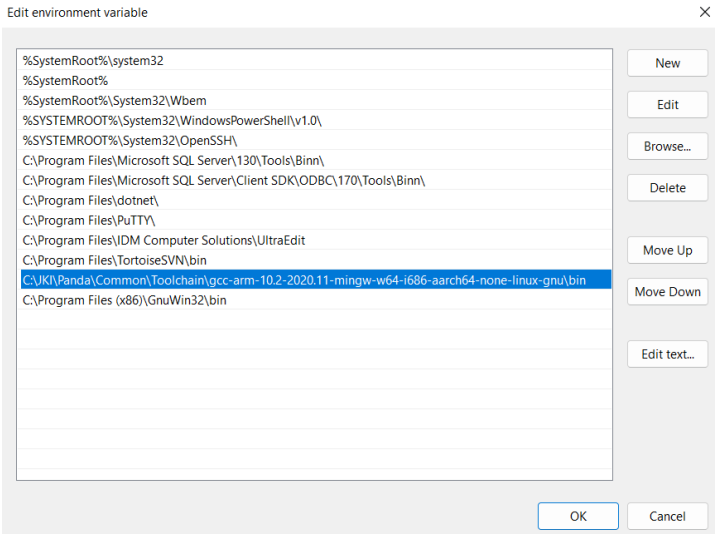
GNU Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files.
http://gnuwin32.sourceforge.net/downlinks/make.php



Setting Environment

You need to register the path of ToolChain and GnuMake in the system environment variable Path. This is necessary to make and build in Visual code.

- ToolChain path: ..\gcc-arm-10.2-2020.11-mingw-w64-i686-aarch64-none-linux-gnu\bin
- GnuMake path: ..\GnuWin32\bin



To build the test program in Visual Studio, register the ToolChain path in the c_cpp_properties.json file.

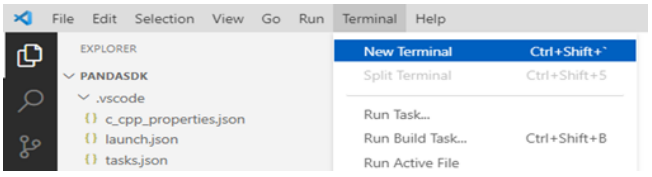
- ..\Panda_SDK-main\Embedded\.vscode\c_cpp_properties.json

PandaSDK > .vscode > {} c_cpp_properties.json > ...

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**",
                "C:/JKI/Panda/Common/Toolchain/gcc-arm-10.2-2020.11-mingw-w64-i686-aarch64-none-linux-gnu/aarch64-none-linux-gnu/libc/usr/include",
                "C:/JKI/Panda/Common/ToolChain/gcc-arm-10.2-2020.11-mingw-w64-i686-aarch64-none-linux-gnu/aarch64-none-linux-gnu/include/c++/10.2.1/bits"
            ],
            "defines": [
                "_DEBUG",
                "UNICODE",
                "_UNICODE"
            ],
            "compilerPath": "C:/JKI/Panda/Common/Toolchain/gcc-arm-10.2-2020.11-mingw-w64-i686-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-g++.exe",
            "cStandard": "c17",
            "cppStandard": "c++17",
            "intelliSenseMode": "linux-gcc-arm64"
        }
    ],
    "version": 4
}
```

Building Source Code

Finally, to build the source, run make in the Visual Studio terminal.



4. Test API Reference

4.1. Types (Structs, Enums, Defines)

All types defined in Test API are as follows.

PinNum

```
enum class PinNum
{
    // Nand interface signals
    CEN,                                                // CE all
    CE0, CE1, CE2, CE3, CE4, CE5, CE6, CE7,             // CEn
    RNB,                                                // RB all
    RB0, RB1, RB2, RB3,                                 // RBn
    ALE, CLE, WEN, REN, WPN,                            // Controls
    DQ0, DQ1, DQ2, DQ3, DQ4, DQ5, DQ6, DQ7, DBI,        // DQx
    DQS,                                                // DQS
    GP0, GP1, GP2, GP3, GP4, GP5, GP6, GP7,             // GPIO

    // Pattern control signals
    RX_STRB ,    // Read Strobe Enable (SDR mode: EDGE, DDR mode: HIGH)
    RX_RST  ,    // Read FIFO Reset (Assert high during read preamble)
    TX_CMD  ,    // Send Command/Address/Data from Pattern Memory
    TX_DATA ,    // Send Data from Data Buffer Memory
    TX_DQS  ,    // DQS Control (SDR mode: Follow the TimingSet,
                 // DDR mode: Toggle during TX_DATA is high)
    DQ_DIR       // IO(DQ, DQS) Direction Control (0: DUT input, 1: DUT output)
};
```

PpsNum

```
enum class PpsNum
{
    VCC, VCCQ, VPP, VREF, VCCIO, VREFIO, VREFQ, VEXT, DUT5V
};
```

TsetNum

```
enum class TsetNum
{
    TSET0, ... TSET63
};
```

PinState

```
enum class PinState
{
    LOW, HIGH, EDGE, PRIOR, HIZ
};
```

DelayType

```
enum class DelayType
{
    Drive, DriveEnable, Strobe
};
```

BusyType

```
enum class BusyType
{
    RB_None = 0x00,
    RB_Pin = 0x80,
    RB_Status = 0x40
};
```

TimingMode

```
enum class TimingMode
{
    SDRMODE, DDRMODE
};
```

CeSelectMode

```
enum class CeSelectMode
{
    Single, Multi
};
```

DdrReadMode

```
enum class DdrReadMode
{
    BOTH, EVEN, ODD
};
```

PowerSpec

```
struct PowerSpec
{
    PpsNum pps;
    double voltage;
    double delay;
    bool enable;
};
```

DcSpec

```
struct DcSpec
{
     PowerSpec VCC = {PpsNum::VCC, 2.5, 0, true};
     PowerSpec VCCQ = {PpsNum::VCCQ, 1.2, 0, true};
     PowerSpec VPP = {PpsNum::VPP, 12.0, 0, false};
     PowerSpec VREF = {PpsNum::VREF, 0.6, 0, true};
     PowerSpec VCCIO = {PpsNum::VCCIO, 1.2, 0, true};
     PowerSpec VREFIO = {PpsNum::VREFIO, 0.6, 0, true};
     PowerSpec VREFQ = {PpsNum::VREFQ, 0.0, 0, false};
     PowerSpec VEXT = {PpsNum::VEXT, 0.0, 0, false};
     PowerSpec DUT5V = {PpsNum::DUT5V, 5.0, 0, true};
};
```

PatternTimeout

```
typedef struct PatternTimeout
{
    double GlobalTimeOut = 2;
    double NoBusyTimeOut = 1;
    double NoReadyTimeOut = 1
    bool NoBusyHalt = false;
```

```
    bool NoReadyHalt = true;
} PatternTimeout;
```

PatternResult

```
typedef struct PatternResult
{
    bool NoSocketBoard;
    double ElapsedTime;
    uint32_t InternalFlag;
    uint32_t ReadCount;
} PatternResult;
```

BusyResult

```
typedef struct BusyResult
{
    bool NoBusy;
    bool NoReady;
    double Time;
} PatternResult;
```

4.2. Units (MKS)

The MKS system of units is a physical system of measurement that uses the meter, kilogram, and second (MKS) as base units. It forms the base of the International System of Units (SI).

```
// MKS Units
#define nV  *1E-09
#define uV  *1E-06
#define mV  *1E-03
#define V   *1E-00
#define nA  *1E-09
#define uA  *1E-06
#define mA  *1E-03
#define A   *1E-00
#define pS  *1E-12
#define nS  *1E-09
#define uS  *1E-06
#define mS  *1E-03
#define S   *1E-00
#define Hz  *1E-00
#define KHz *1E+03
#define MHz *1E+06
#define GHz *1E+09
```

4.3. Tester Base API (class Tester)

Initializing tester

Method	`bool Init();`
Definition	Initializing tester.
Parameters	none
Returns	`bool: return true` if the initialization succeeds; otherwise, `false`.
Remark	After initialization, the tester parameters are all reset.

Set/Get version

Method	`void SetUserVersion(string version);` `string GetUserVersion();`
Definition	Set/Get a user program version.
Parameters	`string version`: version string to set.
Returns	`string: return current user version.`

Set/Get board information

Method	`bool SetBoardInfo(string text);` `string GetBoardInfo();`
Definition	Set/Get board information. .
Parameters	`string text`: board information string to set. (Max 2k byte)
Returns	`bool: return true` if the write succeeds; otherwise, `false`. `string: return board information.`

Set/Get socket board information

Method	`bool SetSocketBoardInfo(string text);` `string GetSocketBoardInfo();`
Definition	Set/Get socket board information. .
Parameters	`string text`: socketboard information string to set. (Max 2k byte)
Returns	`bool: return true` if the write succeeds; otherwise, `false`. `string: return socketboard information.`

4.4. Timing Generator API (class TimingGenerator)

Clock generator

Method	`void SetClock(double freq);` `double GetClock();`
Definition	Set/Get test clock frequency.
Parameters	`double freq:` test frequency to set (Hz) - range: 50 MHz ~ 1200 MHz (DDR: 100 Mbps ~ 2400 Mbps)
Returns	`double: return` set frequency (Hz)
Exceptions	Timeout: PLL lock timeout error.
Remark	For SDR mode, freq should be fixed as 400 MHz. After setting the clock, all I/O blocks are reset.

Method	`double MeasClock();`
Definition	Measure test clock frequency.
Parameters	none
Returns	`double: return` measured frequency (Hz)

Changing modes

Method	`void SetTimingMode(TimingMode mode);` `TimingMode GetTimingMode();`
Definition	Set/Get timing mode
Parameters	`TimingMode mode:` timing mode to set - `TimingMode::SDRMODE:` SDR mode - `TimingMode::DDRMODE:` DDR mode
Returns	`TimingMode:` return set timing mode
Remark	In SDR mode, DQ data is fetched by an internal clock. In DDR mode, DQ data is fetched by an external strobe input (DQS).

Timing generator

Method	`void SetTime(TsetNum tsetNum, PinNum pin,` `std::list<std::pair<PinState, double>> pat);`
Definition	Set timing set for a specified pin. The timing set is changed with the test vector on-the-fly.
Parameters	`TsetNum tsetNum:` timing set number (`enum class TsetNum`) `PinNum pin:` pin number (`enum class PinNum`) `list<pair<PinState, double>> pat`: define a list of patterns <pin state, edge time> - `PinState::LOW:` goto low (0) state

	- `PinState::HIGH:` goto high (1) state - `PinState::EDGE:` make edge event pulse - `PinState::PRIOR:` keep prior state - `PinState::HIZ:` goto high impedance - `double time:` edge time
Returns	none
Exceptions	If the timing set is greater than hardware resources.
Remark	The timing generator edge placement resolution is limited as minimum unit interval time (tUI) which is defined by setting test frequency.

Method	`void AppendTime(TsetNum toTset, std::list<TsetNum> fromTset);`
Definition	Append timing set.
Parameters	`TsetNum tsetNum:` timing set number (`enum class TsetNum`) `std::list<TsetNum> fromTset`: Timing sets to be merged
Returns	none

Method	`void SetDelay(PinNum pin, DelayType delayType, double delay, bool apply = false);`
Definition	Set pin delay for a specified pin.
Parameters	`PinNum pin:` pin number (`enum class PinNum`) `DelayType delayType:` delay type (`enum class DelayType`) - `DelayType::Drive`: drive delay (device input delay) - `DelayType::DriveEnable:` driver enable delay - `DelayType::Strobe:` strobe delay (device output delay) `double delay:` delay time (range: 0 ~ 1.5 ns, resolution: 4 ps) `bool apply:` `true` if the delay to be applied immediately; `false` if the delay to be applied at timing set loading time.(`LoadTimeSet`)
Returns	none
Exceptions	If the delay value is out of range.
Remark	This pin delay is a static delay.

Method	`void LoadTimeSet();`
Definition	Apply frequency and loading timing set to TSET memory.
Parameters	none
Returns	none
Remark	The timing set waveform is resampled by minimum time resolution (tUI) and loaded to TSET memory. All pin delays are also applied.

4.5. Pattern Generator API (class PatternGenerator)

Selecting CH, CE and RB

Method	`void SelectNandChannel(int ch);`
Definition	Select device channel
Parameters	`int ch:` CH number (0 or 1)
Returns	none
Remark	In the case of a device with 4 channels, it is switched through an external switch on the socket board using the gpio pin.

Method	`void SelectChipEnable(int ce, CeSelectMode mode = CeSelectMode::Single );`
Definition	Select device chip enable pin
Parameters	`int ce:` CE pin number (0 ~ 7) or CE pin status (0x00, enable at low) - When mode is Single, CE pin number (0 ~ 7) - When mode is Multi, From bit 0 to 7, CE0 ~ CE7, enable at Low `CeSelectMode mode:` CE usage mode - `CeSelectMode::Single:` Use only one CE pin - `CeSelectMode::Multi:` Use of multiple CE pins
Returns	none

Method	`void SelectBusyPin(int rb);`
Definition	Select device ready & busy pin
Parameters	`int rb:` RB pin number (0 ~ 4)
Returns	none

Selecting RB type

Method	`void SetBusyType(BusyType busyType);`
Definition	Set device ready & busy type
Parameters	`BusyType busyType:` busy type - `BusyType::None:` wait no busy - `BusyType::Pin:` wait busy by pin monitoring - `BusyType::Status:` wait busy by read status cycle
Returns	none
Remark	RB pin should be selected in advance before `BusyType::Pin`. Timing set for the read status should be defined in advance before `BusyType::Status`.

Method	`BusyType GetBusyType();`
Definition	Get device ready & busy type
Parameters	none
Returns	`BusyType busyType: busy type` - `BusyType::None: wait no busy` - `BusyType::Pin: wait busy by pin monitoring` - `BusyType::Status: wait busy by read status cycle`

Method	`void SetReadStatus(TsetNum tsetNum, unsigned char expected, unsigned char mask, unsigned char cmd, unsigned char add1 = 0, unsigned char add2 = 0, unsigned char add3 = 0, unsigned char add4 = 0,);`
Definition	Set Status of BusyType.

Parameters	`TsetNum tsetNum: timing set number`
	`unsigned char expected: read status return data`
	`unsigned char mask: data masking value, data & mask = data (Pass)`
	`unsigned char cmd: Nand read status command`
	`unsigned char addN: Nand read status address`

Returns	none

Changing Read mode

Method	`void SetDdrReadMode(DdrReadMode mode);` `DdrReadMode GetDdrReadMode();`
Definition	Set/Get read data selection mode in DDR mode.
Parameters	`DdrReadMode mode: read mode to set` - `DdrReadMode::BOTH: acquire both data (rising and falling edge)` - `DdrReadMode::EVEN: acquire even data (rising edge only)` - `DdrReadMode::ODD: acquire odd data (falling edge only)`
Returns	`DdrReadMode:` return set read mode
Remark	This function can be used when operating as SDR in DDR mode.(Read ID, Set feature, Get feature)

Adding vectors

Method	`void Add(TsetNum tsetNum, int repeat = 1);`
Definition	Add vector without data
Parameters	`TsetNum tsetNum: timing set number` `int repeat: vector repeat count`
Returns	none
Remark	This function can be used for none-data cycles. (Select CE, Deselect CE)

Method	`void AddWriteData(TsetNum tsetNum, unsigned char data);`
Definition	Add vector with write data
Parameters	`TsetNum tsetNum: timing set number` `unsigned char data: immediate write data`
Returns	none
Remark	This function can be used for command, address cycle and short data write cycles. (Command, Address, Set feature data)

Method	`void AddReadData(TsetNum tsetNum, int length);`
Definition	Add vector for read data
Parameters	`TsetNum tsetNum: timing set number` `int length: read data size in bytes`
Returns	none
Remark	This function can be used for short data read cycles. (Read ID, Get feature data)

Method	`void AddWriteDBM(TsetNum tsetNum, int tsetBytes, int repeat, int DBMaddress);`
Definition	Add vector for write data from DBM.(Data Buffer Memory)

Parameters	`TsetNum tsetNum: timing set number`
	`int tsetBytes: number of data cycle of single timing set`
	`int repeat: total repeat count of current timing set`
	`int DBMaddress: DBM address from which data will be loaded`

Returns	none
Remark	This function can be used for large data write cycles. (Page write) Data should be loaded in advance before using this function.

Method	`void AddReadDBM(TsetNum tsetNum, int tsetBytes, int repeat, int DBMaddress);`
Definition	Add vector for read data to DBM.(Data Buffer Memory)

Parameters	`TsetNum tsetNum: timing set number`
	`int tsetBytes: number of data cycle of single timing set`
	`int repeat: total repeat count of current timing set`
	`int DBMaddress: DBM address from which data will be saved`

Returns	none
Remark	This function can be used for large data read cycles. (Page read)

Method	`void AddBusy(TsetNum tsetNum);`

Definition	Add vector for read/busy waiting
Parameters	`TsetNum tsetNum: timing set number`
Returns	none

Loading/Saving/Clear DBM (Data Buffer Memory)

Method	`void LoadDataToDBM(void *src,int DBMaddress, int length);`
Definition	Load write data to DBM.(Data Buffer Memory)

Parameters	`void *src: user source buffer pointer`
	`int DBMaddress: DBM address to which data will be loaded`
	`int length: total data size in bytes`

Returns	none
Remark	Data should be loaded in advance before using the WriteDBM vector.

Method	`void SaveDataFromDBM(void *dst, int DBMaddress, int length);`
Definition	Save(copy) read data from DBM.(Data Buffer Memory)

Parameters	`void *dst: user destination buffer pointer`
	`int DBMaddress: DBM address from which data will be copied`
	`int length: total data size in bytes`

Returns	none
Remark	Read data should be copied to the user buffer for comparison with expected data.

Method	`void ClearWriteDBM(unsigned char data, int DBMaddress = 0, int length = 0);`
Definition	Clear write DBM with data.

Parameters	`void data: Initial value`
	`int DBMaddress: DBM address from which data will be clear`
	`int length: total data size in bytes`

Returns	none
Remark	If only a data parameter is entered, 64MB of total space is initialized.

Method	`void ClearReadDBM(unsigned char data, int DBMaddress = 0, int length = 0);`
Definition	Clear read DBM with data.

Parameters	`void data: Initial value`
	`int DBMaddress: DBM address from which data will be clear`
	`int length: total data size in bytes`

Returns	none
Remark	If only a data parameter is entered, 64MB of total space is initialized.

Starting pattern & getting result

Method	`void SetPatternTimeout(PatternTimeout patternTimeout);`
Definition	Set pattern timeout and exit conditions.
Parameters	`PatternTimeout patternTimeout:` - `double GlobalTimeOut: A timeout that does not end pattern` - `double NoBusyTimeOut: A timeout that does not cause busy` - `double NoReadyTimeOut: A timeout that does not cause ready` - `bool NoBusyHalt: Exit if busy timeout occurs` - `bool NoReadyHalt: Exit if ready timeout occurs`
Returns	none

Method	`bool StartPattern(std::string patternName = "sim");`
Definition	Start pattern bursting.
Parameters	`string patternName: simulation wave file name for debug`
Returns	`bool: return true` if succeeds; otherwise, `false`.
Remark	From the start of the pattern, the timing of pattern generation is managed by hardware logic, so it is performed without idle time. Therefore, it is preferable to execute time-critical patterns by grouping them into one vector.

Method	`PatternResult GetPatternResult();`
Definition	Get results for current pattern bursting.
Parameters	none
Returns	`PatternResult:` - `bool NoSocketBoard: return false` if socket board detected - `double ElapsedTime:` total elapsed time - `uint32_t InternalFlag:` run status flag - `uint32_t ReadCount:` total read count

Method	`void ReadData(void *dst, int length);`
Definition	Read data from read fifo after running read vector (`AddReadData`)
Parameters	`void *dst: user destination buffer pointer` `int length: total data size in bytes`
Returns	none
Remark	The data size between read vector and read data should be matched.

Method	`double GetBusyTime();`
Definition	Get the last busy time.
Parameters	none
Returns	`double`: busy time (S)

Method	`list<BusyResult> GetBusyTimes();`
Definition	Get all busy time and flag during `StartPattern`.
Parameters	none
Returns	`list<BusyResult>`: - `bool NoBusy`: no busy flag - `bool NoReady`: no ready flag - `double Time`: busy time

Setting GPIO

Method	`void SetGpio(PinNum pin, PinState state);`
Definition	Set GPIO pin output state.
Parameters	`PinNum pin`: GPIO pin number `PinState state`: - `PinState::LOW`: goto low (0) state - `PinState::HIGH`: goto high (1) state - `PinState::HIZ`: goto high impedance
Returns	none

Setting DBI

Method	`void SetDBI(bool readDBI, bool writeDBI);`
Definition	Enable/disable read or write DBI function.
Parameters	`bool readDBI`: `true` to enable read DBI function; otherwise, `false`. `bool writeDBI`: `true` to enable write DBI function; otherwise, `false`.
Returns	none

4.6. Power Source API (class PowerSource)

Setting power up/down sequence

Method	`void SetPowerUpSeq(DcSpec seq);` `void SetPowerDownSeq(DcSpec seq);`
Definition	Set power up/down sequence.

Parameters	`DcSpec seq: struct of PowerSpec`
	`- PpsNum pps: Power source number`
	`- double voltage: Power voltage level`
	`- double delay: Time interval in voltage enable`
	`- bool enable: Enable or disable for each power source`

Returns	none

Appling power up/down

Method	`void PowerUp();` `void PowerDown();`
Definition	Apply power up/down in the order of specified power up/down sequence.
Parameters	none
Returns	none

Set/Get voltage

Method	`void SetVoltage(PpsNum pin, double vol);` `double GetVoltage(PpsNum pin);`
Definition	Set/Get voltage level for a specified power source.
Parameters	`PpsNum pin: Power source number` `double vol: Voltage to set`
Returns	`double: return set voltage`

Method	`double MeasVoltage(PpsNum pin);` `double MeasCurrent(PpsNum pin);`
Definition	Measure voltage or current for a specified power source.
Parameters	`PpsNum pin: Power source number`
Returns	`double: return measured voltage or current`

Short test

Method	`bool ShortTest(PpsNum pin, double thr_min, double thr_max, double* measVol, double* measCur);`
Definition	Short test of NAND power(VCC, VCCQ, VPP) rail.

Parameters	`PpsNum pin: Power source number`
	`double thr_min: Minimum threshold value`
	`double thr_max: Maximum threshold value`
	`double* measVol: measured voltage`
	`double* measCur: measured current`

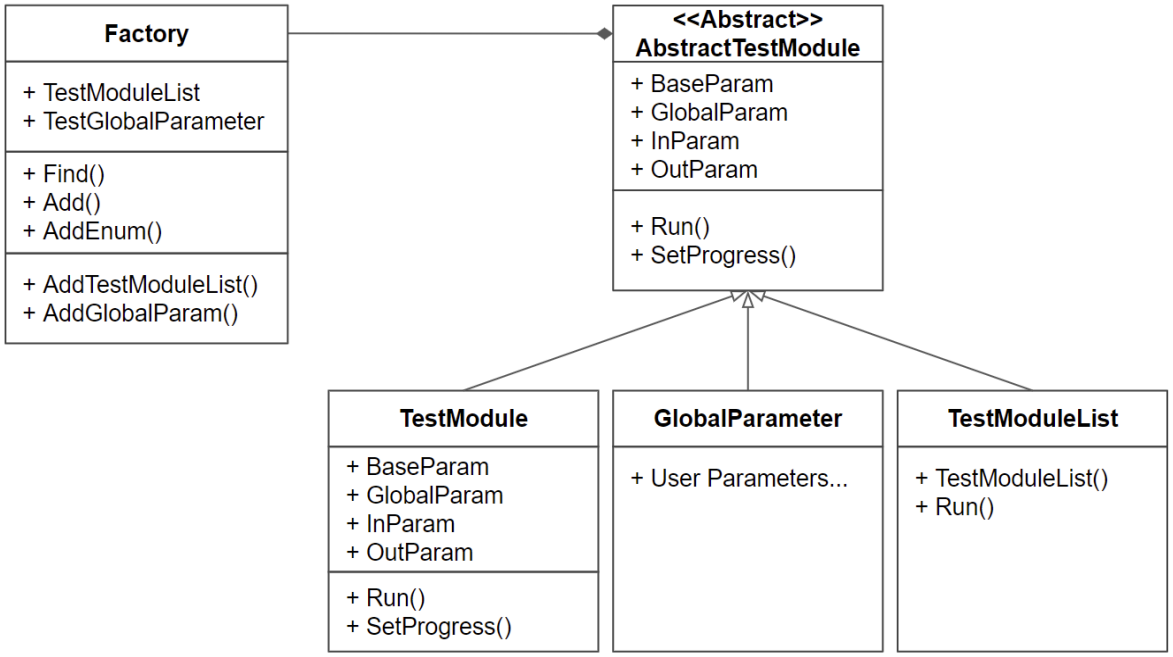
Returns	`bool:` `true` if the current value is within range; otherwise, `false`.

5. Creating Test Program

5.1. Creating Test Module

A new user TestModule can be created by inheriting the AbstractTestModule abstract class (refer to AbstractTestModule.h) included in the SDK we provide. AbstractTestModule class includes Base, Global, In, OutParam properties and methods such as Run and SetProgress.
In TestModule, you can add your own code by overriding these members.
The parameters added by the user must be registered in the `_JSON_` macro to be synchronized with the GUI.

- **BaseParam**: Basic properties associated with the UI
- **InParam**: A set of input parameters associated with the UI.
- **OutParam**: a set of output parameters associated with the UI
- **GlobalParam**: local variable corresponding to user global variable



Code example(NandReadId.h)

```
#pragma once
#include "AbstractTestModule.h"
#include "GlobalParameter.h"
#include "CommonTech.h"
#include "utility.h"
#include "log.h"

class NandReadId : public AbstractTestModule
{
public:
    NandReadId()
    {
        // Communicate parameters with UI
        BaseParam.Enable = false;                // Module enable option
        BaseParam.Group = "Nand Functions";      // Module group
        BaseParam.Description = "Nand Read Id."; // Module description
        BaseParam.Status = "IDLE";               // Module status
        BaseParam.Progress = 0.0;                // Module progress

        // Send files from target to host pc automatically
        BaseParam.FolderName = "Result";         // Folder with result files
        BaseParam.FileDownload = false;          // Folder download option
    }
    GlobalParameter GlobalParam;

    struct InParam
    {
        //-- User input parameters here !!!
        string ID[6] = {"0x01", "0x02", "0x03", "0x04", "0x05", "0x06"};

        //-- Add all parameters to json
        _JSON_(InParam, ID);
    } InParam;

    struct OutParam
    {
        //-- User output parameters here !!!
        u8 Read_ID[6] = {0,};

        //-- Add all parameters to json
        _JSON_(OutParam, Read_ID);
    } OutParam;

    //-- Add all parameters to json
    _JSON_(NandReadId, GlobalParam, BaseParam, InParam, OutParam);

    //-- The execution behavior is described here.
    bool Run()
    {
        BaseParam.Status = "Reading...";
        tech->ReadId(OutParam.Read_ID, 6);
        INFO("ID : %s", Util::ArrToHex(OutParam.Read_ID, 6).c_str());
        SetProgress(100);
        return true;
    }
};
```

5.2. Registering Test Module

Define TestModuleList class to register user-defined TestModule, Enumeration and GlobalParameter classes and display them in GUI. In the TestModuleList class, the user registers the desired module by using the following PandaFactory class method (refer to PandaFactory.h). It is also possible to create a different module list for each device or socket type by using an appropriate branch.

Adding Global Param

Method	`template <class T> void AddGlobalParam()`
Definition	Add GlobalParameter class object.

Adding Test Module

Method	`template <class T> void Add()`
Definition	Add Test Module class object.

Adding Enum

Method	`template <class T> void AddEnum()`
Definition	Add an Enumeration.

Code example(TestModuleList.h)

```
#include "PandaFactory.h"
#include "Enumerations.h"
#include "GlobalParameter.h"
#include "NandReset.h"
#include "NandReadId.h"

class TestModuleList : public AbstractTestModule
{
public:
    TestModuleList()
    {
        factory->AddGlobalParam<GlobalParameter>();
        factory->AddEnum<_Tech>();
    }
    bool Run()
    {
        if(global->Socket == _Socket::BI152)
        {
            socket = new BI152();
            factory->Add<NandReset>();
        }
        if(global->Tech == _Tech::Nand_512G)
        {
            tech = new Nand_512G();
            factory->Add<NandReadId>();
        }
    }
}
```

5.3. Accessing Test Module

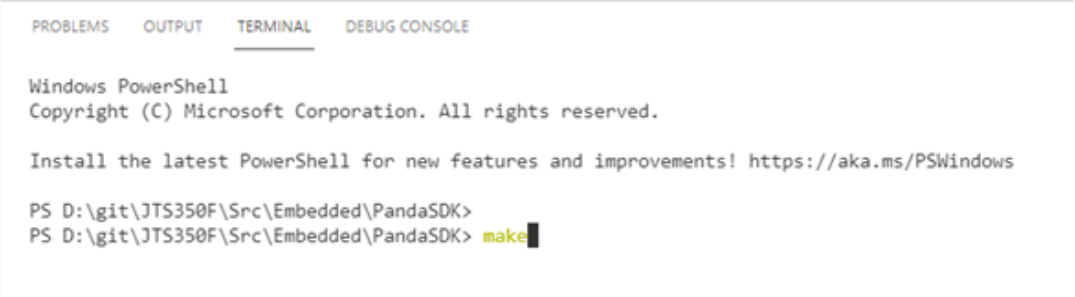
To find the TestModule registered in the user TestModuleList and access the object, you can use the Find method in the PandaFactory class.

Method	`AbstractTestModule *Find(string moduleName);`
Definition	Find Test Module class object.
Parameters	`string moduleName` : Test Module class name.
Returns	`AbstractTestModule *` : Test Module Instance pointer.

```
bool Run()
{
    NandReadId* m_ReadId = (NandReadId*)factory->Find("NandReadId");
    INFO("ID : %s", util->ArrToHex(m_ReadId.OutParam.Read_ID, 6).c_str());
}
```

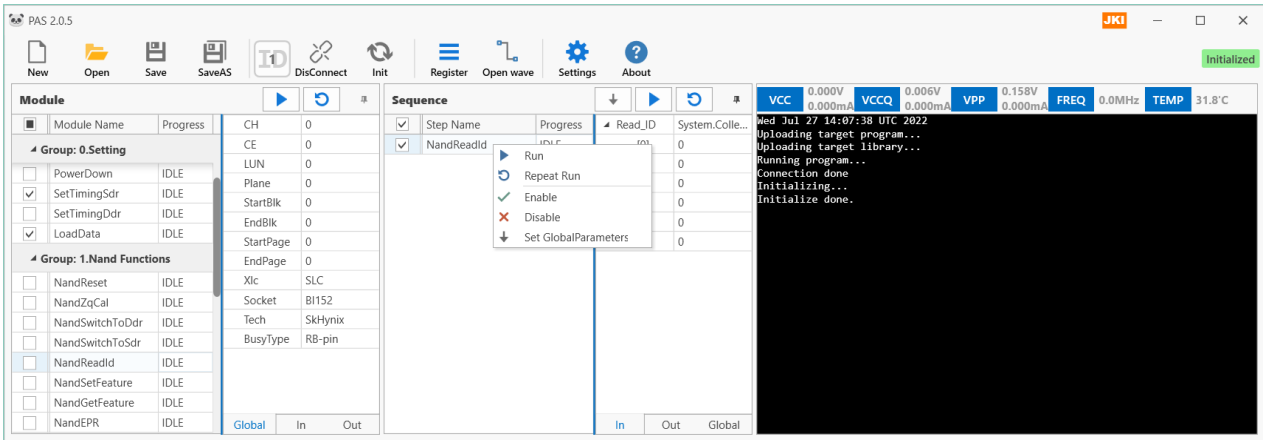
5.4. Building Test Program

After writing the test program, build it with the "make" command in the Visual Code terminal. The built executable files are created in the PandaSDK\output folder.



5.5. Running Test Program

To run the created test program, copy the executable files to the test program path in the GUI setting menu. If you click the Connect button, it is automatically downloaded to the tester. If you press the Init button, the modules and parameters defined by the user appear on the module list panel, and you can execute it by pressing the Run button.



6. Example Test Program

The example described here is a simple code snippet for NAND operation, please refer to the separately provided NAND SDK package for the complete code.

6.1. Defining Device Specification

Defining DC specification (Refer to DefineSpec.h)

As in the example, DC Power Spec and Power Up/Down sequence can be defined.

```
DcSpec dcSpecUp
{
    .VCC = {Panda::PpsNum::VCC, 2.5, 0, true},
    .VCCQ = {Panda::PpsNum::VCCQ, 1.2, 0, true},
    .VPP = {Panda::PpsNum::VPP, 12.0, 0, false},
    .VREF = {Panda::PpsNum::VREF, 0.6, 0, true},
    .VCCIO = {Panda::PpsNum::VCCIO, 1.2, 0, true},
    .VREFIO = {Panda::PpsNum::VREFIO, 0.6, 0, true},
};
```

Defining AC specification (Refer to DefineSpec.h)

As in the example, the AC timing spec of the NAND device can be defined.

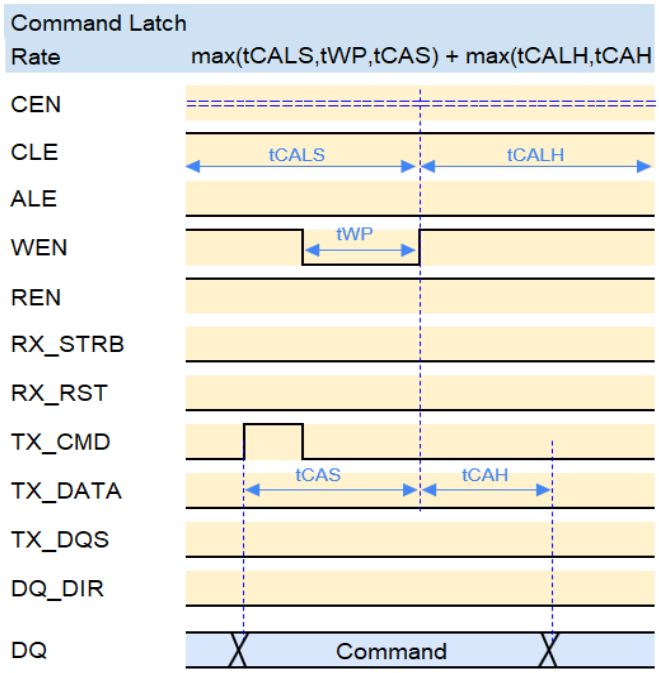
```
struct AcSpecDdr
{
    // Key parameters
    double FreqMHz = 500;
    bool AutoCenter = true;
    double tUI = ((1 / (FreqMHz MHz)) * 1E09) / 2; // nS

    double tCS = 150;                       // CE Setup Time
    double tCH = 10;                        // CE Hold Time
    double tCEH = 40;                       // CE High Hold Time
    double tCALS = 20;                      // CLE/ALE Setup Time
    double tCALH = 20;                      // CLE/ALE Hold Time
    double tCAS = 20;                       // Command/Address Setup Time
    double tCAH = 20;                       // Command/Address Hold Time
    double tCSD = 20;                       // CLE/ALE/WE Hold Time from CE High
    double tCHZ = 30;                       // CE High to Output Hi-Z
    double tWC = 20;                        // Write Cycle Time
    double tWP = 20;                        // WE Pulse Width
    double tADL = 500;                      // Address to Data Loading Time
    double tWHR = 240;                      // WE High to RE Low
    double tRR = 40;                        // Ready to RE Low
    double tDS = tUI / 2;                   // Data Setup Time
    double tCDQSS = 200;                    // DQS Setup Time for Data Input Start
    double tWPRE = 30;                      // Write Preamble
    double tWPST = 13;                      // Write Postamble
    double tWPSTH = 30;                     // Write Postamble Hold time
    double tSTRB = tUI / 2;                 // Read Strobe Time (DDR mode)
    double tDQSRE = 25;                     // RE to DQS and DQ Delay
    double tRPRE = 50;                     // Read Preamble
    double tRPST = 40;                      // Read Postamble
    double tRPSTH = 50;                     // Read Postamble Hold Time
    double tRSTdly = 30;                     // FIFO reset (tester internal)
};
```

6.2. Defining Timing Set

A basic timing set that satisfies the AC timing spec can be defined. The code below is an example of expressing the command cycle as a timing set.

Defining Timing Set (Refer to DeviceSpec.h)




```
//------------------------------------------------------
// Command latch cycle
//------------------------------------------------------

double tS_max = std::max({tWP, tCALS, tCAS});
double tH_max = std::max({tCALH, tCAH});
rate = tS_max + tH_max;

api->TG.SetTime(tset.Command, PinNum::CEN, {{P, 0}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::CLE, {{H, tS_max - tCALS}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::ALE, {{L, tS_max - tCALS}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::WEN, {{L, tS_max - tWP}, {H, tS_max}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::REN, {{H, 0}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::WPN, {{H, 0}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::RX_STRB, {{L, 0}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::RX_RST, {{L, 0}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::TX_CMD, {{L, 0}, {E, tS_max - tCAS}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::TX_DATA, {{L, 0}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::TX_DQS, {{H, 0}, {P, rate}});
api->TG.SetTime(tset.Command, PinNum::DQ_DIR, {{L, 0}, {P, rate}});
```

6.3. Test Pattern Programming

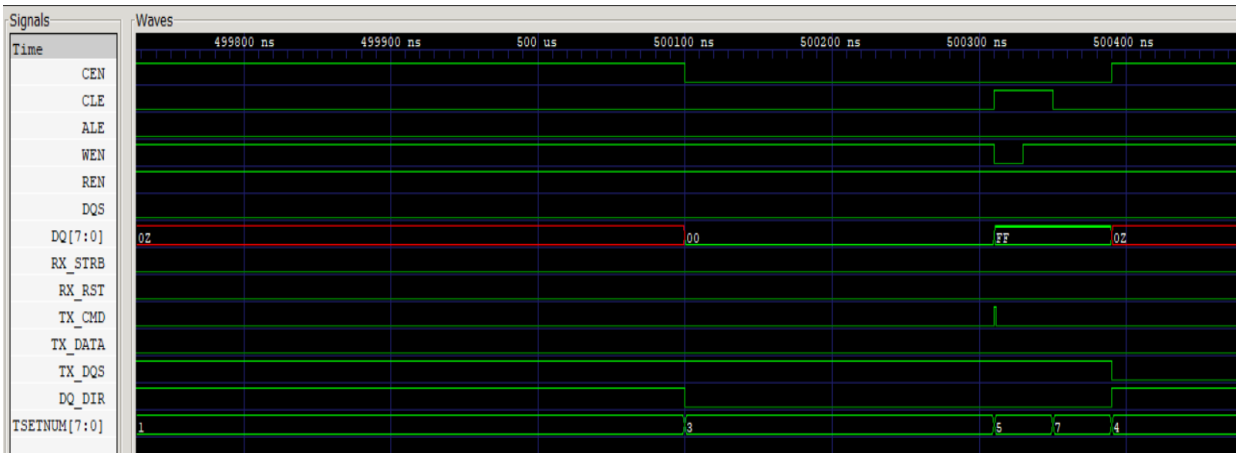
This shows sample code that implements basic NAND operation, as well as simulation waveforms and logic analyzer waveforms obtained by running them.

Reset

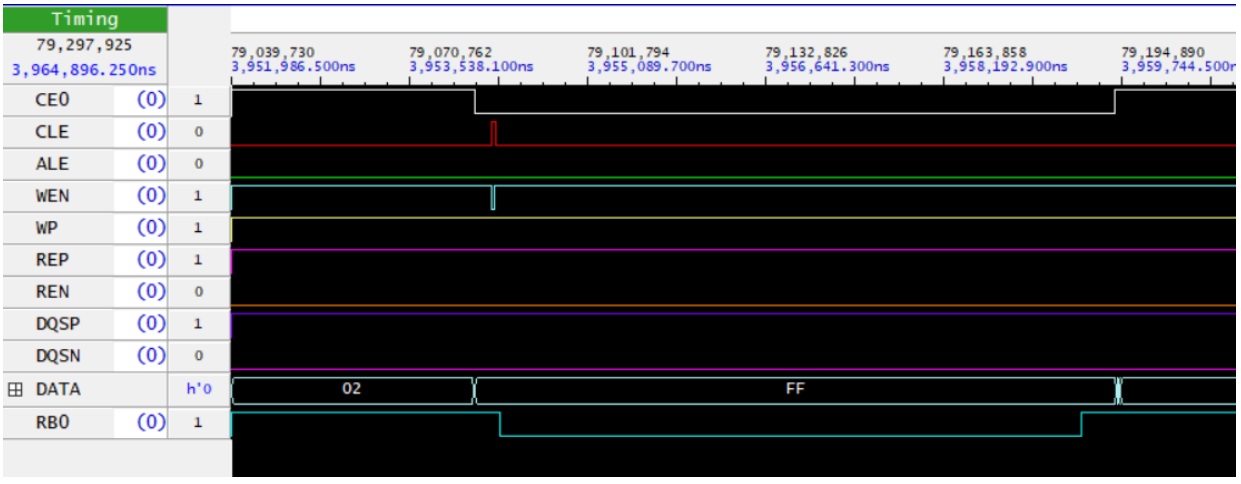
Code

```
api->PG.Add(tset.Init);
api->PG.Add(tset.Wait100nS, 5000);
api->PG.Add(tset.Select);
api->PG.AddWriteData(tset.Command, 0xFF);
api->PG.AddBusy(tset.WaitReady, BusyType::RB_Pin);
api->PG.Add(tset.Deselect);
```

Simulation



Logic

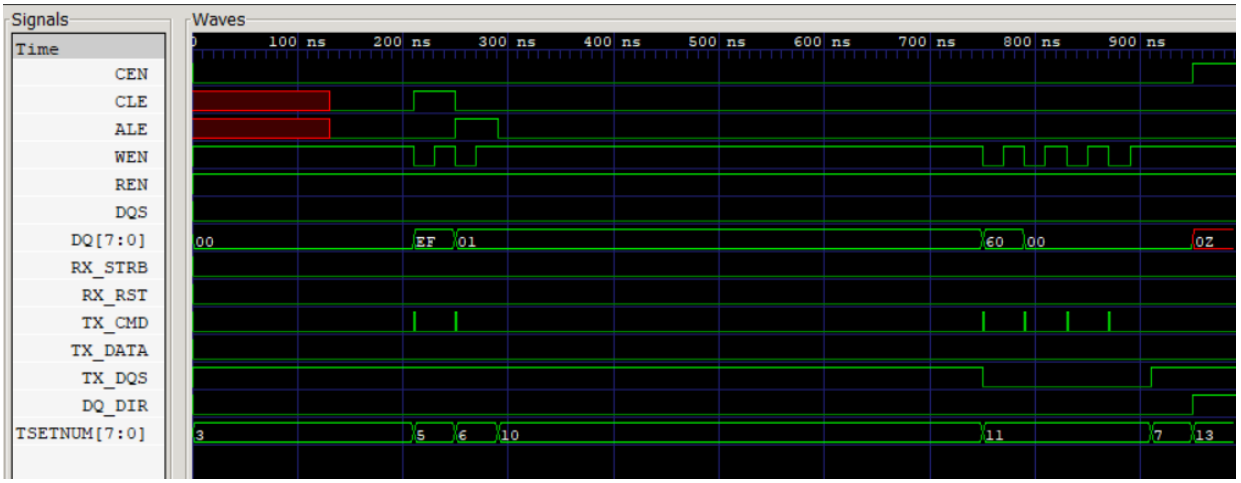


Set feature

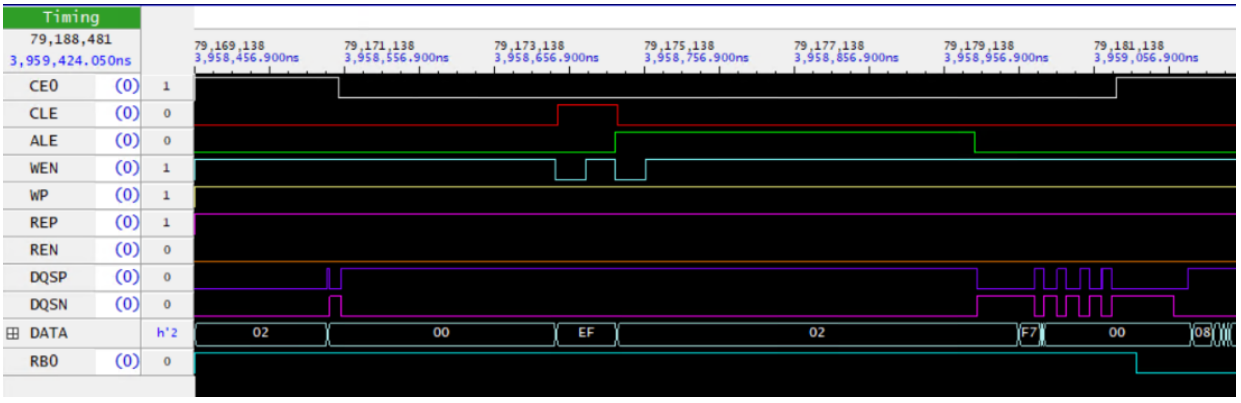
Code

```
api->PG.Add(tset.Select);
api->PG.AddWriteData(tset.Command, 0xEF);
api->PG.AddWriteData(tset.Address, add);
api->PG.Add(tset.WrPre);
for (int i = 0; i < length; i++)
{
    api->PG.AddWriteData(tset.WrData1B, wdata[i]);
}
api->PG.AddBusy(tset.WaitReady, BusyType::RB_Pin);
api->PG.Add(tset.WrPost);
```

Simulation



Logic

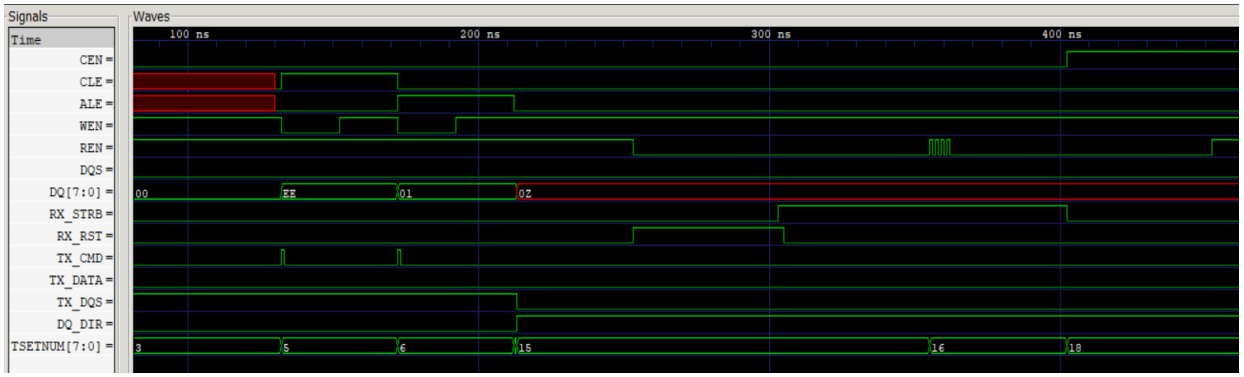


Get feature

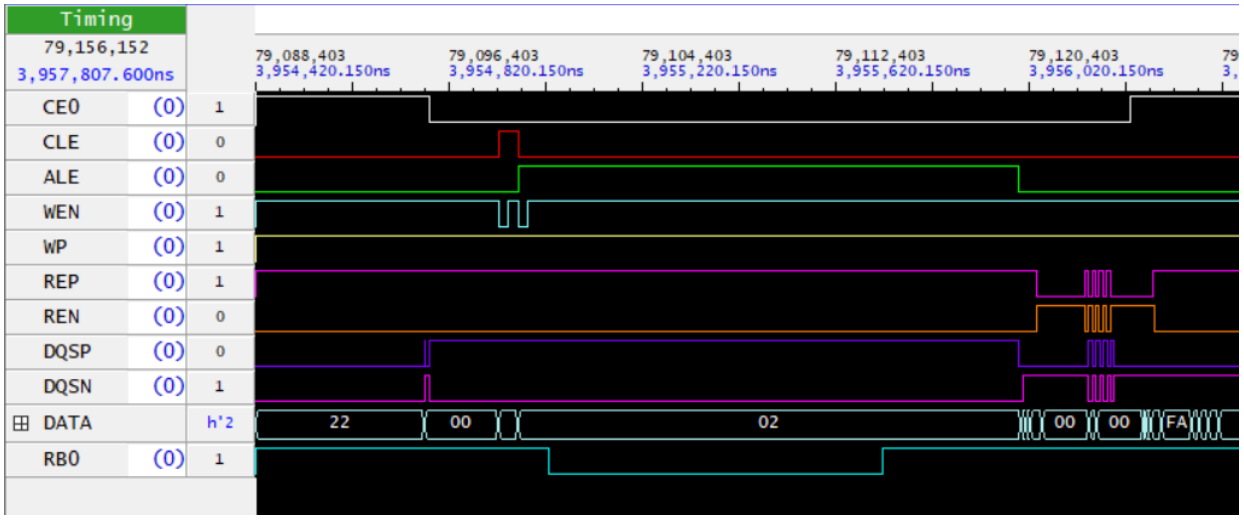
Code

```
api->PG.Add(tset.Select);
api->PG.AddWriteData(tset.Command, 0xEE);
api->PG.AddWriteData(tset.Address, add);
api->PG.AddBusy(tset.WaitReady, BusyType::RB_Pin);
api->PG.Add(tset.RdFromReady);
api->PG.AddReadData(tset.RdData1B, length);
api->PG.Add(tset.RdPost);
```

Simulation



Logic

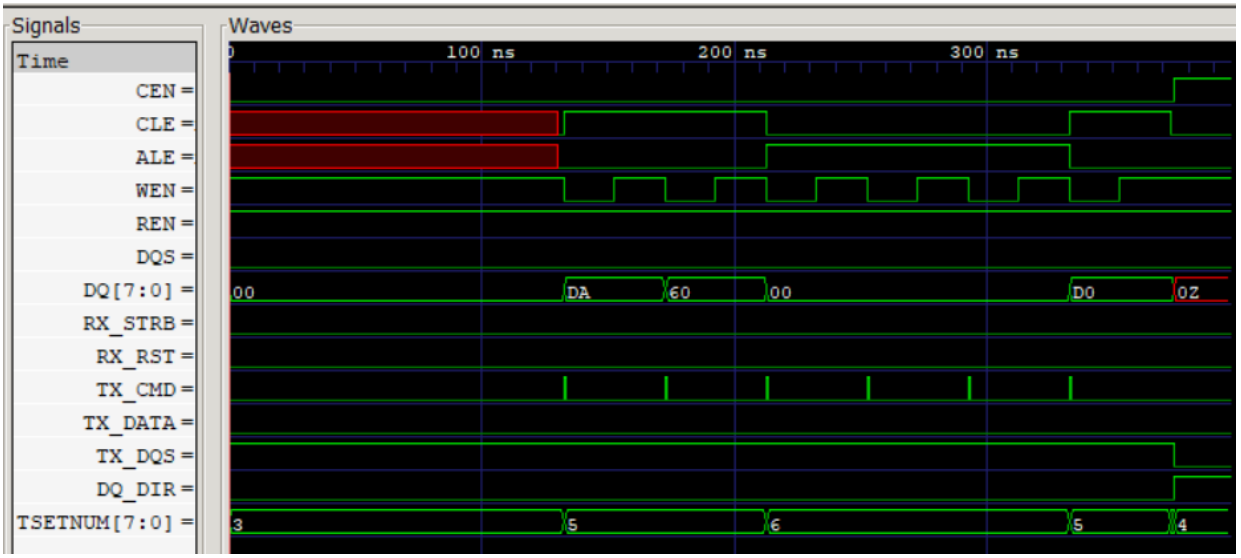


Block Erase

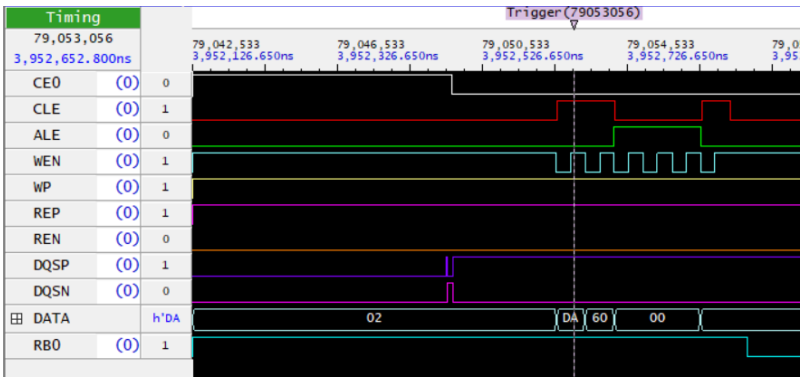
Code

```
api->PG.Add(tset.Select);
api->PG.AddWriteData(tset.Command, 0xDA);
api->PG.AddWriteData(tset.Command, 0x60);
api->PG.AddWriteData(tset.Address, RowAddress_1st);
api->PG.AddWriteData(tset.Address, RowAddress_2nd);
api->PG.AddWriteData(tset.Address, RowAddress_3rd);
api->PG.AddWriteData(tset.Command, 0xD0);
api->PG.AddBusy(tset.WaitReady, BusyType::RB_Pin);
api->PG.Add(tset.Deselect);
```

Simulation



Logic

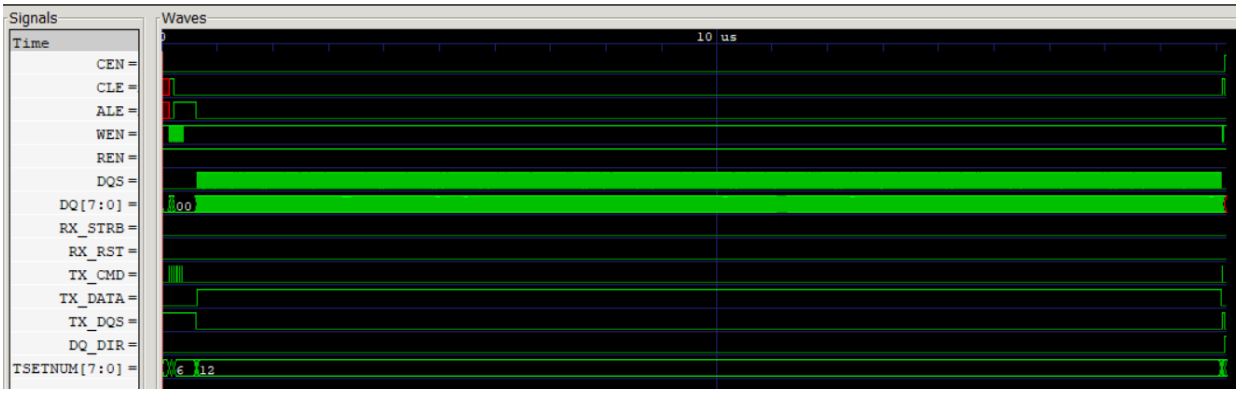


Page Program

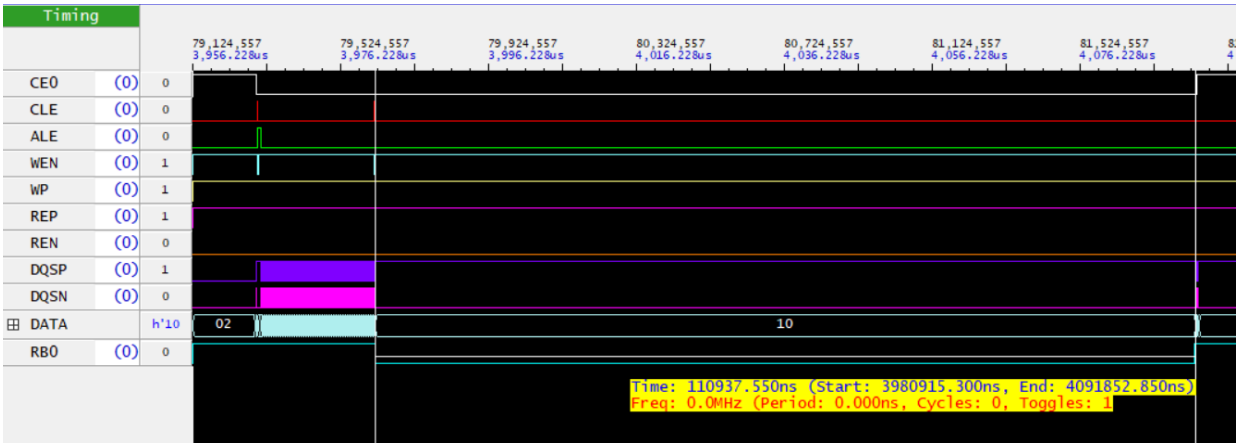
Code

```
api->PG.Add(tset.Select);
api->PG.AddWriteData(tset.Command, 0xDA);
api->PG.AddWriteData(tset.Command, 0x80);
api->PG.AddWriteData(tset.Address, ColAddress_1st);
api->PG.AddWriteData(tset.Address, ColAddress_2nd);
api->PG.AddWriteData(tset.Address, RowAddress_1st);
api->PG.AddWriteData(tset.Address, RowAddress_2nd);
api->PG.AddWriteData(tset.Address, RowAddress_3rd);
api->PG.Add(tset.WrPre);
api->PG.AddWriteDBM(tset.WrDbm8B, 8, length / 8, DBM_Address);
api->PG.Add(tset.WrPostToCmd);
api->PG.AddWriteData(tset.Command, 0x10);
api->PG.AddBusy(tset.WaitReady, BusyType::RB_Pin);
api->PG.Add(tset.Deselect);
```

Simulation



Logic

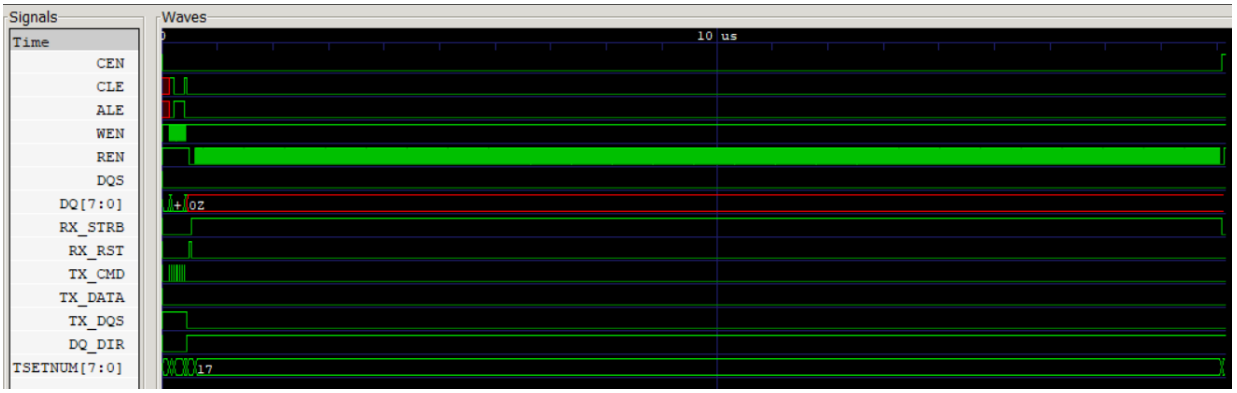


Page Read

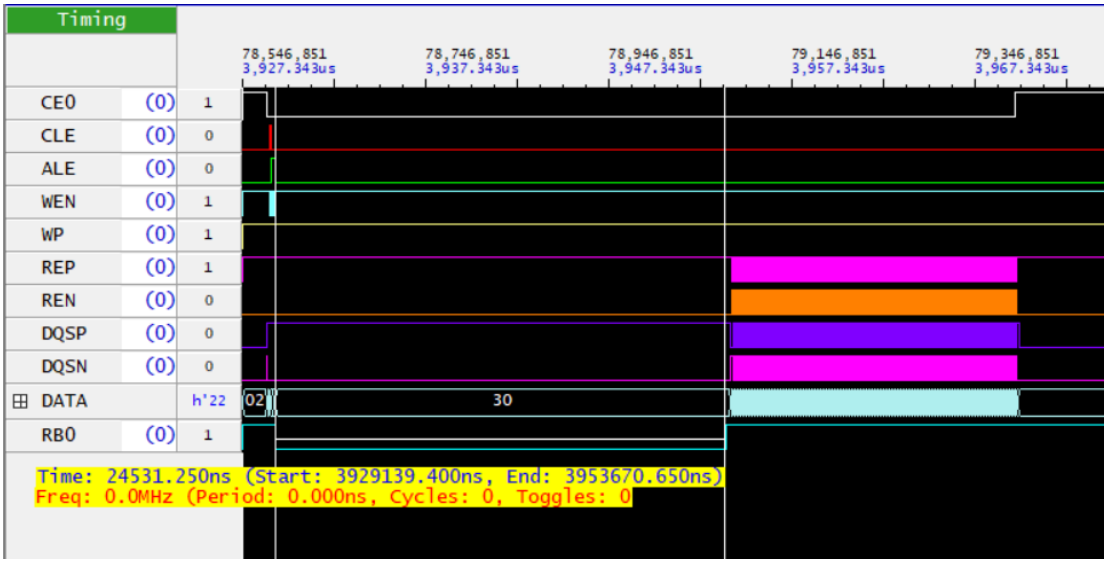
Code

```
api->PG.Add(tset.Select);
api->PG.AddWriteData(tset.Command, 0xDA);
api->PG.AddWriteData(tset.Command, 0x00);
api->PG.AddWriteData(tset.Address, ColAddress_1st);
api->PG.AddWriteData(tset.Address, ColAddress_2nd);
api->PG.AddWriteData(tset.Address, RowAddress_1st);
api->PG.AddWriteData(tset.Address, RowAddress_2nd);
api->PG.AddWriteData(tset.Address, RowAddress_3rd);
api->PG.AddWriteData(tset.Command, 0x30);
api->PG.AddBusy(tset.WaitReady, BusyType::RB_Pin);
api->PG.Add(tset.RdFromReady);
api->PG.AddReadDBM(tset.RdDbm8B, 8, length / 8, DBM_Address);
api->PG.Add(tset.RdPost);
```

Simulation



Logic

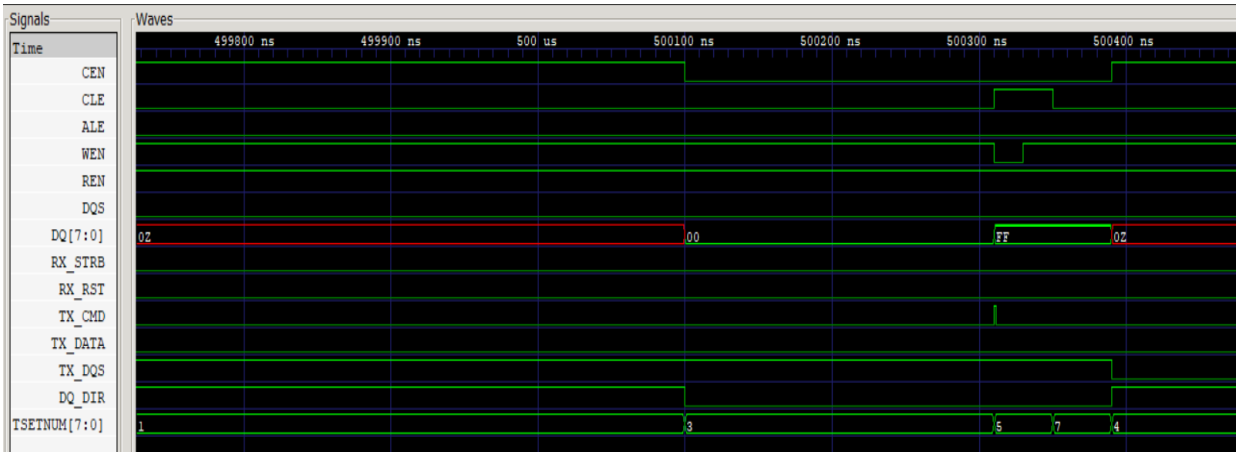


Busy check by read status

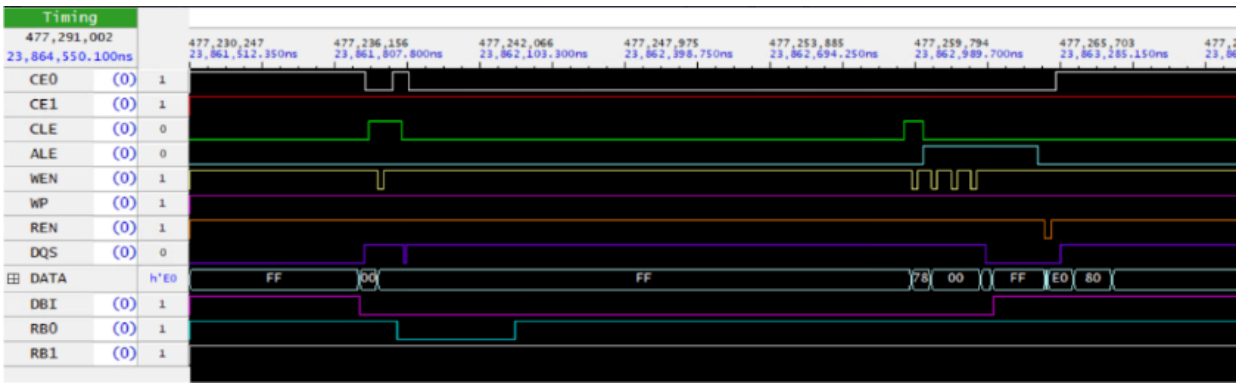
Code

```
api->PG.SetBusyType(BusyType::Status);
api->PG.SetReadStatus(tset.RdStatusEnhanced, 0xE0, 0xE0, 0x78, 0,0,0);
api->PG.Add(tset.Init);
api->PG.Add(tset.Wait100nS, 5000);
api->PG.Add(tset.Select);
api->PG.AddWriteData(tset.Command, 0xFF);
api->PG.AddBusy(tset.WaitReady, BusyType::RB_Pin);
api->PG.Add(tset.Deselect);
```

Simulation



Logic



7. Safety and Warranty

7.1. Safety Summary

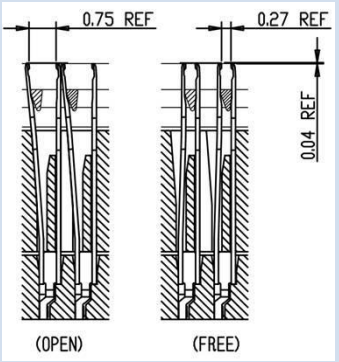
> ***Caution:*** Before operating your JKI product for the first time, please carefully review the safety guidelines below to avoid any injury and damage.

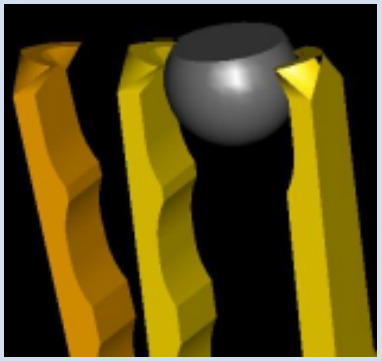
Precaution (ESD)

Since all components of the test board and socket board are exposed to finger touch, special attention should be paid to electrostatic discharge (ESD) precautions. Always get into the habit of first touching the metal surface of one of the USB or Ethernet connectors with both hands for a few seconds before touching any other part of the board. That way you have the same potential as the board, minimizing your ESD risk.

Precaution (EOS)

Do not connect the DUT board while the main board is powered on. Be sure to connect the DUT board while the power is off. Also, when replacing a device in the socket, it must be done with the power off. The pin structure of certain sockets may cause damage to the equipment.





7.2. Warranty and Service

This NAND Tester is provided with a full one year warranty, parts and labor.
Contact your JKI Inc. representative for details.

JKI Inc.

Tower 906, 13, Heungdeok 1-ro, Giheung-gu, Yongin-si, Gyeonggi-do, Korea.
Phone: 031-216-0533
Sales: sales@jkic.co.kr
Support: support@jkic.co.kr
Web: www.jkic.co.kr