VITIUG

Voice Intelligent Technology Wake Word and Voice Command Integration User's Guide

Rev. 5 — 18 October 2023 **User guide**

Document Information

Information	Content
Keywords	Voice Intelligent Technology (VIT), VIT Wake Word Engine (VIT WWE), VIT Voice Command Engine (VIT VCE)
Abstract	The Voice Intelligent Technology (VIT) product provides voice services aiming to wake up and control the IoT devices. This guide describes integration of the VIT Wake Word Engine and the VIT Voice Command Engine.

1 Introduction

The Voice Intelligent Technology (VIT) product provides voice services aiming to wake up and control the IoT devices. This guide describes integration of the VIT Wake Word Engine (VIT WWE) and the VIT Voice Command Engine (VIT VCE).

The current version of VIT WWE and VCE supports a low-power VAD (Voice Activity Detection), a Wake Word Text to Model, and Voice Command Text to Model functionalities, see Figure 1.

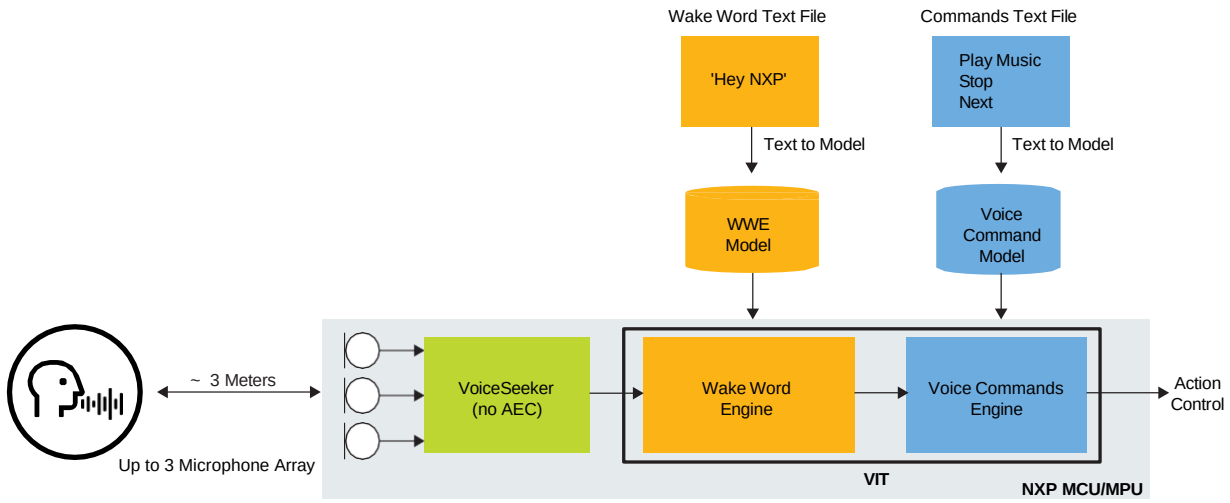

Figure 1. VIT overview

The Wake Word model and the Voice Command model are built from a Text to Model approach, which does not require any audio dataset.

VIT WWE can support the detection of up to 3 wake words in parallel.

The VIT WWE and VCE library has multiple models (each for a different supported language). The model files are named `VIT_Model_xx.h`, where xx represents the two-letter language code. The wake words and voice commands supported by a specific model are listed at the beginning of each model file. Depending on the platform and resources, the application may support switching the model in runtime or by selecting a proper one in application configuration files.

New models of Wake Word and Voice commands can be generated via the VIT online tool.

The role of the low-power VAD is to limit CPU load with minimizing Wake Word / Voice Command processing in silence conditions.

The enablement of the different features of VIT WWE and VCE can be controlled via `VIT_OperatingMode`, see the VIT.h file.

Scenario supported by the VIT library (English model example):

- wake word detection only: “Hey NXP”.
- wake word + voice commands detection. For example, “Hey NXP – Play Music” - “Hey NXP – Next”.

- wake word followed by multi commands Voice Command recognition. For example, “Hey NXP – Play Music – Volume Up”(see Multiturn Voice Command feature description in section Section 4.5)

The voice command must be pronounced in a fix time span that must be adapted to the maximum command length. This time span is controlled via `VIT_ControlParams`. See Section 4.2.1.4 and Section 8 for further details.

VIT VCE returns an the “UNKOWN” command if the audio captured after the wake word does not correspond to any targeted command.

The VIT WWE and VCE library is processing 10-ms and 30-ms audio frame @16 kHz - 16-bit data mono.

The VIT WWE and VCE library has been ported on 6 cores:

- The Cortex-M7 core and validated on the i.MX RT1050, i.MX RT1060, i.MX RT1160, and i.MX RT1170 platforms.
- The Cortex-33 core and validated on the LPC55S69 platform.
- The HIFI4 core and validated on the i.MX RT600 platform.
- The FUSIONF1 core and validated on the i.MX RT500 platform.
- The Cortex-A53 core and validated on the i.MX 8M Mini, i.MX 8M Nano and i.MX 8M Plus platforms.
- The Cortex-A55 core and validated on the i.MX 93 platform.

Note: *Enabling the LPVAD can impact the first keyword detection, it is dependent on the ambient conditions (silence / noise).*

The LPVAD decision is maintained during a hangover time of 15 s after the latest burst detection.

2 Acronyms and abbreviations

Table 1. Acronyms and abbreviations

Acronym	Definition
AFE	Audio Front End
VAD	Voice Activity Detection
VCE	Voice Commands Engine
VIT	Voice Intelligent Technology
WWE	Wake Word Engine

3 Release description

The VIT WWE and VCE release includes the following files:

- `lib/libVIT_PLATFORM_VERSION.a`; PLATFORM can be either HIFI4, FUSIONF1, Cortex-M4, Cortex-M7, Cortex-A53, or Cortex-A55.
- The `lib/VIT.h` file describes VIT WWE and VCE public API.
- The `lib/VIT_Model.h` file contains the VIT WWE and VCE model description for the Wake Word and Voice Command Engine, also this file lists the supported commands.
- The `lib/Inc` folder integrates additional VIT WWE and VCE public interface definitions.
- `ExApp/VIT_ExApp.c or ExApp/VIT_alsa_test_app.c:` VIT WWE and VCE integration example.

4 Public interfaces description

This chapter provides the details of the public interfaces.

4.1 Header files

This section gives the description of the header files.

4.1.1 VIT.h

`VIT.h` describes all the definitions required for VIT WWE and VCE configuration and usage:

- operating mode to enable VIT WWE and VCE features
- detection status enumerator
- instance parameters structure
- control parameters structure
- status parameters structure
- all VIT WWE and VCE public functions.

4.1.2 VIT_Model.h

`VIT_Model.h` contains the model array.

The `VIT_Model` array can be stored in fast or slow memory.

- If the model is stored in slow memory (for example, external flash), library makes the necessary memory reservation to copy part of the model in RAM before using it; current Cortex-M7 case.
- If the model is stored in fast memory (for example, external RAM), library uses the model directly from its original memory location; HIFI4 and FusionF1 cases.

4.1.3 PL_platformTypes_CortexM.h

`PL_platformTypes_CortexM.h` describes the dedicated platform definition for the VIT WWE and VCE library.

4.1.4 PL_platformTypes_HIFI4_FUSIONF1.h

`PL_platformTypes_HIFI4_FUSIONF1.h` describes the dedicated platform definition for the VIT WWE and VCE library.

4.1.5 PL_platformTypes_CortexA.h

`PL_platformTypes_CortexA.h` describes the dedicated platform definition for VIT WWE and VCE.

4.1.6 PL_memoryRegion.h

`PL_memoryRegion.h` describes all the memories definition dedicated to the VIT WWE and VCE handle allocation.

4.2 Public APIs

The VIT library presents different public functions to control and exercise the library:

- `VIT_SetModel`
- `VIT_GetMemoryTable`

- `VIT_GetInstanceHandle`
- `VIT_SetControlParameters`
- `VIT_Process`
- `VIT_GetVoiceCommandFound`
- `VIT_GetWakeWordFound`
- `VIT_GetLibInfo` (subsidiary interface)
- `VIT_GetModelInfo` (subsidiary interface)
- `VIT_ResetInstance` (subsidiary interface)
- `VIT_GetControlParameters` (subsidiary interface)
- `VIT_GetStatusParameters` (subsidiary interface)

For detailed description of the different APIs (Parameters, return values, and usage), see Section 4.1.1..

4.2.1 Main APIs

The main VIT WWE and VCE APIs must be called (in the right sequence) to instantiate, control, and exercise algorithms.

4.2.1.1 VIT_SetModel

To set the model location: `VIT_ReturnStatus_en VIT_SetModel` (`PL_UINT8*pVITModelGroup`, `VIT_Model_Location_en`).

4.2.1.1.1 Goal

Save the address of the VIT WWE and VCE model and check whether the model provided is supported by the library.

4.2.1.1.2 Input parameters

To set the input parameters:

- The address of the model in memory
- The location of the model is in fast or slow memory

4.2.1.1.3 Output parameters

The output parameter is: none.

4.2.1.1.4 Return value

A value of type is `PL_ReturnStatus_en`. If `PL_SUCCESS` is returned, then:

- The model address is saved.
- The model is supported by the library.

4.2.1.2 VIT_GetMemoryTable

```
VIT_ReturnStatus_en VIT_GetMemoryTable(VIT_Handle_t           phInstance,
                                       PL_MemoryTable_st      *pMemoryTable,
                                       VIT_InstanceParams_st  *pInstanceParams);
```

4.2.1.2.1 Goal

The goal is to inform the software application about the required memory needed by the library.

There are 4 kinds of memory:

- Fast data
- Slow data
- Fast coefficient
- Temporary or scratch

4.2.1.2.2 Input parameters

The input parameters are:

1. A pointer to an instance of VIT WWE and VCE. It must be a null pointer as the instance is not reserved yet.
2. A pointer to a memory table structure
3. The instance parameter of the VIT WWE and VCE library

4.2.1.2.3 Output parameters

The memory table structure is filled. It informs about the memory size required for each memory type.

4.2.1.2.4 Return value

A value of type `PL_ReturnStatus_en`. If `PL_SUCCESS` is returned, VIT is succeeding to get memory requirement of:

- Each submodule.
- The model.

4.2.1.3 VIT_GetInstanceHandle

```
VIT_ReturnStatus_en VIT_GetInstanceHandle(
                           VIT_Handle_t           *phInstance,
                           PL_MemoryTable_st      *pMemoryTable,
                           VIT_InstanceParams_st *pInstanceParams);
```

4.2.1.3.1 Goal

The goal is to set and initialize the instance of VIT before processing the call.

All memory is mapped to the required buffer of each submodule.

4.2.1.3.2 Input parameters

The Input parameters are:

1. A pointer to the future instance of VIT.
2. A pointer to the memory table structure. The memory allocation must be done and memory address per memory type has been saved in the table.
3. The instance parameter of the library.

Depending on the value of the instance parameter, the submodule initialization is different.

4.2.1.3.3 Output parameters

The address of the VIT instance is set.

4.2.1.3.4 Return value

A value of type is `PL_ReturnStatus_en`. If `PL_SUCCESS` is returned, then:

- The VIT instance has been set and initialized correctly.
- The VIT model layers are copied in the dedicated memory.

4.2.1.4 VIT_SetControlParameters

```
VIT_ReturnStatus_en VIT_SetControlParameters(
                          VIT_Handle_t                phInstance,
                          const VIT_ControlParams_st  *const pNewParams);
```

4.2.1.4.1 Goal

The goal is to set or modify the control parameter of the VIT instance.

New parameters are not set immediately. Indeed, to avoid processing artifact due to the new parameters themselves the update sequence is under internal processing condition and occurs as soon as possible.

4.2.1.4.2 Input parameters

The Input parameters are:

1. VIT handle
2. A pointer to a control parameter structure: `VIT_ControlParams_st`
   - Operating mode: control enablement of different VIT WWE and VCE features (VAD, AFE, Voice Command modules)
   - `Command_Time_Span`: voice command recognition time span (in seconds)
   - Voice command recognition time span must be adapted to the maximum command length targeted.

For operating mode supported, see VIT.h.

4.2.1.4.3 Output parameters

The output parameter is: none.

4.2.1.4.4 Return value

A value of type is `PL_ReturnStatus_en`. If it is `PL_SUCCESS`, the control parameter structure has been considered and it has to be effective soon.

4.2.1.5 VIT_Process

```
VIT_ReturnStatus_en VIT_Process (  VIT_Handle_t           phInstance,
                                   void                   *pVIT_InputBuffers,
                                   VIT_DetectionStatus_en *pVIT_DetectionResults
                                );
```

4.2.1.5.1 Goal

To detect a “Hot Word” or a voice command, analyze the audio flow.

4.2.1.5.2 Input parameters

The input parameters are:

1. VIT handle
2. Temporal audio samples (160 or 480 samples @16 kHz – 16-bit data).

4.2.1.5.3 Output parameters

The detection status can have 3 different states:

- `VIT_NO_DETECTION`: no detection.
- `VIT_WW_DETECTED`: the wake word has been detected.
- `VIT_VC_DETECTED`: a voice command has been detected.

When `VIT_WW_DETECTED` is returned; VIT switches in the voice commands detection phase for a duration controlled by the `Command_Time_Span`.

When `VIT_VC_DETECTED` is returned; `VIT_GetVoiceCommandFound()` must be called to know which command has been detected.

`VIT_VC_DETECTED` is also indicating the end of the voice command research period and the switch to a wake word detection phase until the wake word is detected again. For further details, see Section 7.

4.2.1.5.4 Return value

A value of type is `PL_ReturnStatus_en`. If it is `PL_SUCCESS`, the process of the new audio frame has successfully been done.

4.2.1.6 VIT_GetVoiceCommandFound

```
VIT_ReturnStatus_en VIT_GetVoiceCommandFound (
                          VIT_Handle_t         pVIT_Instance,
                          VIT_VoiceCommands_t *pVoiceCommand);
```

4.2.1.6.1 Goal

The goal is to retrieve the command ID and name (when present) detected by VIT WWE and VCE.

The function must be called only when `VIT_Process()` informs that a voice command has been detected (*`pVIT_DetectionResults==VIT_VC_DETECTED`).

4.2.1.6.2 Input parameters

The Input parameters are:

1. VIT handle
2. a pointer to a voice commands struct type.

4.2.1.6.3 Output parameters

`pVoiceCommand` must be filled with the ID and name of the command detected.

The `UNKNOWN` command is returned if VIT WWE and VCE does not identify any targeted command during the voice command detection phase.

4.2.1.6.4 Return value

A value of type is `PL_ReturnStatus_en`. If it is `PL_SUCCESS`, `pVoiceCommand` can be considered.

4.2.1.7 VIT_GetWakeWordFound

```
VIT_ReturnStatus_en VIT_GetWakeWordFound (VIT_Handle_t     pVIT_Instance,
                                          VIT_WakeWord_st *pWakeWord);
```

4.2.1.7.1 Goal

Retrieve the Wake Word ID and name (when present) detected by VIT WWE.

The function must be called only when `VIT_Process()` informs that a Wake Word has been detected `(*p VIT_DetectionResults==VIT_WW_DETECTED)`

4.2.1.7.2 Input parameters

The Input parameters are:

1. VIT Handle
2. A pointer to a Wake Word struct type.

4.2.1.7.3 Output parameters

`pWakeWord` is filled with the ID and name of the command are detected.

4.2.1.7.4 Return value

A value of type `PL_ReturnStatus_en`. If `PL_SUCCESS`, `pWakeWord` can be considered.

4.2.2 Secondary APIs

The secondary VIT WWE and VCE APIs are not mandatory for good usage of the algorithms. They can be used to reset VIT WWE and VCE in case of discontinuity in the audio recording flow, (see VIT_ResetInstance description), get information on the VIT WWE and VCE library, model, and on the internal state.

4.2.2.1 VIT_GetLibInfo

```
VIT_ReturnStatus_en VIT_GetLibInfo (VIT_LibInfo_t *pLibInfo);
```

4.2.2.1.1 Goal

This function returns different information of the VIT WWE and VCE library.

4.2.2.1.2 Input parameters

The input parameter is a pointer to the `VIT_LibInfo` structure.

4.2.2.1.3 Output parameters

`VIT_LibInfo` must be filled with the details on VIT WWE and VCE library, see VIT.h.

4.2.2.1.4 Return value

A value of type is `PL_ReturnStatus_en`. If it is `PL_SUCCESS`, *`pLibInfo` can be considered.

4.2.2.2 VIT_GetModelInfo

```
VIT_ReturnStatus_en VIT_GetModelInfo (VIT_ModelInfo_t *pModel_Info);
```

4.2.2.2.1 Goal

This function returns different information of the model registered within VIT WWE and VCE library. The function must be called only when `VIT_SetModel()` is informing that the model is correct (`ReturnStatus == VIT_SUCCESS`).

4.2.2.2.2 Input parameters

The input parameter is a pointer to the `VIT_Model_Info` structure.

4.2.2.2.3 Output parameters

`VIT_Model_Info` must be filled with the details on `VIT_Model`, see Section 4.1.1.

4.2.2.2.4 Return value

A value of type is `PL_ReturnStatus_en`. If it is `PL_SUCCESS`, *`pModel_Info` can be considered.

4.2.2.3 VIT_ResetInstance

```
VIT_ReturnStatus_en VIT_ResetInstance(VIT_Handle_t phInstance);
```

4.2.2.3.1 Goal

Reset the instance of VIT WWE and VCE with instance parameters saved while `VIT_GetInstanceHandle` is called. The reset does not take effect immediately. Indeed, to avoid processing artifact due to the reset itself the reset sequence is under internal processing condition and occurs as soon as possible.

The `VIT_ResetInstance` function must be called whenever there is a discontinuity in the input audio stream. A discontinuity means that the current block of samples is not contiguous with the previous block of samples.

Examples are:

- Calling the VIT process function after a period of inactivity.
- Buffer underrun or overflow in the audio driver.

After resetting VIT instance, VIT must be reconfigured (call to `VIT_SetControlParameters()`) before continuing the VIT detection process (`i.e VIT_Process()`).

4.2.2.3.2 Input parameters

The input parameter is VIT handle.

4.2.2.3.3 Output parameters

The output parameter is: none.

4.2.2.3.4 Return value

A value of type is `PL_ReturnStatus_en`. If it is `PL_SUCCESS`, the reset has been considered and must be effective as soon as possible.

4.2.2.4 VIT_GetControlParameters

```
VIT_ReturnStatus_en VIT_GetControlParameters(
                           VIT_Handle_t          *phInstance,
                           VIT_ControlParams_st *pControlParams);
```

4.2.2.4.1 Goal

Get the current control parameter of the VIT instance.

4.2.2.4.2 Input parameters

The input parameters are:

1. VIT handle
2. A pointer to a control parameter structure.

4.2.2.4.3 Output parameters

The output parameter structure is updated.

4.2.2.4.4 Return value

A value of type is `PL_ReturnStatus_en`. If it is `PL_SUCCESS`, then parameter structure must be updated correctly.

4.2.2.5 GET_StatusParameters

```
VIT_ReturnStatus_en VIT_GetStatusParameters(
                           VIT_Handle_t        phInstance,
                           VIT_StatusParams_st *pStatusParams);
```

4.2.2.5.1 Goal

Get the status parameters of the library.

4.2.2.5.2 Input parameters

The input parameters are:

1. VIT handle
2. A pointer to a status parameter buffer.

4.2.2.5.3 Output parameters

Fill the status parameter structure.

4.2.2.5.4 Return value

A value of type is `PL_ReturnStatus_en`. If it is `PL_SUCCESS`, the status parameters are valid and can be considered.

4.2.2.6 VIT_SetModelUpdate

The model location:

```
VIT_ReturnStatus_enVIT_SetModelUpdate  (
                        VIT Handle t*         phInstance,
                        PL_UINT8* pVITModel, VIT_Model_Location_en);
```

4.2.2.6.1 Goal

This function is used to update the VIT Model. It has several restrictions:

- The new model must be located in the same memory region as the original model (registered via `VIT_SetModel()`)
- The new model must address same language as the original model.
- The new model must be smaller (with a shorter command list) than the original model.

4.2.2.6.2 Input parameters

The input parameters are:

- VIT Handle.
- The address of the model in the memory.
- The location of the model is in the fast or slow memory.

4.2.2.6.3 Output parameters

The output parameter is: none.

4.2.2.6.4 Return value

A value of type is `PL_ReturnStatus_en`. If `PL_SUCCESS` is returned, then:

- The model address is saved.
- The model is supported by the VIT library.

4.3 Programming sequence

See Figure 2 for programming sequence.

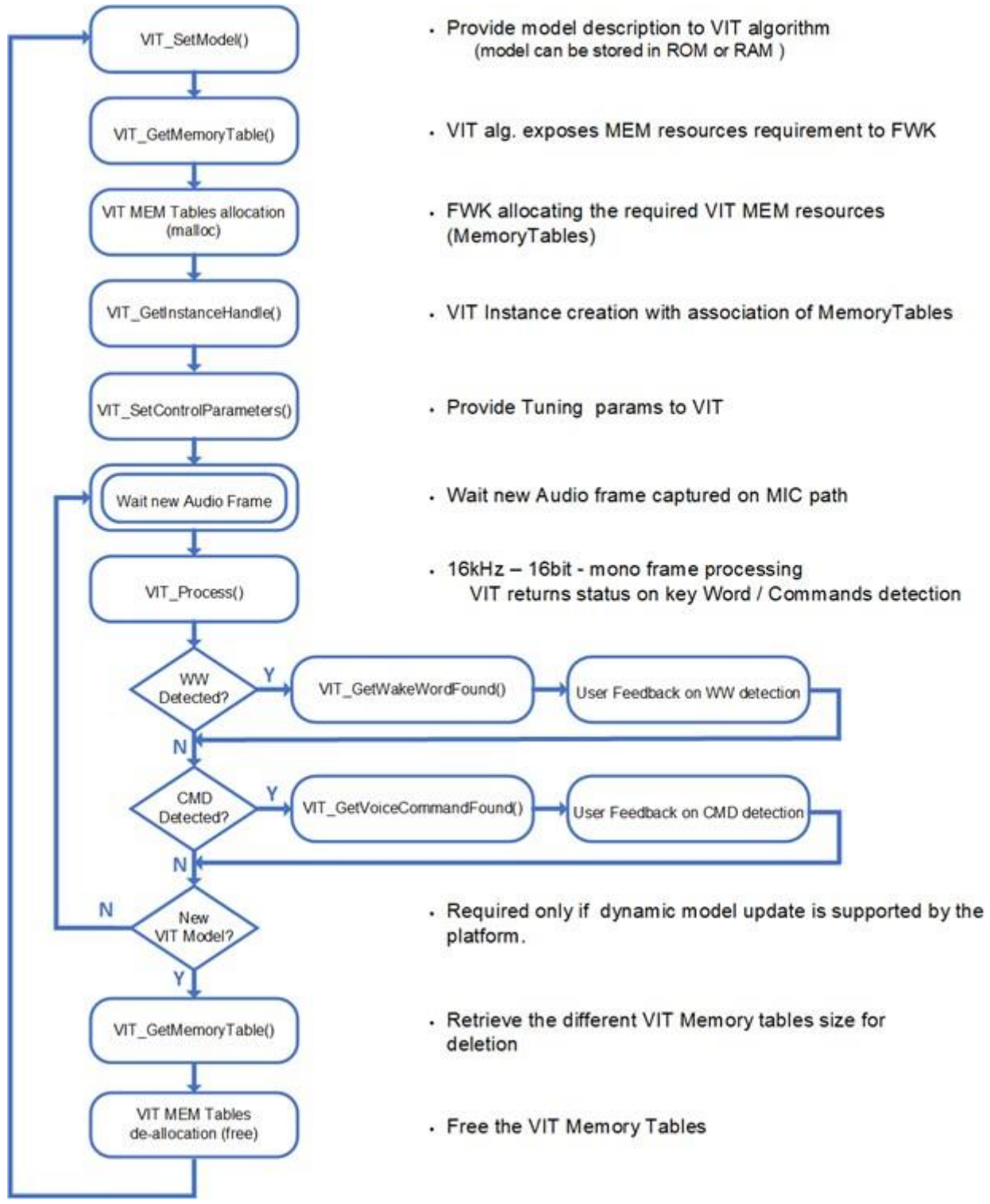

Figure 2. Programming sequence

4.4 Code sample

The code sample in this section is aimed to explain the configuration and usage of the main VIT WWE and VCE interfaces. See `ExApp.c` (available as part of the SDK project) for details.

4.4.1 Initialization phase

The initialization sequence permits setting an instance of VIT. After the initialization sequence, VIT is ready to process audio data. The initialization sequence is in the application code and must respect the following order:

1. Local variable declaration:

```
VIT_Handle_t           VITHandle;         // VIT handle pointer
VIT_InstanceParams_st  VITInstParams;     // VIT instance parameters structure
VIT_ControlParams_st   VITControlParams;  // VIT control parameters structure
PL_MemoryTable_st      VITMemoryTable;    // VIT memory table descriptor
PL_ReturnStatus_en     Status;            // status of the function
VIT_VoiceCommands_t    VoiceCommand;
VIT_DetectionStatus_en VIT_DetectionResults = VIT_NO_DETECTION;
                                          // VIT detection result
PL_INT16              *VIT_InputData;
```

2. Set the instance parameters:
   Software application code set the instance parameters of VIT function.
   As an example:

```
VITInstParams.SampleRate_Hz    = VIT_SAMPLE_RATE;
VITInstParams.SamplesPerFrame = VIT_SAMPLES_PER_FRAME;
VITInstParams.NumberOfChannel = _1CHAN;
VITInstParams.DeviceId         = VIT_IMXRT600;
VITInstParams.APIVersion       = VIT_API_VERSION;
```

3. Set model address:

```
Status = VIT_SetModel(VIT_Model, VIT_MODEL_IN_SLOW MEM);
                                 // Pass the address of the VIT Model
```

4. Get memory size and location requirement:

```
Status = VIT_GetMemoryTable(PL_NULL,
                            &VITMemoryTable,
                            &VITInstParams);
```

5. Reserve memory space:
   Based on the `VITMemoryTable` information, the software application reserve memory space in the required memory type. The start address of each memory type is saved in `VITMemoryTable` structure.

```
#define MEMORY_ALIGNMENT 4

//Following pseudo code applied to MemType =
//PL_MEMREGION_PERSISTENT_SLOW_DATA, PL_MEMREGION_PERSISTENT_COEF and
//PL_MEMREGION_TEMPORARY
if (VITMemoryTable.Region[MemType].Size != 0)
  {
     pMemory = malloc_in_SLOW_MEMORY (VITMemoryTable.Region[MemType].Size +
               MEMORY_ALIGNMENT);
       VITMemoryTable.Region[MemType].pBaseAddress = (void *) pMemory;
```

```
  }
}

//Following pseudo code applied to MemType =
//PL_MEMREGION_PERSISTENT_FAST_DATA
if (VITMemoryTable.Region[MemType].Size != 0)
  {
     pMemory = malloc_in_FAST_MEMORY (VITMemoryTable.Region[MemType].Size +
               MEMORY_ALIGNMENT);
     VITMemoryTable.Region[MemType].pBaseAddress = (void *) pMemory;
  }
}
```

6. Get instance of VIT:

```
VITHandle = PL_NULL;   // force to null address for correct initialization
Status = VIT_GetInstanceHandle(     &VITHandle,
                                    &VITMemoryTable,
                                    &VITInstParams);
```

7. Set control parameters:
   Software application code set the new control parameters and call `VIT_SetControlParameters`:

```
VITControlParams.OperatingMode = VIT_WAKEWORD_ENABLE | VIT_VOICECMD_ENABLE;
VITControlParams.Command_Time_Span = 3.0;        // in second
Status = VIT_SetControlParameters( VITHandle,
                                   &VITControlParams);
```

4.4.2 Process phase

For each new input audio frame, `VIT_Process` is called by the application code.

```
Status = VIT_Process (VITHandle,
                      (void*)VIT_InputData,     // temporal audio input data
                      &VIT_DetectionResults
                      );
```

Check status of the detection:

```
if (VIT_DetectionResults == VIT_WW_DETECTED)
{
   // a Wakeword detected – Retrieve information :
   Status = VIT_GetWakeWordFound(VITHandle, &WakeWord);
   printf("Wakeword : %d detected \n", WakeWord.WW_Id);

   // Retrieve Wakeword name : OPTIONAL
   // Check first if CMD string is present
   if (WakeWord.WW_Name != PL_NULL)
   {
      printf(" %s\n", WakeWord.WW_Name);
   }
}
else if (VIT_DetectionResults == VIT_VC_DETECTED)
{
     // a Voice Command detected – Retrieve command information :
     Status = VIT_GetVoiceCommandFound(VITHandle, &VoiceCommand);
```

```
     printf("Voice Command : %d detected \n", VoiceCommand.Cmd_Id);

    // Retrieve CMD name : OPTIONAL
    // Check first if CMD string is present
     if (VoiceCommand.Cmd_Name != PL_NULL)
     {
       printf(" %s\n", VoiceCommand.Cmd_Name);
     }
}
else
{
    // No specific action since VIT did not detect anything for this frame
}
```

4.4.3 Delete phase

The framework can delete the environment process / task of VIT with stopping calling `VIT_Process`.

There are no specific VIT APIs to free VIT internal memory since the memory allocation is owned by the framework itself (no internal memory allocation).

The framework has to free the memory associated with the different VIT `memoryTables`.

If the framework did not save the `MemoryTables` properties, `VIT_GetMemoryTable` can be called with `VITHandle` to retrieve base addresses and size of different `MemoryTables`.

```
Status = VIT_GetMemoryTable(VITHandle,
                            &VITMemoryTable,
                            &VITInstParams);
// Free memory
for (i = 0; i<PL_NR_MEMORY_REGIONS; i++)
{
    if (VITMemoryTable.Region[i].Size != 0)
    {
        free((PL_INT8 *)VITMemoryTable.Region[i].pBaseAddress);
    }
}
```

4.4.4 Additional code snippet (secondary APIs)

`VIT_GetSatusParameters`

```
VIT_StatusParams_st  VIT_StatusParams_Buffer;
VIT_StatusParams_st* pVIT_StatusParam_Buffer =
 (VIT_StatusParams_st*)&VIT_StatusParams_Buffer;

VIT_GetStatusParameters(VITHandle, pVIT_StatusParam_Buffer,
 sizeof(VIT_StatusParams_Buffer));
printf("\nVIT Status Params\n");
printf(" VIT LIB Release    = 0x%04x\n", pVIT_StatusParam_Buffer-
>VIT_LIB_Release);
printf(" VIT Model Release = 0x%04x\n", pVIT_StatusParam_Buffer-
>VIT_MODEL_Release);
printf(" VIT Features = 0x%04x\n", pVIT_StatusParam_Buffer-
>VIT_Features_Supported);
printf(" VIT Features Selected = 0x%04x\n", pVIT_StatusParam_Buffer-
>VIT_Features_Selected);
```

```
printf(" Nb of channels supported = %d\n", pVIT_StatusParam_Buffer-
>NumberOfChannels_Supported);
printf(" Nb of channels selected = %d\n", pVIT_StatusParam_Buffer-
>NumberOfChannels_Selected);
printf(" Device Selected : device id = %d\n", pVIT_StatusParam_Buffer-
>Device_Selected);
if (pVIT_StatusParam_Buffer->WakeWord_In_Text2Model)
     printf(" VIT WakeWord in Text2Model\n ");
else
     printf(" VIT WakeWord in Audio2Model\n ");
```

4.5 MultiTurn Voice command support

The example above (section Section 4.4) considers a wake-up word followed by recognition of a single voice command.

VIT WWE and VCE library also supports the multi-turn voice command feature. It means that a wake-up word can be followed by multiple recognizable commands.

The amount of time to detect each command is controlled via the `VIT_SetControlParameters` API, see Section 4.2.1.4. The end of the multi-turn sequence (for example, from Voice Command back to wake-up word detection) can be freely controlled by the integrator depending on the specific target use case. The end of the multiturn mode can be based on the detection of a specific command (see the example below) or after a global timeout.

Considering the stage of the process:

In this example, multiturn is re-enabled by default after each Voice Command is detected and disabled after a specific command is recognized: "START" (`VoiceCommand.Cmd_Id == START_CMD_ID`)

See below the special code that controls the multi-turn sequence, consider the additional code in the gray area at the step of defining the command:

For each new input audio frame, `VIT_Process` is called by the application code.

```
Status = VIT_Process (VITHandle,
                      (void*)VIT_InputData,    // temporal audio input data
                      &VIT_DetectionResults
                     );
```

See below the special code that controls the multi-turn sequence, consider the additional code in bold at the command detection phase:

```
if (VIT_DetectionResults == VIT_WW_DETECTED)
{
    // a Wakeword detected - Retrieve information :
    Status = VIT_GetWakeWordFound(VITHandle, &WakeWord);
    printf("Wakeword : %d detected \n", WakeWord.WW_Id);

   // Retrieve Wakeword name : OPTIONAL
   // Check first if CMD string is present
   if (WakeWord.WW_Name != PL_NULL)
   {
         printf(" %s\n", WakeWord.WW_Name);
   }
}
else if (VIT_DetectionResults == VIT_VC_DETECTED)
```

```
{
     // a Voice Command detected – Retrieve command information :
     Status = VIT_GetVoiceCommandFound(VITHandle, &VoiceCommand);
     printf("Voice Command : %d detected \n", VoiceCommand.Cmd_Id);

     // Retrieve CMD name : OPTIONAL
     // Check first if CMD string is present
     if (VoiceCommand.Cmd_Name != PL_NULL)
{
          printf(" %s\n", VoiceCommand.Cmd_Name);
}
//VIT is in command detection phase - we will switch back to WW detection only
// when START
// cmd is detected - otherwise we force to continue in CMD detection mode
if (VoiceCommand.Cmd_Id == START_CMD_ID) // we detect the START cmd here
{
    // back to the default WW/Voice command detection sequence
    VITControlParams.OperatingMode = VIT_WAKEWORD_ENABLE | VIT_VOICECMD_ENABLE;


    VIT_Status = VIT_SetControlParameters(VITHandle, &VITControlParams);
}
else
{
    // force command detection mode (Multiturn voice command mode)
    VITControlParams.OperatingMode = VIT_VOICECMD_ENABLE;


VIT_Status = VIT_SetControlParameters(VITHandle, &VITControlParams);
}

}
else
{
     // No specific action since VIT did not detect anything for this frame
}
```

5 VIT WWE and VCE library profiling

The profiling example for the English model supports 12 commands (with WW in Text to Model and voice commands in Text to Model). The MHz figures are built from platform measurements.

- VIT WWE and VCE figures on RT1060:

Table 2. 1 MIC solution

MHz		Code	Data memory		
Peak	Avg	45 kB	ROM model storage	RAM persistent	RAM scratch
240	156		325 kB	275 kB	47 kB

- VIT WWE and VCE figures on RT600:

Table 3. 1 MIC solution

MHz		Code	Data memory	
Peak	Avg	35 kB	RAM model storage	RAM
65	36		353 kB	190 kB

- VIT WWE and VCE figures on RT500:

Table 4. 1 MIC solution

MHz		Code	Data memory	
Peak	Avg	32 kB	RAM model storage	RAM
84	46		325 kB	167 kB

- VIT WWE and VCE figures on LPC55S69:

Table 5. 1 MIC solution

MHz		Code	Data memory	
Peak	Avg	36 kB	ROM model storage	RAM
92	83		325 kB	167 kB

- VIT WWE and VCE figures on Cortex-A53:

Table 6. 1 MIC solution

MHz		Code	Data memory	
Peak	Avg	40 kB	RAM model storage	RAM
60	50		325 kB	256 kB

VIT WWE and VCE stack usage < 2 kB

6 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN

CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7 Revision history

Table 6 summarizes the changes done to this document since the initial release.

Revision history

Revision number	Release date	Description
5	18 October 2023	Update profiling section
4	31 July 2023	VIT_SetModelUpdate description is added. Section 5 section is updated. Language updates. The name of the document is changed from Voice Intelligent Technology Integration User's Guide to Voice Intelligent Technology Wake Word and Voice Command Integration User's Guide
3	13 January 2023	VIT AFE description is removed, 30-ms input frame support is added.
2	10 October 2022	Updated for the next version
1	19 May 2022	Updated the VIT profiling and platform support list corresponding to VIT in SDK2.11.
0	10 September 2021	Initial release

8 Appendix

For the details about examples presented in MCUXpresso SDK including steps for creation custom wake words or voice commands with VIT online tool, see Getting Started with VIT for i.MX RT Devices.

The example shown in Figure 3 and Figure 4 illustrates the voice command research window: end of voice command utterance shall occur in a ~3 s window from the wake word. (for more information on the window size controlled via `Command_Time_Span`, see section 4.2.1.4)

Example 1:

The voice command utterance is ending 1.7 s after the wake word: Once the wake word is detected, VIT WWE and VCE switches to the voice command research mode. It detects the voice command and switches back to the wake word detection mode.

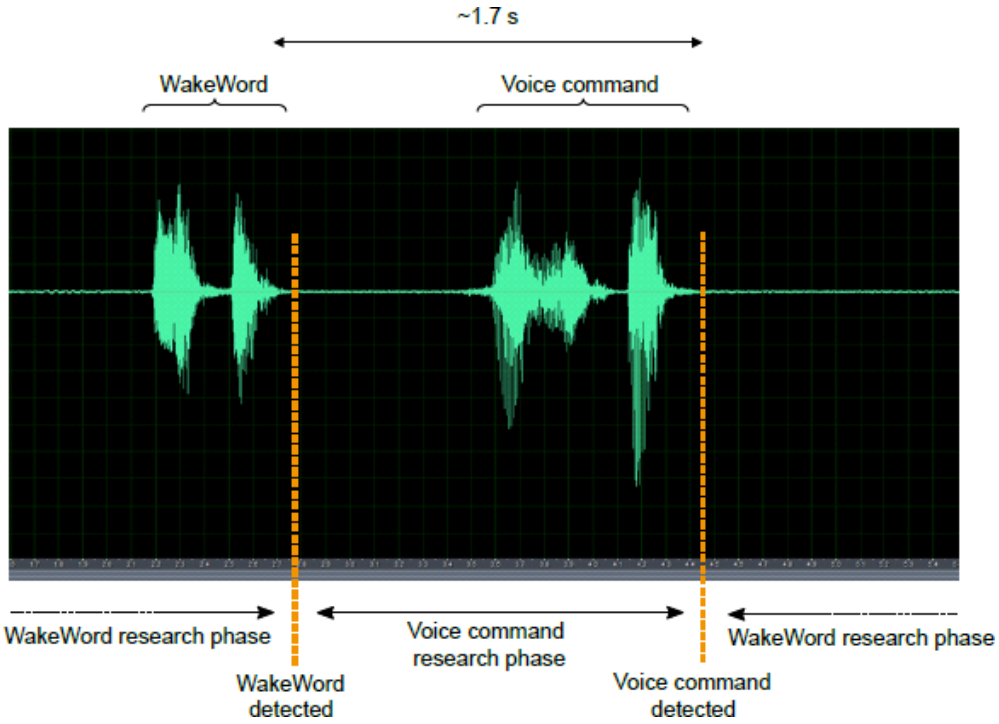

Figure 3. Example 1

Example 2:

The voice command utterance is ending 3 s after the wake word: Once the wake word is detected, library switches to the voice command research mode. Library would not be able to detect the voice command, since the command is not fitting in the 3 s window. (for more information on the window size, see section 4.2.1.4)

At the end of the 3 s research window, VIT returns an “UNKNOWN” command and switch back to the wake word detection mode.

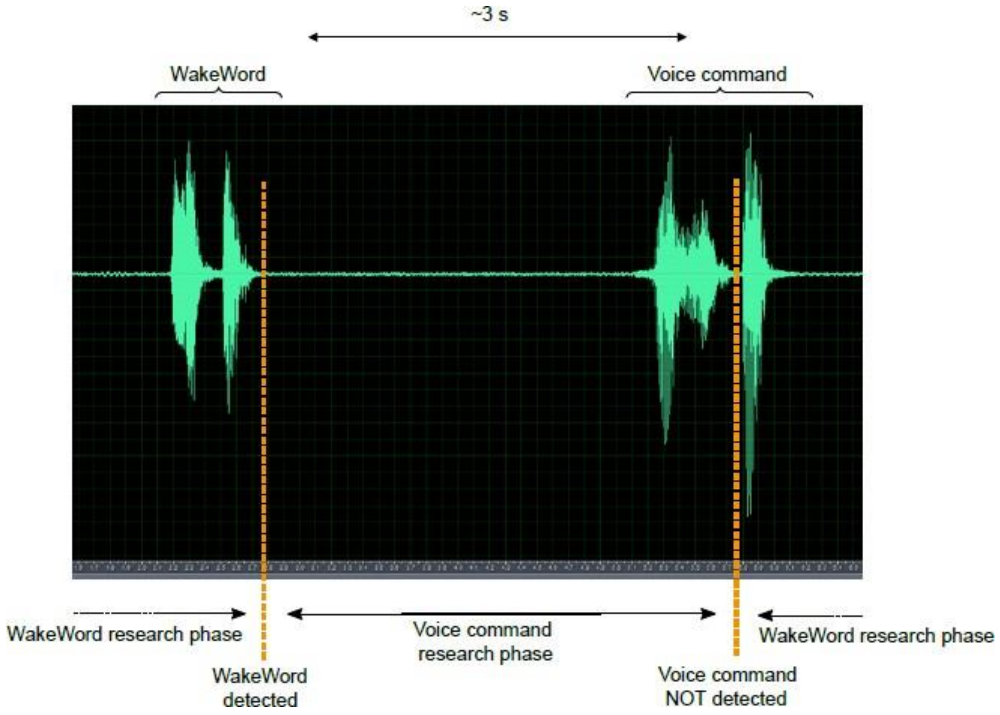

Figure 4. Example 2

9 Legal information

9.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

9.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

9.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

Date of release: 31 July 2023
Document identifier: VITIUG