WOODWARD

MotoHawk Training

# CAN

## Controller Area Network

**(28 October 2009)**

# Notice





# OUTLINE

- CAN Introduction
- CAN Physical Network
- CAN Message Format
- Using the MotoHawk CAN Blocks
- MotoHawk “Post Office”
- MotoHawk Advanced CAN (Message) Blocks
- CAN Protocol Overview
- CAN Challenge Exercise



# INTRODUCTION

- CAN = Controller Area Network
  - Communication specification implemented for automotive applications in the 1980s

- Often, the term “CAN” is misused
  - CAN is a hardware definition for interoperability between modules
  - CAN specification does *not state the data content* of a given message
  - Protocols built on top of CAN *state the data content* (ex. J1939, GMLAN)

- MotoHawk doesn’t *define* a protocol, but allows access to the CAN hardware to *implement* any protocol

**=== By itself, CAN is *not* difficult (I mean it) ===**



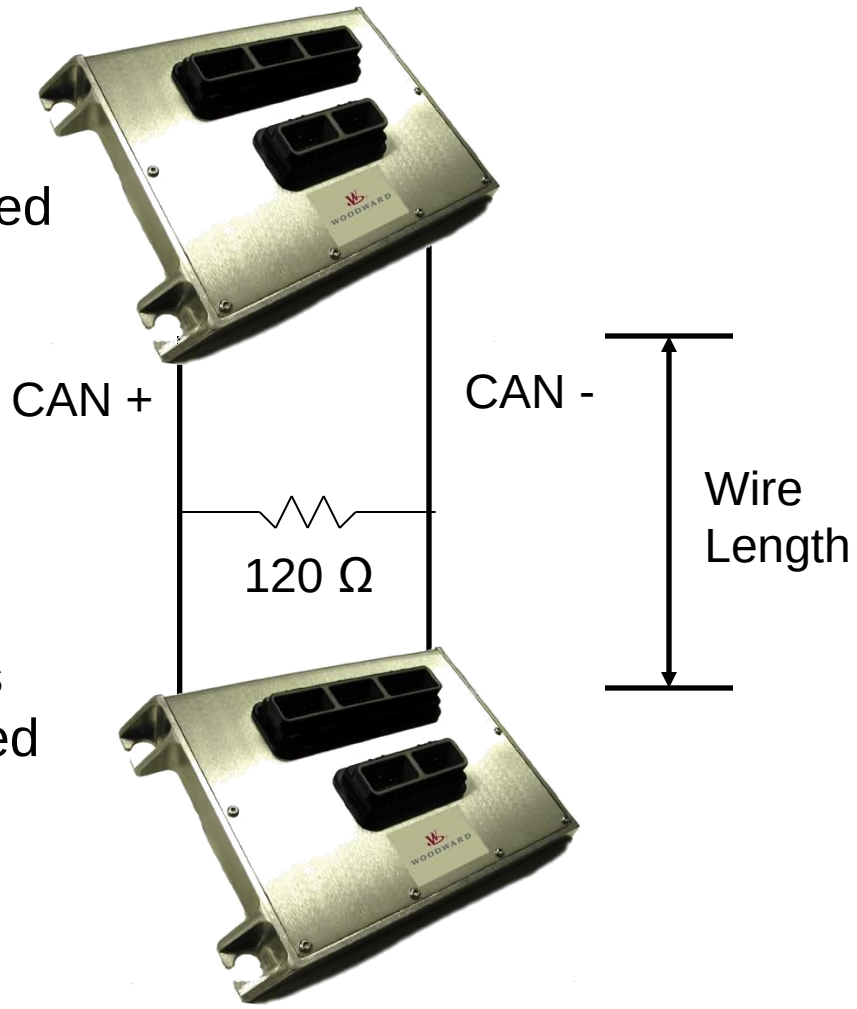
# OUTLINE

- CAN Introduction
- CAN Physical Network
- CAN Message Format
- Using the MotoHawk CAN Blocks
- MotoHawk “Post Office”
- MotoHawk Advanced CAN (Message) Blocks
- CAN Protocol Overview
- CAN Challenge Exercise



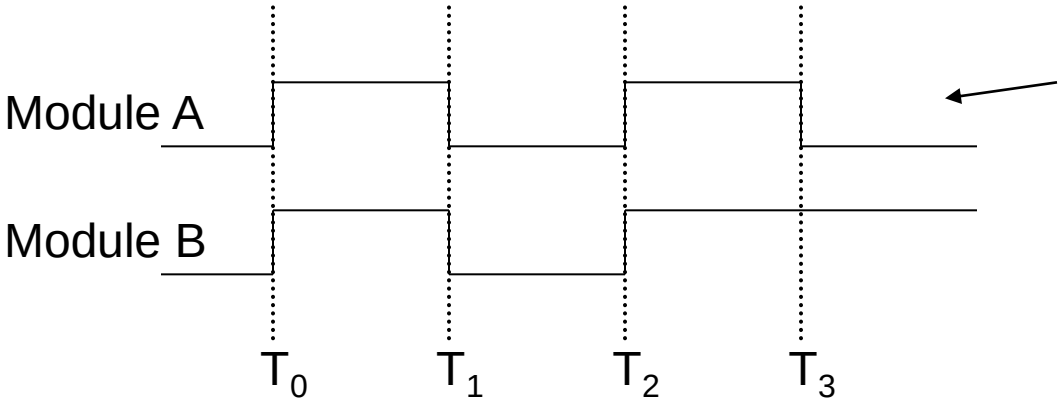
# PHYSICAL NETWORK

- Two wire robust serial communication
- Minimum two talker-listeners on bus
- Up to 1.0Mbps data rate which is limited by wire length and design
  - 100 m @ 250kbps
  - 30 m @ 1Mbps
- Termination resistance required
- Maximum bandwidth
  - 2000 messages per second @250kbps
  - 4000 messages per second @500kbps
  - 8000 messages per second @1.0Mbps
- Designed bandwidth should not exceed 70% of the maximum bandwidth for robust arbitration



# PHYSICAL NETWORK

- Bus arbitration is handled with a simple strategy
  - All modules on the bus attempt to transmit a message at the same time
  - The message with the lowest address wins
- This strategy is handled at the hardware level
  - Dominant Bits (0) vs. Recessive Bits (1)
- Multiple modules on the same bus cannot transmit the same address at the same time – Arbitration fails – Bus crashes




Module A (sending 0) continues to transmit while Module B (sending 1, sees 0 which does not match its 1) and thus it stops to wait for the next opportunity to transmit

# OUTLINE

- CAN Introduction
- CAN Physical Network
- CAN Message Format
- Using the MotoHawk CAN Blocks
- MotoHawk “Post Office”
- MotoHawk Advanced CAN (Message) Blocks
- CAN Protocol Overview
- CAN Challenge Exercise



# MESSAGE FORMAT

- A CAN 2.0B Message can contain up to 131 bits

[Overhead] [Address (11 or 29)] [Overhead] [Length] [Payload (0 to 64)] [Overhead]

- As application developers, 3 fields are important:
  - 4 bits determine the length of the data (aka payload) (Range: 0 to 8 Bytes)
  - Up to 64 bits of data depending on data length
  - ID Format
    - Extended IDs are 29 bits
    - Standard IDs are 11 bits
    - Extended and Standard IDs can exist on the same bus at the same time
    - Standard IDs have less message overhead (higher percentage of data per message)



# CAN in MotoHawk

OK, now that we know how CAN works…

How do we make it happen in MotoHawk?



# OUTLINE

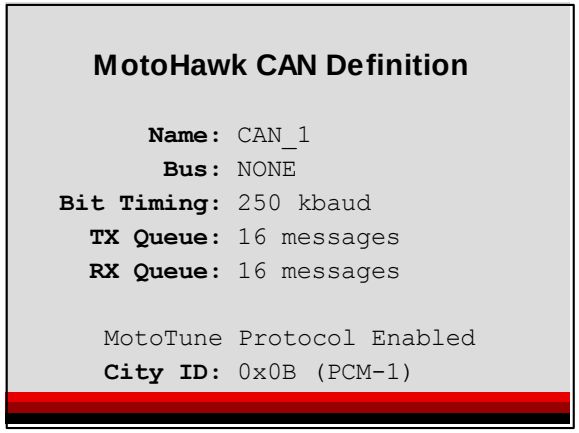
- CAN Introduction
- CAN Physical Network
- CAN Message Format
- Using the MotoHawk CAN Blocks
- MotoHawk “Post Office”
- MotoHawk Advanced CAN (Message) Blocks
- CAN Protocol Overview
- CAN Challenge Exercise



# CAN DEFINITION BLOCK

The CAN hardware needs to be initialized

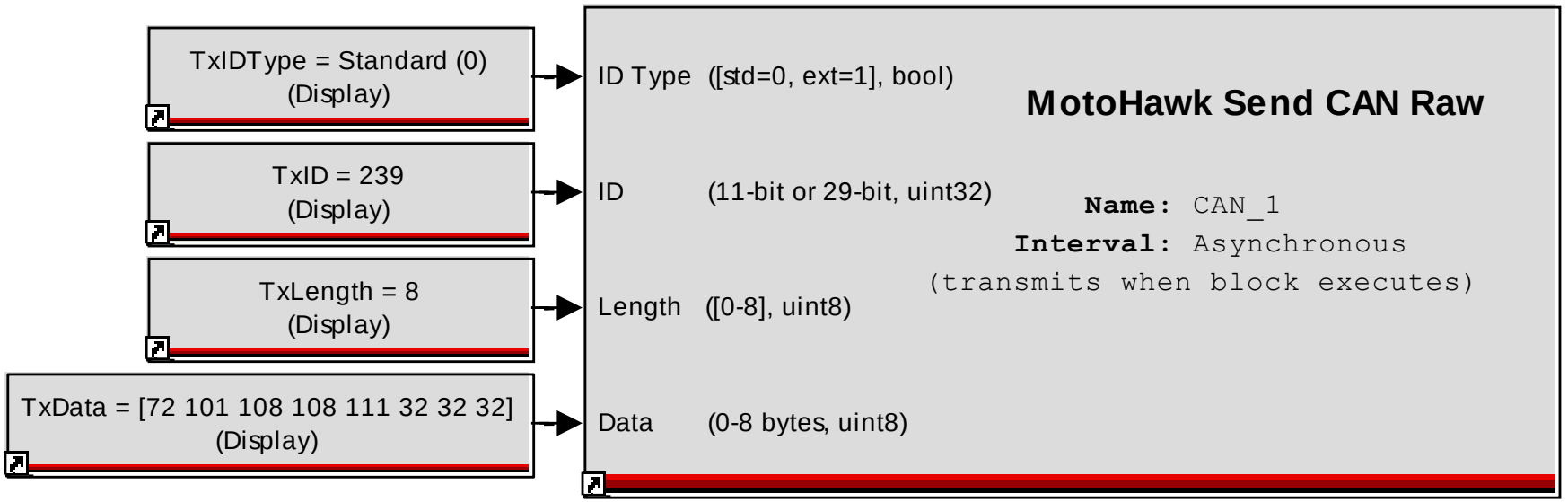
- Configure the incoming and outgoing queue sizes
- Set the baud rate
- Set the City-ID
- Select or remove MotoTune protocol




- If the block settings are not configured properly, then MotoTune can't talk to the module via CAN and therefore programming cannot be performed via CAN
- Recovery is made with a boot key or boot cable
  - Recovery sets CAN baud rate to 250kbps
  - Recovery sets City-ID to 0x0B

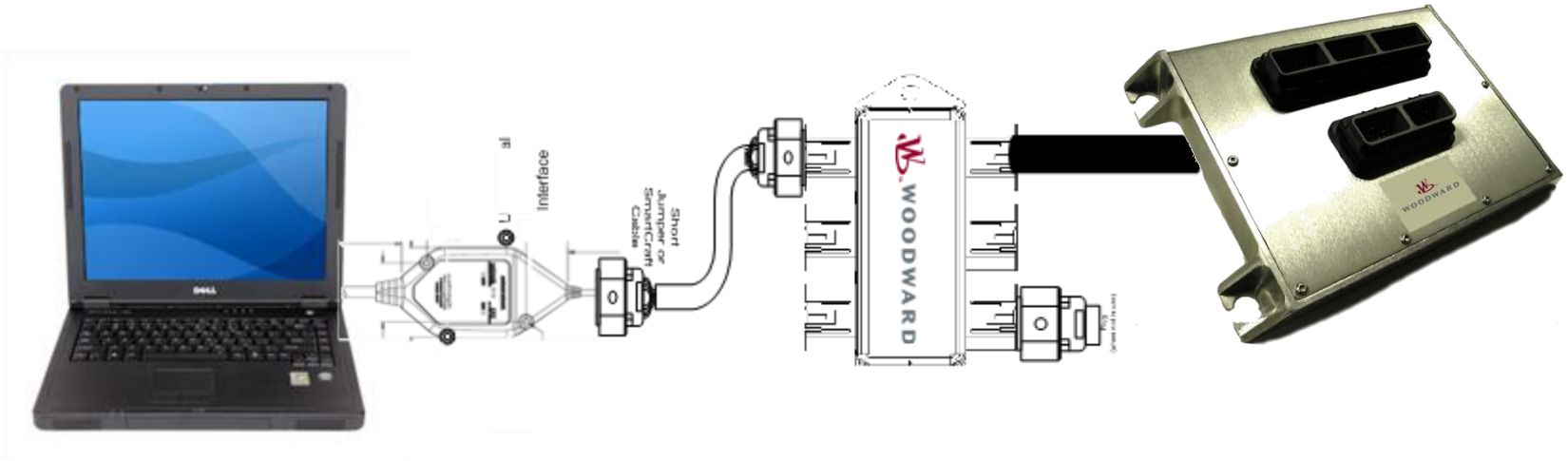
# Send CAN Raw Block

Simple send message block with ASCII text “Hello ”



# CAN Exercise #1

- **CAN Bus Test**
  - **Construct a calibratable CAN message transmitter**
  - **Use CANKing to monitor bus traffic**




System Diagram

# OUTLINE

- CAN Introduction
- CAN Physical Network
- CAN Message Format
- Using the MotoHawk CAN Blocks
- MotoHawk “Post Office”
- MotoHawk Advanced CAN (Message) Blocks
- CAN Protocol Overview
- CAN Challenge Exercise



# POST OFFICE - Description

- CAN is the hardware layer, so how are transmitted messages sorted and filtered in MotoHawk?
- A Post Office offers the simplest analogy
  - Mailboxes
  - Letters
  - Name, House #, Street, City, State, Zip
  - Copy Machine
  - Doorbell

# POST OFFICE - Sorting

Many messages are transmitted on a bus in turn, but a module may only be interested in a small subset requiring sorting

Sorting (filtering) is done at two distinct layers:

- In hardware using the specific buffers on your target module's CAN chips
- In software using a custom dispatcher for your messages

This is similar to a post office where the CAN messages are letters and the hardware/software dispatcher built by MotoHawk is the postman



# POST OFFICE - Sorting

MotoHawk abstracts this to a single interface with Identifiers (CAN addresses) and Masks (which parts to care about)

- A “mail box” in MotoHawk is called a ‘Slot’
- This mail ‘Slot’ has a CAN address call an ID (Identifier)

  (29 bits long or 11 bits long, both are supported)
- The ‘ID’ uses a ‘Mask’ to tell the MotoHawk postman which parts of the address to care about and which parts to ignore

  Care about bits 28 through 8, but not 7 through 0 of a 29-bit address. This equates to care about the house #, street, city state, zip, but not the name on the envelop, for instance.



# POST OFFICE - Sorting

MotoHawk sorting can be done in two places:

- ID Filtering: Message are sorted by the ID (address) using the ID-Mask (tells which bits of the ID to care about)

  ----  This works for 11-bit and 29 bit addresses  ----
- Payload Filtering: Message may be sorted by the data content using the Data (as additional address space) and a Data-Mask (to tell which bits of the data to use as address space)

# POST OFFICE - Sorting

Filtering is best understood in binary math

- The mask contains a “1” in any bit location that will be used in the sort and a “0” in any location where the address data may vary
- Proceed bit by bit to build the slot

An 11-bit example is shown:

- ID = 0x7E4 (111 1110 0100) what it should be
- ID mask = 0x7FC (111 1111 1100) 1 = care, 0 = don’t care
- the resulting in slot looks for (111 1110 01XX) X = either 0 or 1 allowed

- If incoming ID = 0x7E5, (111 1110 0101)
  - ✔ message goes into the mailbox
- If incoming ID = 0x7F4, (111 1111 0100)
  - x message is rejected by the mailbox
- If incoming ID = 0x7E7, (111 1110 0111)
  - ✔ message goes into the mailbox



# POST OFFICE - Sorting

Like with the ID and Mask, sorting can be done with a Payload-Value and Payload-Mask as well

- Payload Value = 0x(01, 00, …., 00, F6)
  (0000 0001, 0000 0000, …. 0000 0000, 1111 0110)
- Payload Mask = 0x(7F, 00, ...., 00, FE)
  (0111 1111, 0000 0000, …. 0000 0000, 1111 1110)
- the resulting payload slot filter looks for
  (X000 0001, XXXX XXXX, …. XXXX XXXX, 1111 011X)

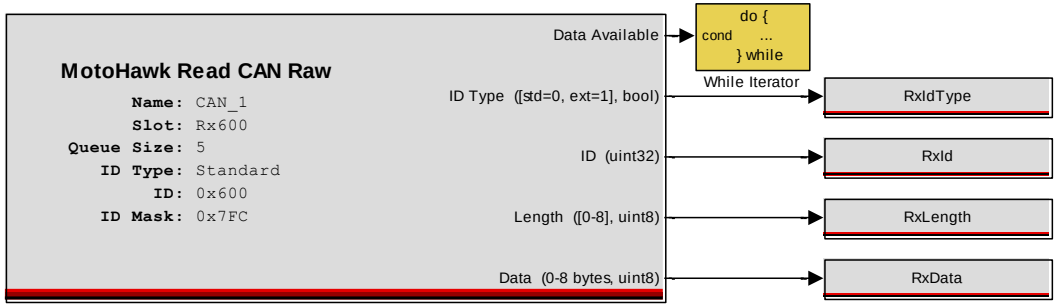
Thus:

- If incoming ID gets through, and has the payload
- = 0x(81, 12, …. 3C, F7)
  (1000 0001, 0001 0010, …. 0011 1100, 1111 0111)
  - ✔ message goes into the mailbox

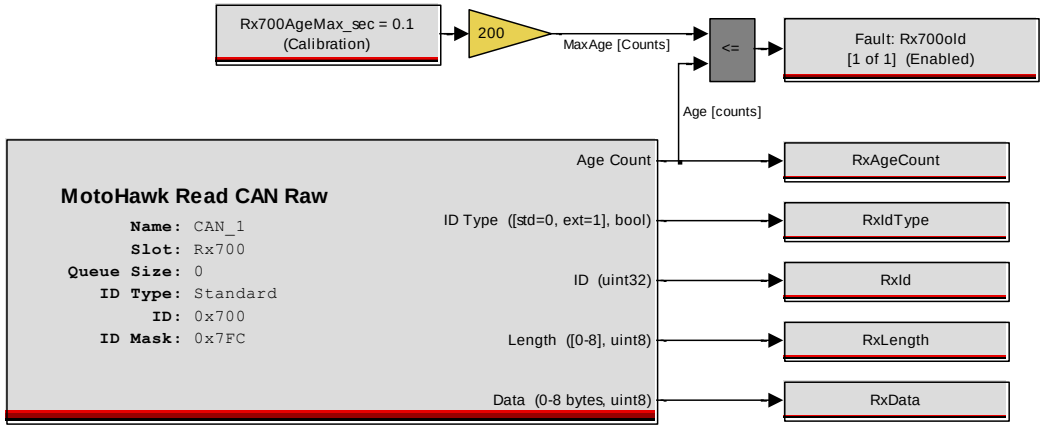


# POST OFFICE - Read CAN Raw BLOCK

Faster message implementation empties queued messages




Slower message implementation with “stale” data check

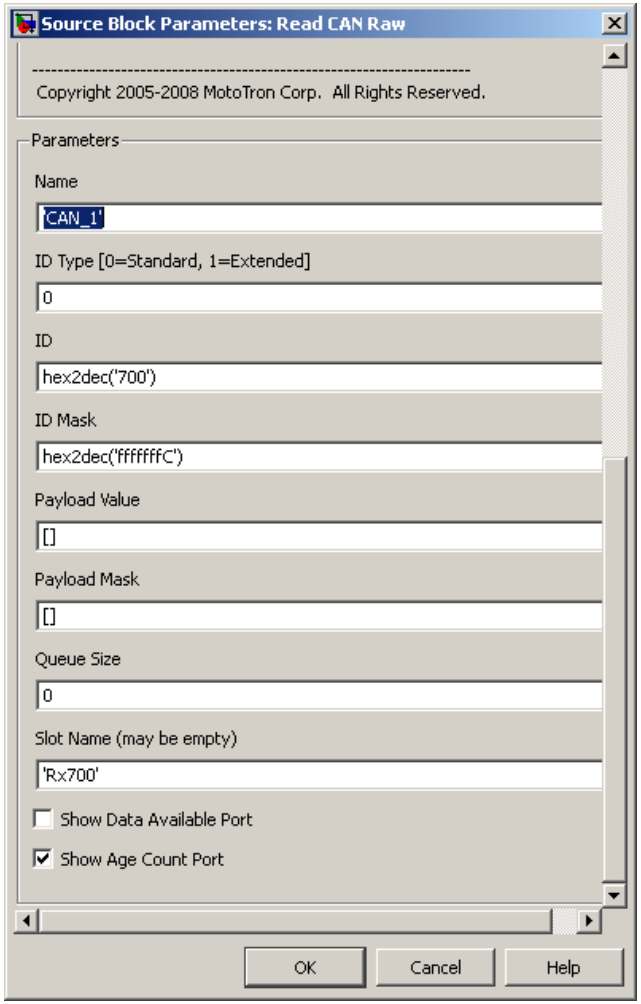





WOODWARD

# POST OFFICE – Configure Mail Box

- Name – Logical CAN bus name from definition blocks
- ID Type 11-bits or 29-bits
- Settings for the ID, ID Mask, Payload (address) Value, and Payload Mask explained earlier
- Queue size - how many messages to store (0 or 1 = 1)
- Slot Name used with ‘Slot Properties’ block to change mail box parameters at Run-Time
- Data Available and Age Count adds ports as needed

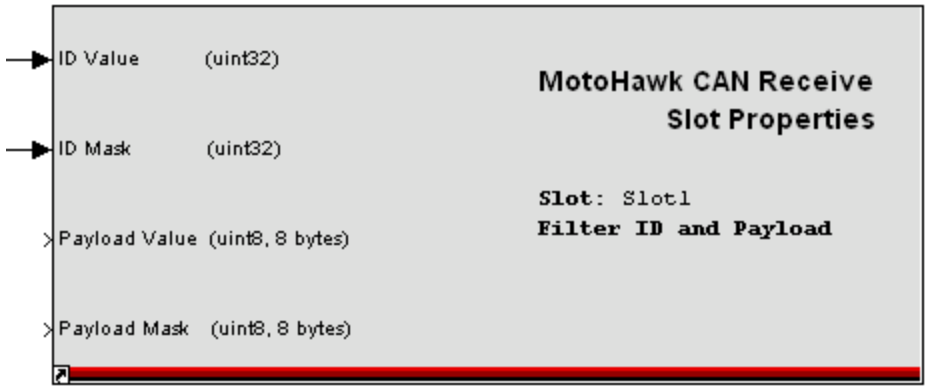


# POST OFFICE

- The properties of a mailbox can be changed at run time using the ‘MotoHawk CAN Receive Slot Properties’ block
- The filters determined at design time in the MotoHawk CAN receive blocks can be strengthened (made more restrictive), but cannot be weakened (made less restrictive)
- Modifications to the sorting filter may be done on ID or payload data fields.
- A CAN receive block and its slot properties block are linked by the ‘Slot’ name

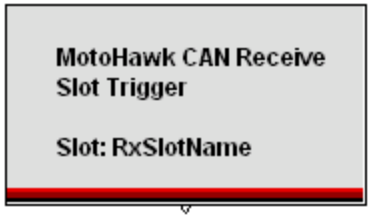
ID Filter Controls

Payload Filter Controls



# POST OFFICE

- We can have the postman ring your doorbell when he delivers a message?
  - May be used to record time of arrival
  - May be used for high operation priority code based on reception
  - May be used to synchronize software between modules over CAN
  - May be used to quickly reply to an asynchronous message without periodically polling for the message




- How does MotoHawk implement this situation?
  - CAN Receive Slot Trigger Block function calls a sub-system when its  CAN message is received.
  - ‘Slot’ name in this block must match the slot name specified in its corresponding ‘Read CAN Raw’ block or ‘Read CAN Message’ block



# OUTLINE

- CAN Introduction
- CAN Physical Network
- CAN Message Format
- Using the MotoHawk CAN Blocks
- MotoHawk “Post Office”
- MotoHawk Advanced CAN (Message) Blocks
- CAN Protocol Overview
- CAN Challenge Exercise



# MOTOHAWK ADVANCED CAN

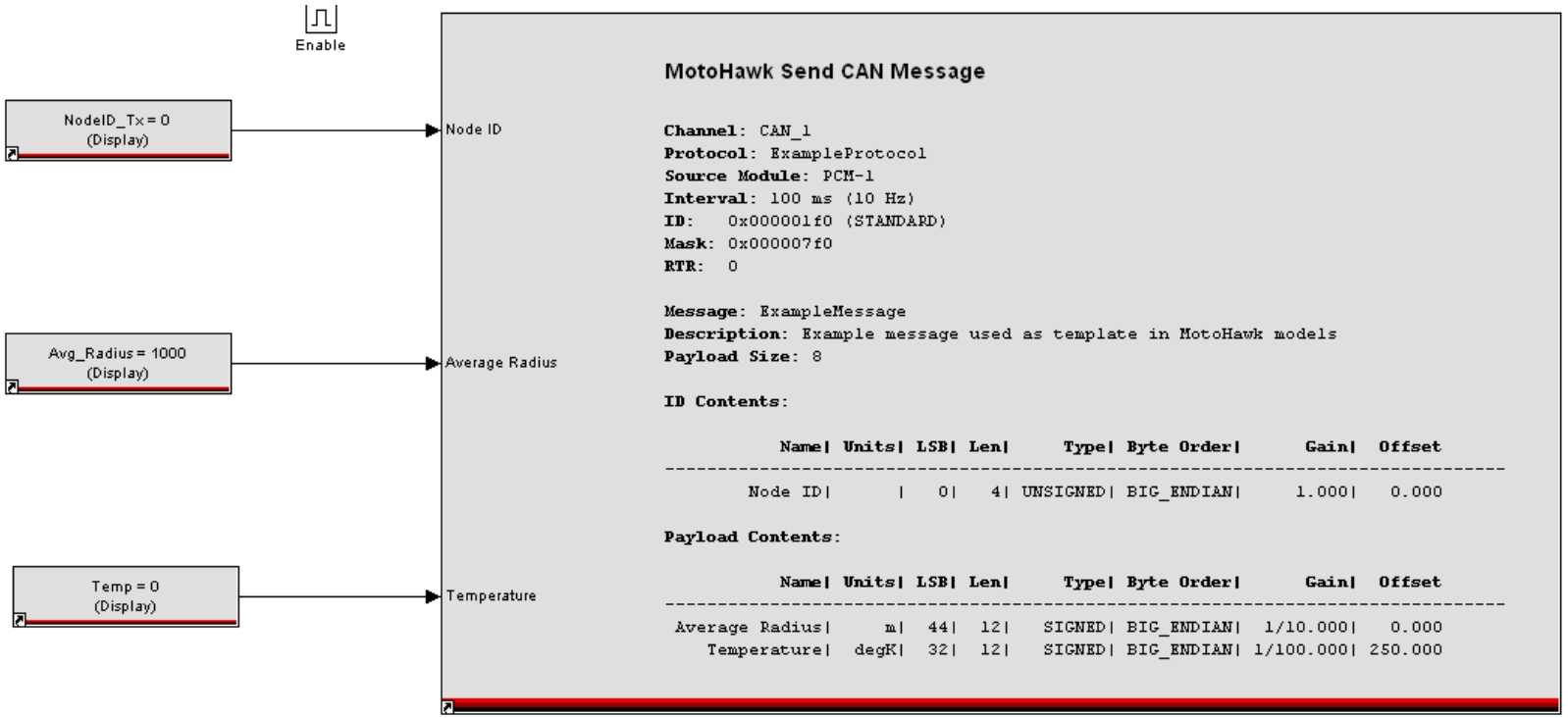
- The Read and Send CAN blocks are nice, but sometimes more advanced data parsing is necessary. Common questions:
  - I have 12 bit scaled data that spans across multiple bytes. How do I convert it into engineering units?
  - I have more data that can fit into 64 bits. How do I create multi-page messages?
  - I'm using a protocol that has a variable ID. How do I dynamically create the ID easily?

- These are valid questions and there is an answer…

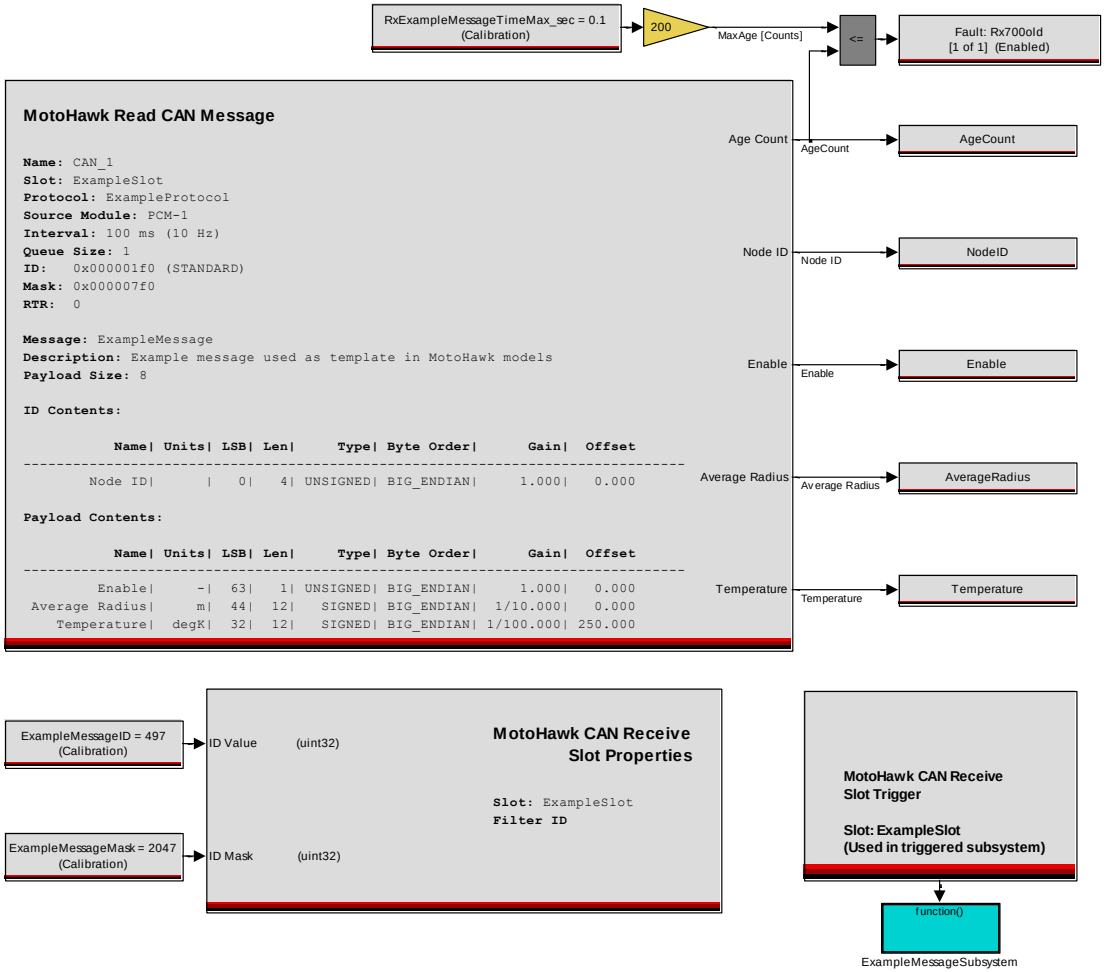


# MOTOHAWK ADVANCED CAN

- MotoHawk has 2 blocks – Read CAN Message and Send CAN Message (below)
- These blocks allow users to set up multi-page documents and parse and scale both changing IDs an variables



# MOTOHAWK ADVANCED CAN




## Motohawk Read Can Message block

- Uses same definition file as transmit
- Works with Slot Properties block
- Works with Slot Trigger block
- Age-Count port optional
- Data-Available port optional
- Name Wires optional



WOODWARD

# MOTOHAWK ADVANCED CAN

Both MotoHawk advanced CAN blocks are specified using one type of message definition file

We have included an example of this file in MotoHawk as well as another in the CAN file folder in your MotoHawk Project

From the command line, type:

```
>> edit motohawk_can_example.m
```



# LITTLE ENDIAN vs BIG ENDIAN

- "Endianness" refers to the order in which bytes are stored in memory (Intel vs. Motorola…..LSB vs. MSB first in memory)
- Terminology originates from "Gulliver's Travels" from the war over "from which end to eat a hard-boiled egg"
- This example shows the 4 byte storage for 1025 (0x401)

| Address | Big Endian Representation | Little Endian Representation |
|---|---|---|
| 00 | 0000 0000 | 0000 0001 |
| 01 | 0000 0100 | 0000 0000 |
| 02 | 0000 0000 | 0000 0100 |
| 03 | 0000 0001 | 0000 0000 |

- By default, MotoHawk CAN scripts use 'Big-Endian' byte ordering, but do 'Little-Endian' just as easily
- Unpack must occur in the same order as the bytes are packed

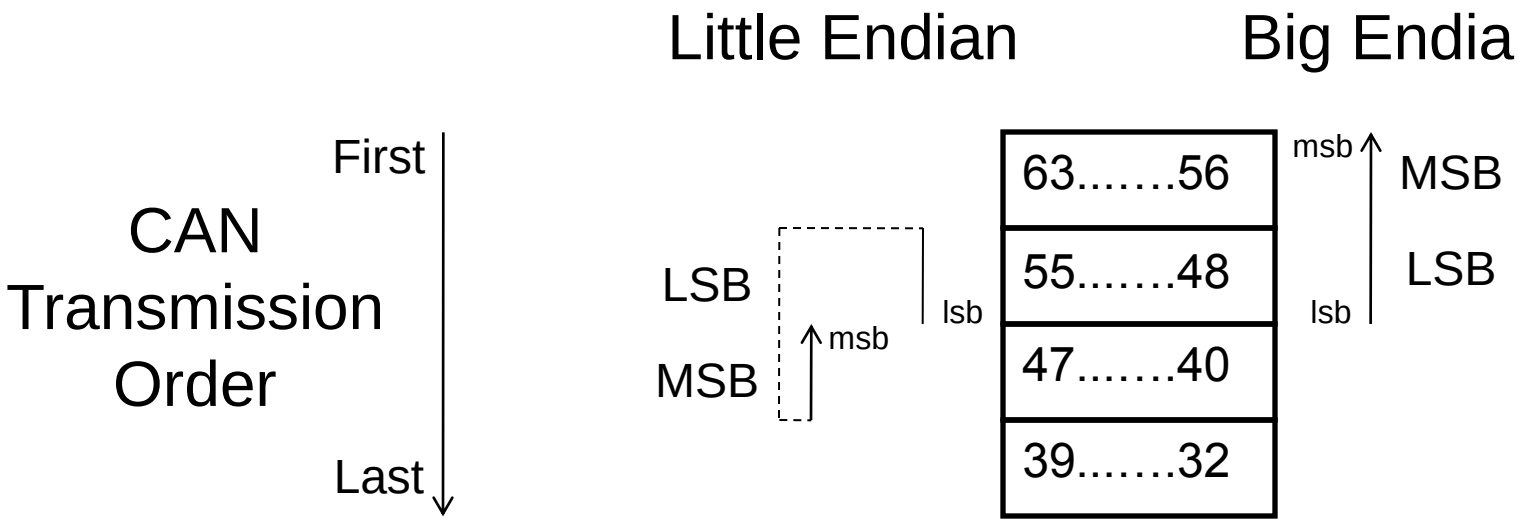


# LITTLE ENDIAN vs BIG ENDIAN

Our example shows byte ordering for a 2 byte variable (uint16)

- Both start with the least significant bit of the least significant byte at bit 48
- Both fill from least significant bit toward most significant bit
- Both orders “wrap” at byte boundaries only:
  - Big endian wraps upward into the lsb of the preceding byte (ahead)
  - Little endian wraps downward into the lsb of the byte below (to follow)

We care only so that the unpack matches whatever pack was chosen



# OUTLINE

- CAN Introduction
- CAN Physical Network
- CAN Message Format
- Using the MotoHawk CAN Blocks
- MotoHawk “Post Office”
- MotoHawk Advanced CAN (Message) Blocks
- CAN Protocol Overview
- CAN Challenge Exercise



# PROTOCOL OVERVIEW

A protocol is NOT CAN!  It is a language on CAN.

- CAN defines our post office and mail slots
- Protocols define the content of the letters, the adress structure, ect.  They are like French or English or German used to tell a story.  Our post office will deliver “letters” written in any of them.
  - Does MotoHawk speak J1939?
  - Does MotoHawk do GM LAN?
  - Does MotoHawk implement NMEA-2000?
  - Does MotoHawk support Mercury SmartCraft

Answer: Yes! but only in so far as the application does.



# PROTOCOL OVERVIEW

- CAN only gets complicated when protocols are considered
  - J1939, SmartCraft, CCP, GMLAN, etc. are all examples of protocols that define and adhere to strict rules about the message address and contents
  - Protocols like J1939 use PGNs to define messages
- Simulink and/or Stateflow in your application, used with the MotoHawk CAN blocks, is/are able to implement virtually any CAN based protocol
- Some message formats (J1939 for example) have already been implemented in CAN message definition files and are available



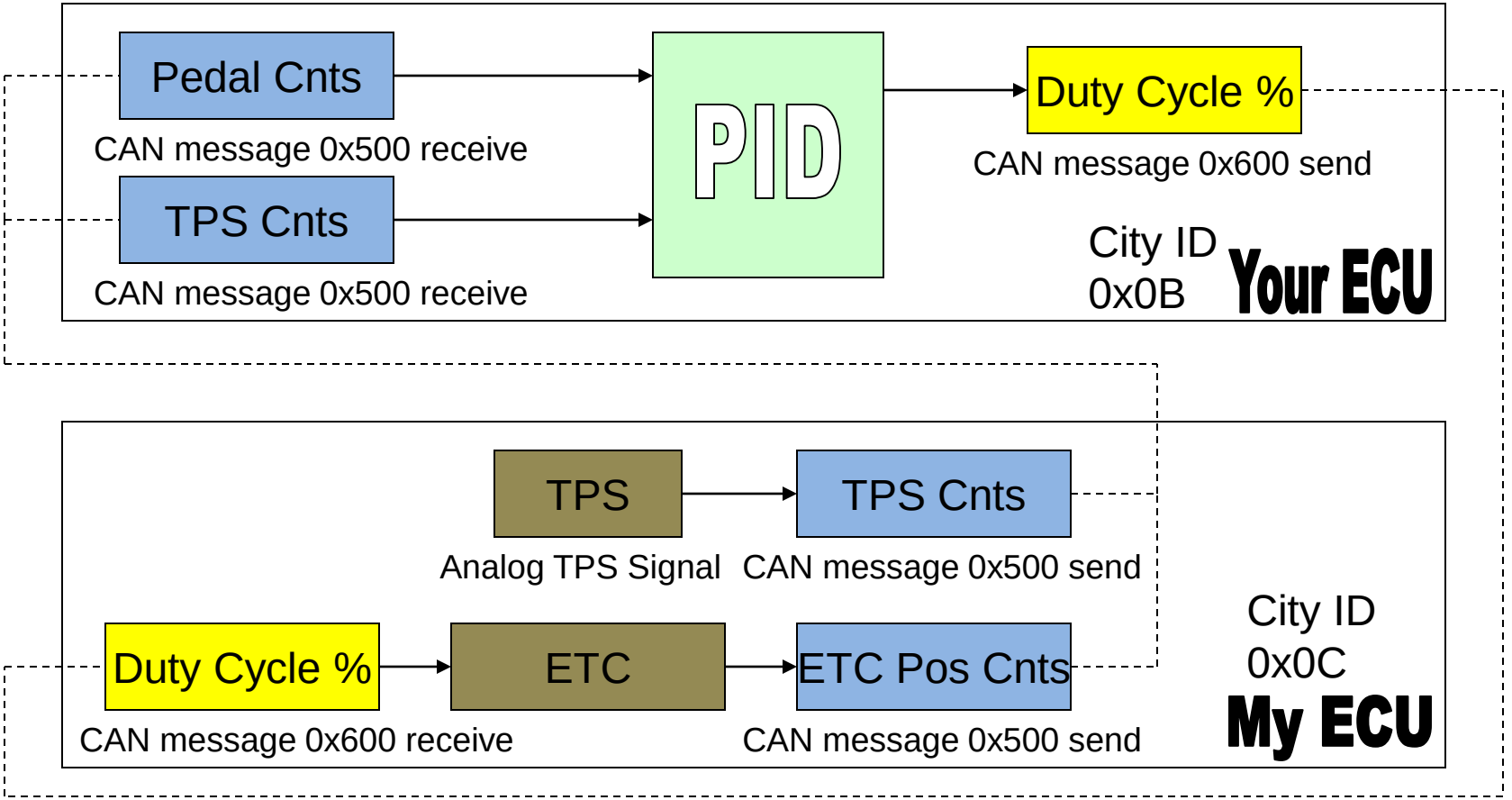
# OUTLINE

- CAN Introduction
- CAN Physical Network
- CAN Message Format
- Using the MotoHawk CAN Blocks
- MotoHawk “Post Office”
- MotoHawk Advanced CAN (Message) Blocks
- CAN Protocol Overview
- CAN Challenge Exercise



# CLASS CAN CHALLENGE

- **Create a distributed control system to control the ETC over CAN**



# CLASS CAN CHALLENGE

- **Control ETC Remotely via CAN. You will Receive Slider Position and ETC Position via a message and send PWM duty Cycle via a message.**

- **RX Message (from instructor case):**
  - **ID: 0x500 (Std)**
  - **Rate: 10mSec Payload Size: 5**
  - **Content:**
    - **Slider Position, unsigned 10 bits, start bit 40, little endian, ADC counts**
    - **TPS Position, unsigned 10 bits, start bit 38, little endian, ADC counts**

- **Tx Message (to instructor case):**
  - **ID: 0x600 (Std)**
  - **Rate: 10mSec Payload Size: 3**
  - **Content:**
    - **PWM Duty Cycle, signed 16 bits, start bit 48, big endian, PWM %, 1/256 %/cnt**
    - **PWM Enable, boolean 1 bit, start bit 40, big endian, 1=Enable PWM**

    - **(Note, % goes from -100 to 100 in engineering units)**



WOODWARD

# MotoTron Control Solutions

## Production Controls in a Flash