VMX pi

Vision/Motion Processor & Robotics Controller

Hardware Abstraction Layer (HAL) Library Reference Guide

Version 1.4

Contents

Overview

The VMX-pi HAL is an Application Programming Interface (API) providing robot applications access to all VMX-pi capabilities.

The VMX-pi HAL is implemented as set of shared libraries for C++, Java, C# and Python.

C++ applications must be compiled with the GCC C++ compiler in order to use the VMX-pi HAL.

Java applications should be compiled with the Java 1.8 compiler in order to use the VMX-pi HAL.

C# applications should be compiled with the Mono C# compiler in order to use the VMX-pi HAL.

NOTE: The standard Raspbian Stretch image includes all compilers by default except for the Mono C# compiler. This can be installed via the following linux command-line commands:

```
sudo apt-get -y install mono-runtime
```

Installation Location

/usr/local/lib/vmxpi

This directory contains the installed shared libraries for use by C++, Java, C# and Python Applications, as follows:

Language	Library Files(s)
C++	libvmxpi_hal_cpp.so
Java	libvmxpi_hal_java.so, vmxpi_hal_java.jar
C#	libvmxpi_hal_csharp.so, vmxpi_hal_csharp.dll
Python	_vmxpi_hal_python.so, vmxpi_hal_python.py

/usr/local/include/vmxpi

This directory contains the installed header (.h) files for use by C++ applications.

/usr/local/src/vmxpi/hal_cpp_examples

This directory contains various sub-directories, one for each of the C++ VMX-pi HAL samples.

Each subdirectory contains a make file. “make” to build the sample, “make run” to run the sample.

/usr/local/src/vmxpi/hal_java_examples

This directory contains various sub-directories, one for each of the Java VMX-pi HAL samples. Each subdirectory contains a make file. “make” to build the sample, “make run” to run the sample.

/usr/local/src/vmxpi/hal_csharp_examples

This directory contains various sub-directories, one for each of the C# VMX-pi HAL samples. Each subdirectory contains a make file. “make” to build the sample, “make run” to run the sample.

/usr/local/src/vmxpi/hal_python_examples

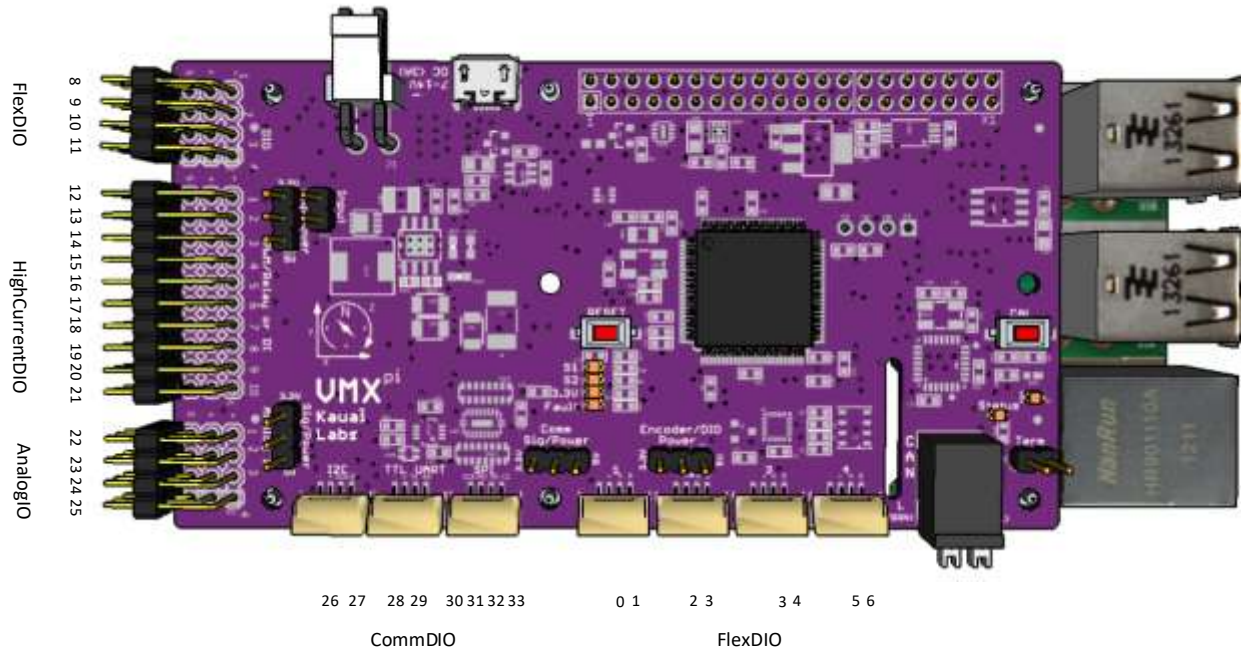
This directory contains various sub-directories, one for each of the Python VMX-pi HAL samples. Each subdirectory contains a make file. "make" to build the sample, "make run" to run the sample.

Installing/Updating the VMX-pi HAL

The VMX-pi HAL can be easily installed (or updated) via the Kauai Labs apt server.

Instructions for installing/updating the HAL libraries is at: https://www.kauailabs.com/apt/

IO Resources and Channels



Introduction to IO Resources

VMX-pi IO Resources are electronic circuits (or software algorithms) that implement a well-defined function on an electrical signal and have one or more ports.

Resource Type	Function	Number of Ports
Digital Input	Detects high and low levels	1 Digital Input Port
Digital Output	Sends high and levels	1 Digital Output Port
PWM Generator	Sends alternating high and low levels at a frequency and duty cycle	1 (HighCurrent DIO or CommDIO) or 1 or 2 (FlexDIO) Digital Output Ports
PWM Capture	Measures the frequency and duty cycle of a signal from an external PWM generator	1 (FlexDIO) Input Port
Quadrature Encoder decoder	Measure the number of encoder pulses in either 1x, 2x or 4x mode	2 (FlexDIO) Input Ports

Interrupt	Generates an asynchronous notification when a transition of a signal level (e.g., from low to high) occurs	1 Digital Input Port
Analog Accumulator	Samples and Accumulates an Analog Input signal level	1 Analog Input Port
Analog Trigger	Generates an Interrupt, using the output from an Analog Accumulator as an input	1 (an Analog Accumulator) Input Port
I2C	Implements the Inter-Integrated Circuit (i2C) digital communications protocol	1 Digital Output Port, 1 Digital Open Drain Port
UART	Implements the Universal Asynchronous Receiver/Transmitter (UART) digital communications protocol	1 Digital Input Port, 1 Digital Output Port
SPI	Implements the Serial Peripheral Interface (SPI) digital communications protocol	3 Digital Ouptut Ports, 1 Digital Input Ports

IO Resource Ports

Each Resource has one or more Ports, each of which can have one VMX-pi channel routed to it.

Some VMX-pi resources support multiple ports.

Single-channel IO Resource Routing

Many Resources provide a function on a single Channel, and thus only a single Channel is routed to the Resource.

Multi-channel IO Resource Routing

Certain Resources provide a function on multiple Channels, and thus may have multiple Channels routed to the Resource.

Multi-channel function IO Resources

Resources providing “multi-channel functions” require that VMX-pi Channels be routed to all Ports in order to execute the Resource function (e.g., UART Resources require a channel routed to the “TX” function and a different channel routed to the “RX” function).

Multi-instance IO Resources

Other “multi-instance” Resources can perform multiple instances of the same function, and therefore do not necessarily require Channels to be routed to all Ports in order to operate (e.g, FlexDIO PWM Generator resources can output independent PWM signals onto 2 channels).

Shared FlexDIO Timer Resource Limits

Certain Resources used with FlexDIO Channels (PWM Generation, PWM Capture and Quadrature Encoder Decode) are implemented by Hardware Timers; each timer may be routed to only 2 specific FlexDIO Channels.

	FlexDIO Ch0/1	FlexDIO Ch2/3	FlexDIO Ch4/5	FlexDIO Ch6/7	FlexDIO Ch8/9	FlexDIO Ch 10/11
PWM Generation	0 and/or 1	2 and/or 3	4 and/or 5	6 and/or 7	8 and/or 9	10 and/or 11
Quad Encoder Decode	0 and 1	2 and 3	4 and 5	6 and 7	8 and 9	10 and 11
PWM Capture	0 or 1	2 or 3	4 or 5	6 or 7	8 or 9	10 or 11

Because FlexDIO Timer Resources are shared, it is not possible for a single pair of adjacent FlexDIO Channels to be used with a different resource. For example, it is not possible for FlexDIO Channel 0 to be used for PWM output and FlexDIO Channel 1 to be used for PWM Capture.

Introduction to IO Channels

Table 1 VMX-pi IO Channels and IO Resources

Resource	Flex DIO	HighCurrent DIO	Analog Inputs	Comm DIO	Maximum Count
Channel #s	All Channels	12-21	22-25	26-33	34
Digital Output	All Channels	When “Output” jumper enabled – all channels controlled by one jumper	--	Channels 26-28, 30-31, 33	28
Digital Input	All Channels	When “Output” Jumper disabled – all channels controlled by one jumper	--	Channels 29, 32	24
PWM Generation	All Channels	When “Output” jumper enabled – all channels controlled by one jumper	--	All Digital Output channels	28
HW Quad Encoder decode	Channels 0-9	--	--	--	5
PWM Capture	All Channels (up to 6 total)	--	--	--	6
I2C	--	--	--	Channels 26-27	1

UART	--	--	--	Channels 28-29	1
SPI	--	--	--	Channels 30-33	1
Analog Accumulation	--	--	All Channels	--	4
Analog Trigger	--	--	All Channels	--	4

Flex DIO Channels (channels 0-11)

Each Flex DIO Channel may be a Digital Input or Digital Output, depending upon software configuration, and may also be used to decode Quadrature Encoder signals, generate PWM signals, capture PWM signals, or generate Interrupts.

HighCurrent DIO Channels (channels 12-21)

High Current DIO Channels may either all be in “input mode” or “output mode”, depending upon the HighCurrent DIO “Output” jumper.

When configured in “input mode”, each and every HighCurrent DIO Channels can be used as Digital Inputs or Interrupt Inputs.

When configured in “output mode”, each and every HighCurrent DIO Channels can be used as Digital Outputs or for PWM Generation.

Analog Input Channels (channels 22-25)

Analog Input Channels are used with Analog-to-Digital Converters (ADCs) to measure the analog voltage level at a 12-bit resolution (4096 discrete values). Each input is managed by an Analog Accumulator, which implements both oversampling and averaging.

Each Analog Input may also be routed to an Analog Trigger resource in order to generate an Interrupt.

Comm DIO Channels (channels 26-33)

Comm DIO Channels are used for digital communications using several different protocols.

Certain Comm DIO Channels are dedicated as outputs, and other Comm DIO Channels are dedicated as inputs. When not used for protocol communication, these channels can be used as digital inputs and outputs.

Output Comm DIO Channels behave similarly to HighCurrent DIO Channels in “output” mode.

Input Comm DIO Channels behave similarly to High Current DIO Channels in “input” mode.

Allocating and Configuring IO Resources

Although different VMX-pi IO Channels can be configured for use with several IO Resources, each IO Channel may only be allocated to one IO Resource at a time.

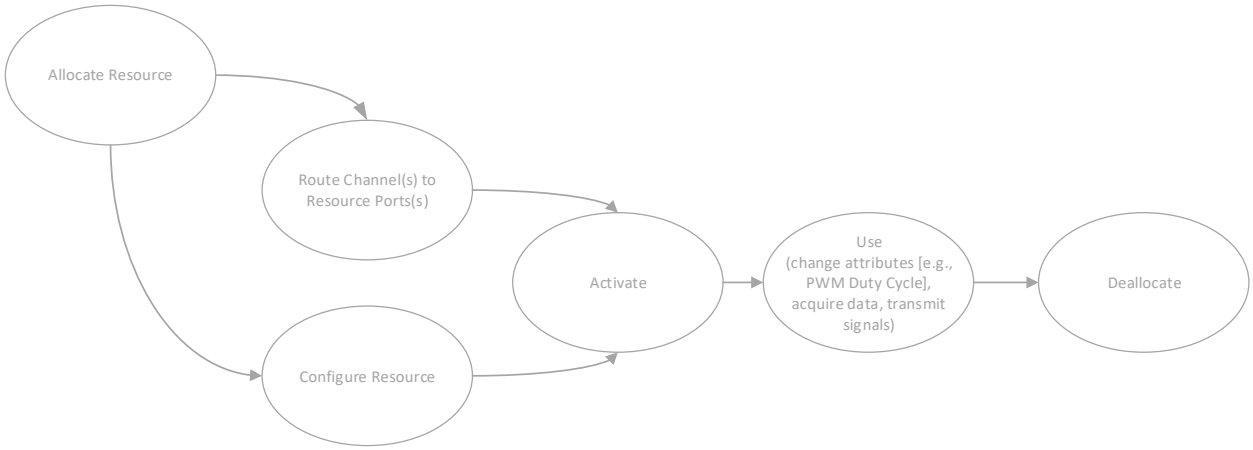


Figure 1 VMX-pi Resource Allocation/Deallocation Flow

The VMX-pi HAL VMXIO class provides methods for easily allocating, routing, configuring and activating channels and resources.

Using Resources

Once activated, the VMXIO class provides methods - which are specific to each VMX-pi resource type – for interacting with an activated resource.

Relevant Source Code Examples

Many of the VMX-pi examples demonstrate how to use these methods:

IO Resource type	Example
Analog Accumulator	analog_inputs
Digital Input	digital_inputs
Digital Output	digital_outputs
Interrupt	Interrupts
PWM Generator	pwm_generation
Quadrature Encoders	encoders
PWM Capture	pwm_capture
I2C Communication	i2c
SPI Communication	spi
UART Communication	uart

Inertial Measurement Unit (IMU) /Attitude & Heading Reference System (AHRS)

Access to the VMX-pi IMU and AHRS circuitry occurs via the VMX-pi HAL AHRS class. The AHRS class automatically buffers the latest data allowing access to the data at any time. Additionally, applications may register a callback for immediate notification when new data has been received.

Relevant Source Code Examples

The “imu” VMX-pi HAL example demonstrates both polled and notification-based access to IMU/AHRS data.

CAN Bus

Access to the VMX-pi CAN Bus Interface occurs via the VMX-pi HAL VMXCAN class.

Relevant Source Code Examples

The “can_bus_monitor” and “can_tx_loopback” VMX-pi HAL examples demonstrate CAN configuration, data reception and data transmission.

Receive Streams

Multiple different receive “streams” can be configured for receiving CAN messages. Each stream has an Acceptance Mask, an Acceptance Filter, and a buffer for temporarily storing CAN messages until the HAL application is ready to consume them.

CAN IDs

Each data sender on the CAN bus must send a unique ID, often referred to as a “message ID”. There are two types of CAN IDs: Standard and Extended.

Standard IDs

Standard IDs are 11 bits in length, and thus have a range (in hexadecimal) of 0x0 - 0x7FF.

Extended IDs

Extended IDs are 29 bits in length, and thus have a range (in hexadecimal) of 0x0 – 0x1FFFFFFF.

To limit the messages received to only those of interest and to avoid needless processing overhead to manage them, the use of “Acceptance Masks” and “Acceptance Filters” is recommended. When used together, Acceptance Masks and Filters can identify ranges of CAN message IDs of interest.

Acceptance Masks

Each “1” bit in the Mask corresponds to a bit position in a CAN message ID

NOTE: An Acceptance Mask of 0 will allow all messages to be received. This is not recommended due to the potentially very high volume of messages which may be received.

Standard ID Acceptance Masks

When creating an Acceptance Mask for receiving Standard IDs, a “Standard Frame Bit” (0x40000000) must be included, as shown below:

```
// This masks in all 11 standard ID bits

uint32_t standardIdMask = 0x000007FF | VMXCAN_IS_FRAME_11BIT;
```

Extended ID Acceptance Mask

When creating an Acceptance Mask for receiving Extended IDs, the lowest 29 bits of the mask are used, as shown below:

```
// This masks in all 29 extended ID bits

uint32_t extendedIdMask = 0x1FFFFFFF;
```

Acceptance Filters

Acceptance Filters are used together with Acceptance Masks to receive only those CAN IDs of interest.

A CAN Message is received if all bits masked in by an Acceptance Mask match the corresponding Acceptance Filter bit value. The following Truth Table indicates each possible case.

Filter/Mask Truth Table

Mask Bit n	Filter Bit n	Message Bit n	Accept or Reject Bit n
0	X	X	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

A simplified set of rules is this:

- If mask bit is 0: that bit is always accepted.
- If mask bit is 1: that bit is only accepted if it's value equals that of the filter bit

Standard ID Acceptance Filters

When creating an Acceptance Filters for receiving Standard IDs, a "Standard Frame Bit" must be included, as shown below:

```
// This filter will only accept Standard IDs, since the VMXCAN_IS_FRAME_11BIT bit (0x40000000) is set.

uint32_t standardIdFilter = 0x00000710 | VMXCAN_IS_FRAME_11BIT;
```

Extended ID Acceptance Filters

When creating an Acceptance Filter for receiving Extended IDs, the lowest 29 bits of the filter are used, and the "standard id bit" (0x40000000) is not set, as shown below:

```
// This filter will only accept Extended IDs, since the VMXCAN_IS_FRAME_11BIT bit is not set.

uint32_t extendedIdFilter = 0x11092AD6;
```

Acceptance Mask/Filter Examples

This section contains several examples demonstrating how to receiving CAN Messages from any Cross the Road Electronics (CTRE) Power Distribution Panel (PDP).

The PDP regularly transmits messages indicating the amount of current flowing through each output, as well as the input battery voltage level. As documented in the PDP.cpp file (in the wpilib suite on Github), the transmitted messages are as follows:

Message	Message ID	Description
STATUS1	0x08041400	Current levels of PDP outputs 1-6 (10 bits/output)

STATUS2	0x08041440	Current levels of PDP outputs 7-12 (10 bits/output)
STATUS	0x08041480	Current levels of PDP output 13-16 (10 bits/output) and battery voltage
STATUS_ENERGY	0x08041740	

The source code in PDP.cpp includes the definitions of the data in each message. To convert each 10bit current level value to amps, multiply the value by 0.125

NOTE: In the case of the PDP and other CAN devices on a FRC CAN bus, the lowest 6 bits of each message ID are reserved as a device ID (which is unique to each type of device). This allows multiple devices of the same type to exist on a single CAN bus.

Mask & Filter Example 1: receive only the STATUS1 message from PDP device id 3:

Acceptance Mask	Acceptance Filter
0xFFFFFFFF	0x08041403

Mask & Filter Example 2: receive only the STATUS1 message from any PDP device id (0-63)

In this example, CAN message IDs 0x08041400-0x0804143F are accepted.

Therefore, the lowest 6 bits may be any value; any higher-order bits must be 0x08041400.

Acceptance Mask	Acceptance Filter
0xFFFFFFFC	0x08041400

Mask & Filter Example 3: receive the STATUS1, STATUS2, STATUS3 and STATUS_ENERGY messages from any PDP device id:

In this example, CAN message IDs 0x08041400-0x0804177F are accepted.

Acceptance Mask	Acceptance Filter
0xFFFFF800	0x08041000