Odyssey Offload Processing User Reference

Mpression Odyssey IoT Solutions Kits

For Odyssey FW v2.0

Index

1. Introduction and Overview……………………………………………………………….3

- Some things users can do with Odyssey v2.0
- Features
- Firmware & Smartphone App Requirements
- Command Structure and Command Entry
- Mixing Standard Commands and Offload Commands

2. Offload Processing Limitations and Special Enhancements……………….6

3. Command Opcodes and Descriptions Reference……………………………….8

3.1 Accumulator / Variables / User RAM

3.2 Arithmetic

- Addition
- Subtraction
- Multiplication (add/subtract options)
- Division (add/subtract options)

3.3 Bit / Logical / Mask / Byte swap Operations

3.4 Miscellaneous Control

- Delays
- LEDs: duration, flashing…

3.5 Program Control

- State machine reset
- Branches / Compares

3.6 Input / Output

- I2C Reads, Writes, multiple
- To/From Accumulator, User RAM

3.7 Test & Diagnostics

- Run custom commands from Smartphone

Appendix A: Considerations using Math………………………………………………22

Appendix B: New Personalities & Example Entry…………………………………23

Appendix C: Personality creation template…………………………………………26

Appendix D: Odyssey I2C addresses & Byte Order………………………………30

Document Revision History…………………………………………………………………31

1. Introduction and Overview

Beginning with Odyssey release v2.0, the on-board EFM32 microcontroller can perform offload processing on Sensor data and user entered data. There are several other operations that involve timing, I2C communications, LED indication and general state machine control.

1.1 Some things users can do with Odyssey v2.0

- Customize how, when and what data is collected – and display it
- Collect data from sensors or FPGA in a loop and store on-board:
  - Temperature, light, proximity, FPGA acquired data including sound, voltage, etc
  - Timing between samples can be milliseconds, hours, days (programmable)
  - Scale or process data samples using math, shifting, masking
- Convert data to display multiple formats (i.e. Temperature in both Celsius and Fahrenheit)
- Build data averages or other statistics
- Give user indication of data or status using the BLE LED, flashing patterns
- On the smartphone app…
  - Use a single button to display all collected data (auto indexing)
  - View in decimal or hex format
- Use and Learn from new example personalities
  - Custom Command Test: Type in custom instructions directly to the App screen, run them and display results without building a personality first
  - Temperature collection personality: Collects temperature data in a timed loop, then allows display of the collected and converted values with single buttons (auto-indexed)

1.2 Features

- Arithmetic: addition, subtraction, multiplication, division, signed math
- Multiple sources & destinations for data: accumulators, user RAM, user input data
- Logic: masking, OR, AND, XOR
- Byte operations: byte ordering / swap, endian changes
- Delays, LED on/duration/flashing
- Branches, looping, compares (branch-on-compare)
- I2C reads/writes, looped IO to/from accumulators or user RAM with single commands, auto indexing of RAM
- Test commands

1.3 Resources available to the offload processing state machine

- Two accumulators: Accumulator A and B
- Variables/Counters: Counter1, Counter2, I2CdeviceAdrs, IOcounter
- User RAM: 256 indexed bytes, UserRAMindex (points to RAM index)
- Over 100 pre-defined opcodes
- Full state machine with branching (capable of processing loops)

1.4 Firmware and App requirements for Offload Processing

The Odyssey onboard processors (BLE and EFM32) must have matching firmware.

- BLE firmware v2.0
- EFM32 firmware v2.0
- Smartphone App v2.0: “Mpression Odyssey” from Google Play or Apple App Store

Check the Odyssey Software Download site for firmware updates, release notes, etc.

To determine what firmware your Odyssey has programmed, see the table below:

Firmware Version	At power-on, **green LED** on BLE board **flashes 3 times** in first seconds	**Version displayed** on the User Console display (within first 2 times hitting <enter> key)
1.0	No	None
1.1	Yes	None
2.0	Yes	2.0

1.5 Command Structure & Command Entry

Each opcode executes specific actions and most can use various sources or destinations for data (i.e. accumulator, user data, user RAM). Each offload processing command has the following information associated with it:

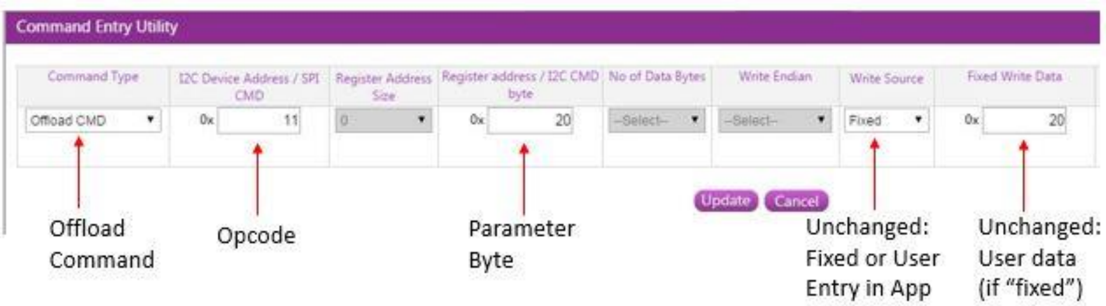
- Pre-defined opcode (1 byte) "OPCODE"
- Parameter byte "PARAM"
- User Data word (4 bytes) "USER_DATA"

NOTE: The Web Application GUI was not modified to show opcode names as this would be an extensive upgrade and not flexible. So Opcodes, Parameter bytes and User Data are all entered as hex data to the existing Web Application, after selecting the new Command Type: ***Offload_CMD.***
Users may also refer to the example in Appendix B for entering offload processing personality data.

Within the Odyssey Web Utility, users create "Offload Commands" using the following fields:
-*Offload CMD* (new drop-down selection) is selected in the "Command Type" Field
-*Opcode byte* values are placed in the "I2C Device Address/SPI CMD" field
-*Parameter Byte* values are placed in the "Register Address/ I2C CMD byte" field
-*User Data* values use the "Write Source" and "Fixed Write Data" fields
(or entered in entry field next to the button – according to "Write Source")
-Other fields will be grayed out

All offload processing is accomplished through the two new drop-down Command Types:

- **Offload CMD**
- **Offload Read**



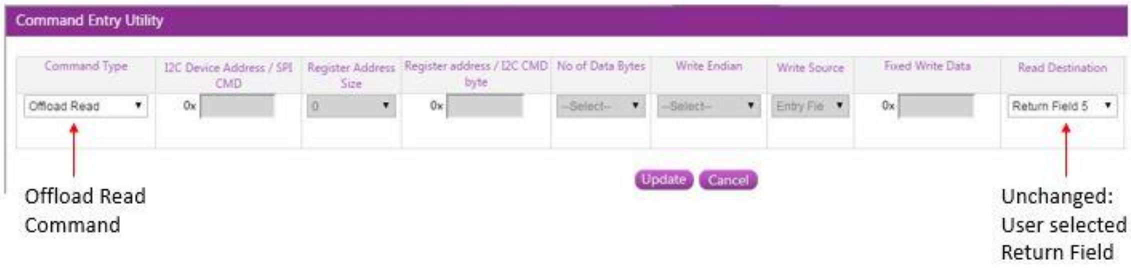
NOTE: If a write source is selected for user entry, the Field number must match the button number.

1.6 Offload Commands (Offload_CMD)

When Offload_CMD is selected as Command Type, the opcode, parameter byte and user data information is passed to the Offload processor (EFM32) by the Bluetooth processor (BLE).
Opcodes can load accumulators or operate on data in accumulators, user RAM or counters/indices/flags. They can also control I2C communications, LED or timer operations and allow branching (loops). Each opcode is associated with a routine in the EFM32 firmware that performs the operation.

1.7 Offload Reads (Offload_Read)

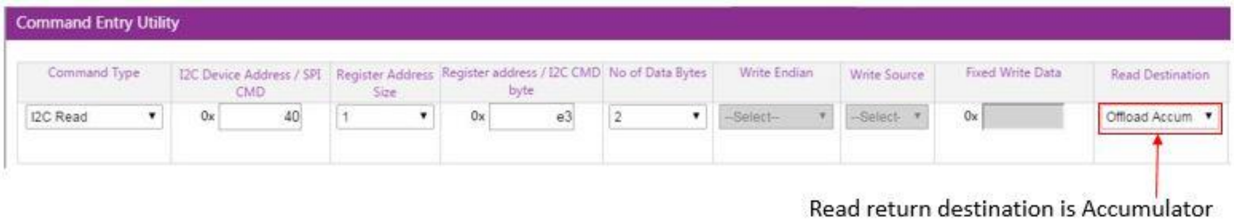
Eventually, the user will typically want to get data back to the smart phone app for display. For this, the new drop down Command Type ***Offload_Read*** command will read the resulting value from the accumulator for display in the smart phone App.



NOTE: The Return Field number (in Read Destination field) must match the button number.

1.8 Mixing Standard Commands with Offload Commands

Users may mix the original commands with Offload Commands (except for within loops). For example, a user can do an I2C Read (an original command) and set the data destination field to “Offload Accumulator”. Then the user can do Offload commands to process the data. Seen below is an example of an I2C Read to accumulator.



For more details on creating personalities, see section 3 in the *Mpression Odyssey User Guide*. Appendix B in this document will also show details for creating personalities with offload processing commands.

NOTES:

- The accumulators are NOT cleared between operations – to allow continuity between multiple processing commands. They are cleared to zero at power-on or using the reset opcode.
- Accumulator A (ACCUM_A) contents are returned via the Offload_Read command unless a special opcode is used to return Accumulator B.
- Invalid data (i.e. out of range) for opcode or parameter byte will generally result in NOP (no activity) but can result in undetermined behavior.

2. Offload Processing Limitations & Special Enhancements

2.1 Limitations

- **Error checking**

  The offload processing state machine does minimal range checking. If a user enters data that is out of range, the resulting action may be unknown. In most cases, the operation becomes a NOP (nothing happens) for that command. The user may also debug a new personality using extra Offload Read commands to see the Accumulator contents, or use the new "Custom Command Test" personality.

- **Loops**

  When users create loops with branching commands, a non-Offload command cannot be in the loop. This is because the BLE processor must receive original (non-offload) commands from Bluetooth directly for processing and cannot intervene in the state machine loop process.

- **Standard Command Count Limitations**

  Odyssey has always had the following limitations for the number of commands allowed per personality:
  - Setup Commands: 15 commands
  - Button Commands: 5 commands per button
  - Background commands: 5 commands (execute every 1 second when enabled)

  Users may wish to use the "Setup Commands" area to allow for more commands, for example, when creating a loop for collecting data. See the new personality "Temp Collection" details in Appendix B.

- **Signed Arithmetic**

  As with any compiler, users must take care and be clear what size their data is when working with <u>signed</u> math. For example, signed arithmetic must use operands the same size. This is due to the position of the sign bit (and 2's complement conversions). There are opcodes available for converting sizes to accommodate mixed size operands.

  See Appendix A: *Considerations using Math* for more information.

2.2 Special Enhancements

- **Data Retained between Buttons, Personalities**

  If users do not reset/clear accumulators or data, the information in those resources remain *unchanged* for the next button or personality – so the user can take advantage of more command slots. The following remain unchanged unless cleared:
  - Accumulators, user RAM, Counter1/Counter2, I2C_adrs_reg, User_ram_index, IOcounter

- **I2C looped Accesses and Auto-Indexing**

  I2C reads and writes can be done as single or multiple IO accesses with a single command. I2C sources and destinations:
  - Accumulator A and user RAM and user Data (command input) can be sources
  - Accumulator A and user RAM can be destinations

  Likewise, a single button can read out multiple RAM locations due to auto-indexing

3. Command Opcodes and Descriptions

The following is a list of the Opcodes by category with descriptions. For each opcode, any required Parameter Byte (PARAM) and User Data word (USER_DATA) settings are also described.
Anywhere a setting is marked "don't care", it is best set to 0 (zero).

Commands User Reference

In order of Opcode value 0x00 to 0xFF

- OPCODE **0x00: NOP**
  No activity
  PARAM: don't care USER_DATA: don't care

3.1 SIMPLE ACCUMULATOR, VARIABLES & USER_RAM

--- ACCUMULATOR

- OPCODE **0x01: Load ACCUMULATOR with USER DATA**
  PARAM: selects Accumulator to load: 0=ACCUM_A 1=ACCUM_B
  USER_DATA: data to be placed in selected Accumulator

- OPCODE **0x02: Load ACCUMULATOR with USER DATA (signed)**
  PARAM: selects Accumulator to load and signed data size:
  0=ACCUM_A, 32-bit signed data
  1=ACCUM_A, 16-bit signed data
  2=ACCUM_A, 8-bit signed data
  3=ACCUM_B, 32-bit signed data
  4=ACCUM_B, 16-bit signed data
  5=ACCUM_B, 8-bit signed data
  USER_DATA: data to be placed in selected Accumulator

- OPCODE **0x03: Copy ACCUMULATOR**
  PARAM: sets copy direction: Copy ACCUM_A to B or Copy ACCUM_B to A. Afterwards, both hold the same value.
  0=ACCUM_A -> ACCUM_B 1=ACCUM_B -> ACCUM_A
  USER_DATA: don't care

- OPCODE **0x04: SWAP ACCUMULATORS**
  ACCUM_A gets ACCUM_B's contents and ACCUM_B gets ACCUM_A's contents
  Accum_A <-> Accum_B
  PARAM: don't care USER_DATA: don't care

- OPCODE **0x05: Set ACCUM_B Read**
  ACCUM_A is the default source for Offload Read. This command sets a flag so the next Offload Read will read ACCUM_B instead (once). Afterward the default to ACCUM_A is re-set.
  PARAM: don't care USER_DATA: don't care

--- ACCUMULATOR – SIGNED SIZE & ABSOLUTE VALUE

- OPCODE **0x06: Convert signed data size in ACCUMULATOR**

Converts the size of a signed integer between 8, 16 and 32-bit sizes.
PARAM: sets the Accumulator and data size conversion:

0=NOP
1=Use ACCUM_A, convert 32-bit to 8-bit   0x11=Use ACCUM_B, convert 32-bit to 8-bit
2=Use ACCUM_A, convert 16-bit to 8-bit   0x12=Use ACCUM_B, convert 16-bit to 8-bit
3=Use ACCUM_A, convert 32-bit to 16-bit   0x13=Use ACCUM_B, convert 32-bit to 16-bit
4=Use ACCUM_A, convert 16-bit to 32-bit   0x14=Use ACCUM_B, convert 16-bit to 32-bit
5=Use ACCUM_A, convert 8-bit to 16-bit   0x15=Use ACCUM_B, convert 8-bit to 16-bit
6=Use ACCUM_A, convert 8-bit to 32-bit   0x16=Use ACCUM_B, convert 8-bit to 32-bit

USER_DATA: don't care

- OPCODE **0x07: Convert signed to int ACCUMULATOR**
  Converts a signed value (MS-bit set for negative number) to a signed integer (2's complement).
  User must select accumulator and data size.
  PARAM: sets the Accumulator and data size conversion:
  0=Use ACCUM_A, convert byte   0x10=Use ACCUM_B, convert byte
  1=Use ACCUM_A, convert 16-bit hword   0x11=Use ACCUM_B, convert 16-bit hword
  2=Use ACCUM_A, convert 32-bit word   0x12=Use ACCUM_B, convert 32-bit word
  USER_DATA: don't care

- OPCODE **0x08: Convert int to signed ACCUMULATOR**
  Converts a signed integer (2's complement) to a signed value (MS-bit set for negative number)
  User must select accumulator and data size.
  PARAM: sets the Accumulator and data size conversion:
  0=Use ACCUM_A, convert byte   0x10=Use ACCUM_B, convert byte
  1=Use ACCUM_A, convert 16-bit hword   0x11=Use ACCUM_B, convert 16-bit hword
  2=Use ACCUM_A, convert 32-bit word   0x12=Use ACCUM_B, convert 32-bit word
  USER_DATA: don't care

- OPCODE **0x09: Set ACCUMULATOR value to Absolute value**
  Assumes a signed integer is in Accumulator. Uses 2's complement if negative to make it positive signed integer.
  User must select accumulator and data size.
  PARAM: sets the Accumulator and data size:
  0=Use ACCUM_A, value is byte   0x10=Use ACCUM_B, value is byte
  1=Use ACCUM_A, value is 16-bit hword   0x11=Use ACCUM_B, value is 16-bit hword
  2=Use ACCUM_A, value is word (32-bits)   0x12=Use ACCUM_B, value is word (32-bits)
  USER_DATA: don't care

3.2 VARIABLES & USER_RAM

--- COUNTERS, REGS, RAM, FLAGS

- OPCODE **0x0A: Set/Update Variables**
  Set/Update Counters, User RAM Index
  PARAM: identifies counter/register/index and selects update action:
  0x0: IO_Counter set to User Data   0x9: RAM_Index set to user data
  0x1: IO_Counter decremented   0x0a: RAM_Index set to Accum_A data(LS byte)
  0x2: IO_Delay set to User Data (milliSecs)   0x0b: RAM_Index decremented by 1
  0x3: Counter1 set to User Data   0x0c: RAM_Index incremented by 1
  0x4: Counter1 decremented   0x0d: RAM_Index decremented by 4 (word)
  0x5: Counter1 incremented   0x0e: RAM_Index incremented (4 bytes, 1 word)
  0x6: Counter2 set to User Data

0x7: Counter2 decremented
0x8: Counter2 incremented

0x0f: I2C_Adrs_Reg set to User Data (LS byte)
0x10: I2C_byte_order set to User Data (0=MS Byte first [default], 1=LS Byte first)
0x11: Counters and Ram Index reset to 0 (USER_DATA ignored)
Includes: Counter1, Counter2, Ram Index
USER_DATA: value to set the counter/register (does not apply for increment/decrement)

- OPCODE **0x0B: Read USER_RAM to ACCUM**
  Reads from USER_RAM and loads read data into selected Accumulator
  PARAM: selects data size and accumulator
  0x0=Byte: USER_RAM to ACCUM_A (LS-Byte)
  0x1=Half word (16 bits): USER_RAM to ACCUM_A (LS 16-bits)
  0x2=Word (32-bits): USER_RAM to ACCUM_A
  0x10=Byte: USER_RAM to ACCUM_B (LS-Byte)
  0x11=Half word (16 bits): USER_RAM to ACCUM_B (LS 16-bits)
  0x12=Word (32-bits): USER_RAM to ACCUM_B
  USER_DATA -> USER_RAM Index to be read (index auto-increments to next byte)

- OPCODE **0x0C: Write byte to USER_RAM**
  Writes a byte to USER_RAM and then increments RAM Index
  PARAM holds the source of the write data byte:
  0=USER_DATA (LS-Byte)
  1=ACCUM_A (LS-Byte)
  2=ACCUM_B (LS-Byte)
  NOTE: RAM Index should be known or set PRIOR to this write command (reset default=0)

- OPCODE **0x0D: Write half word to USER_RAM**
  Writes a half word (16-bit value) to USER_RAM and then RAM index is incremented (+2)
  PARAM: holds the source of the write data (16-bits):
  0=USER_DATA LS-16-bits
  1=ACCUM_A LS-16-bits
  2=ACCUM_B LS-16-bits
  LS-byte is written first, then index incremented, then MS-byte
  NOTE: RAM_Index should be known or set PRIOR to this write command (reset default=0)

- OPCODE **0x0E: Write word to USER_RAM**
  Writes a word (32-bit value) to USER_RAM
  PARAM: holds the source of the write data word (32-bits)
  0=USER_DATA
  1=ACCUM_A
  2=ACCUM_B
  USER_DATA: holds 32-bit write data
  LS-byte is written first, incremented for each byte with last being MS-byte
  NOTE: RAM_INDEX should be known or set PRIOR to this write command (reset default=0)

- OPCODE **0x0F: Write / Fill USER_RAM**
  Writes a value or pattern of bytes to USER_RAM – all 256 bytes
  PARAM: holds the fill option:
  0=All zero’s
  1=ALL FF’s

2=Incrementing pattern - starting with byte value in USER_DATA (LS-byte)
3=Fill with byte value in USER_DATA (LS-byte)
USER_DATA: LS-byte holds initial byte/pattern byte (when PARAM=2 or 3)

3.3 ARITHMETIC

--- ADDITION

- OPCODE **0x11: Add (unsigned) USER_DATA to ACCUMULATOR**
  Adds Accumulator to User Data (unsigned data addition), then puts result in Accumulator.
  User selects which accumulator.
  PARAM sets Accumulator:
  0=ACCUM_A (default) 1=ACCUM_B
  USER_DATA holds data (unsigned) to be added to accumulator

- OPCODE **0x12: Add (signed) USER_DATA to ACCUMULATOR**
  Adds Accumulator to User Data (signed data addition), then puts result in Accumulator.
  User selects which accumulator and data size.
  PARAM sets which Accumulator and data size:
  0=ACCUM_A, Byte (LS-Byte, signed) 0x10=ACCUM_B, Byte (LS-Byte, signed)
  1=ACCUM_A, Halfword (LS-16-bits, signed) 0x11=ACCUM_B, Halfword (LS-16-bits, signed)
  2=ACCUM_A, Word (32-bits, signed) 0x12=ACCUM_B, Word (32-bits, signed)
  USER_DATA: don't care

- OPCODE **0x13: Add (unsigned) ACCUMULATORS**
  Adds the data in the two accumulators together: ACCUM_A + ACCUM B (unsigned data addition)
  PARAM sets Accumulator destination:
  0=ACCUM_A (default) 1=ACCUM_B
  USER_DATA: don't care

- OPCODE **0x14: Add (signed) ACCUMULATORS**
  Adds the data in the two accumulators together: ACCUM_A + ACCUM B (signed data addition)
  PARAM sets Accumulator destination and data size:
  0=ACCUM_A, Byte (LS-Byte, signed) 0x10=ACCUM_B, Byte (LS-Byte, signed)
  1=ACCUM_A, Halfword (LS-16-bits, signed) 0x11=ACCUM_B, Halfword (LS-16-bits, signed)
  2=ACCUM_A, Word (32-bits, signed) 0x12=ACCUM_B, Word (32-bits, signed)
  USER_DATA: don't care

--- SUBTRACTION

- OPCODE **0x15: Subtract (unsigned) USER_DATA From ACCUMULATOR**
  Subtracts User data from Accumulator (unsigned data subtraction):
  Accumulator (unsigned) - USER_DATA (unsigned) -> Accumulator (unsigned)
  PARAM sets which Accumulator to use:
  0=ACCUM_A (default) 1=ACCUM_B
  USER_DATA: don't care

- OPCODE **0x16: Subtract (signed) USER_DATA from ACCUMULATOR**
  Subtracts User data from Accumulator (signed data subtraction):
  Accumulator (signed) - USER_DATA (signed) -> Accum_A or B (signed)
  PARAM sets which Accumulator to use and data size:
  0=ACCUM_A, Byte (LS-Byte, signed) 0x10=ACCUM_B, Byte (LS-Byte, signed)

1=ACCUM_A, Halfword (LS-16-bits, signed) 0x11=ACCUM_B, Halfword (LS-16-bits, signed)
2=ACCUM_A, Word (32-bits, signed) 0x12=ACCUM_B, Word (32-bits, signed)
USER_DATA: holds data to be subtracted from accumulator

- OPCODE **0x17: Subtract (unsigned) ACCUMULATOR From USER_DATA**
  Subtracts Accumulator from User Data (unsigned data subtraction):
  USER_DATA (unsigned) - Accum_A (unsigned) -> Accum_A or B (unsigned)
  PARAM sets Accumulator destination:
  0=ACCUM_A (default) 1=ACCUM_B
  USER_DATA: holds unsigned data to subtract Accumulator from

- OPCODE **0x18: Subtract (signed) ACCUMULATOR from USER_DATA**
  USER_DATA (signed) - Accumulator (signed) -> Accum_A or B (signed)
  PARAM sets which Accumulator to use and data size:
  0=ACCUM_A, Byte (LS-Byte, signed) 0x10=ACCUM_B, Byte (LS-Byte, signed)
  1=ACCUM_A, Halfword (LS-16-bits, signed) 0x11=ACCUM_B, Halfword (LS-16-bits, signed)
  2=ACCUM_A, Word (32-bits, signed) 0x12=ACCUM_B, Word (32-bits, signed)
  USER_DATA: holds signed data to subtract Accumulator from

- OPCODE **0x19: Subtract (unsigned) ACCUMULATORS**
  Subtracts unsigned data in one accumulator from unsigned data in another
  PARAM:
  0=ACCUM_A - ACCUM_B (unsigned) -> Accum_A (unsigned)
  1=ACCUM_B - ACCUM_A (unsigned) -> Accum_B (unsigned)
  USER_DATA: don't care

- OPCODE **0x1A: Subtract (signed) ACCUMULATORS**
  Subtracts signed data in one accumulator from signed data in another
  PARAM sets Accumulator destination and data size:
  0x0: ACCUM_A – ACCUM_B –> ACCUM_A, Byte (LS-Byte, signed)
  0x1: ACCUM_A – ACCUM_B –> ACCUM_A,, Halfword (LS-16-bits, signed)
  0x2: ACCUM_A – ACCUM_B –> ACCUM_A,, Word (32-bits, signed)
  0x10: ACCUM_B – ACCUM_A –> ACCUM_B, Byte (LS-Byte, signed)
  0x11: ACCUM_B – ACCUM_A –> ACCUM_B, Halfword (LS-16-bits, signed)
  0x12: ACCUM_B – ACCUM_A –> ACCUM_B, Word (32-bits, signed)
  USER_DATA: don't care

--- MULTIPLICATION (plus ADDITION/SUBTRACTION option)

NOTES: For the following multiplication opcodes:
-If user does NOT want param byte value added to the multiplication result, set PARAM=0.
-To <u>subtract</u> a value from a <u>signed</u> multiplication result, set PARAM to a negative number.
In this case a negative number is created by setting the MS-bit (simple signed number).
Example: To subtract 5 from multiplication product, PARAM = 0x85 (0x5 with MS-bit set)

---ACCUM A

- OPCODE **0x20: Mult (unsigned) ACCUM_A by USER_DATA, Add PARAM**
  [Accum_A x USER_DATA] (unsigned) + PARAM (unsigned) -> Accum_A (unsigned)

- OPCODE **0x21: Mult (unsigned) ACCUM_A by USER_DATA, Subtract PARAM**
  [Accum_A x USER_DATA] (unsigned) - PARAM (unsigned) -> Accum_A (unsigned

- OPCODE **0x22: Mult (signed byte) ACCUM_A by USER_DATA, Add PARAM**
  [Accum_A x USER_DATA] (signed LS-byte) + PARAM (signed LS-byte) -> Accum_A (LS-byte)

- OPCODE **0x23: Mult (signed halfword: 16b) ACCUM_A by USER_DATA, Add PARAM**
  [Accum_A x USER_DATA] (signed LS-16b) + PARAM (signed LS-16b) -> Accum_A (LS-16b)

- OPCODE **0x24: Mult (signed word: 32b) ACCUM_A by USER_DATA, Add PARAM**
  [Accum_A (signed 32b) x USER_DATA (signed 32b)] + PARAM (signed 32b) -> Accum_A

  ---ACCUM B
- OPCODE **0x25: Mult (unsigned) ACCUM_B by USER_DATA, Add PARAM**
  [Accum_B x USER_DATA] (unsigned) + PARAM (unsigned) -> Accum_B (unsigned)

- OPCODE **0x26: Mult (unsigned) ACCUM_B by USER_DATA, Subtract PARAM**
  [Accum_B x USER_DATA] (unsigned) + PARAM (unsigned) -> Accum_B (unsigned)

- OPCODE **0x27: Mult (signed byte) ACCUM_B by USER_DATA, Add PARAM**
  [Accum_B x USER_DATA] (signed LS-byte) + PARAM (signed LS-byte) -> Accum_B (LS-byte)

- OPCODE **0x28: Mult (signed halfword: 16b) ACCUM_B by USER_DATA, Add PARAM**
  [Accum_B x USER_DATA] (signed LS-16b) + PARAM (signed LS-16b) -> Accum_B (LS-16b)

- OPCODE **0x29: Mult (signed word: 32b) ACCUM_B by USER_DATA, Add PARAM**
  [Accum_B (signed 32b) x USER_DATA (signed 32b)] + PARAM (signed 32b) -> Accum_B

  ---Both Accumulators

- OPCODE **0x2A: Mult (unsigned) ACCUMULATORS**
  Multiply data in one accumulator by data in the other accumulator (unsigned multiplication)
  Accum_A (unsigned) x ACCUM_B (unsigned) -> Accum_A or B (unsigned)
  PARAM: sets which Accumulator to place result (destination):
  0=ACCUM_A (default) 1=ACCUM_B
  USER_DATA: don't care

- OPCODE **0x2B: Mult (signed) ACCUMULATORS**
  Multiply data in one accumulator by data in the other accumulator (signed multiplication)
  Accum_A (signed) x ACCUM_B (signed) -> Accum_A or B (signed)
  PARAM sets which Accumulator is destination and data size:
  0=ACCUM_A, Byte (LS-Byte, signed) 0x10=ACCUM_B, Byte (LS-Byte, signed)
  1=ACCUM_A, Halfword (LS-16-bits, signed) 0x11=ACCUM_B, Halfword (LS-16-bits, signed)
  2=ACCUM_A, Word (32-bits, signed) 0x12=ACCUM_B, Word (32-bits, signed)
  USER_DATA: don't care

--- DIVISION (plus Addition/Subtraction option)

NOTES: For the following division opcodes:
-If user does NOT want param byte value added to the division result, set PARAM=0.
-To <u>subtract</u> a value from a <u>signed</u> division result, set PARAM to a negative number.
In this case a negative number is created by setting the MS-bit (simple signed number).
Example: To subtract 5 from the division result, PARAM = 0x85 (0x5 with MS-bit set)

---ACCUM A

- OPCODE **0x30: Divide (unsigned) ACCUM_A by USER_DATA, Add PARAM**
  [Accum_A ÷ USER_DATA] (unsigned) + PARAM (unsigned) -> Accum_A (unsigned)

- OPCODE **0x31: Divide (unsigned) ACCUM_A by USER_DATA, Subtract PARAM**
  [Accum_A ÷ USER_DATA] (unsigned) - PARAM (unsigned) -> Accum_A (unsigned)

- OPCODE **0x32: Divide (signed byte) ACCUM_A by USER_DATA, Add PARAM**
  [Accum_A ÷ USER_DATA] (signed LS-byte) + PARAM (signed LS-byte) -> Accum_A (signed LS-byte)

- OPCODE **0x33: Divide (signed halfword: 16b) ACCUM_A by USER_DATA, Add PARAM**
  [Accum_A ÷ USER_DATA] (signed LS-16b) + PARAM (signed LS-16b) -> Accum_A (signed LS-16b)

- OPCODE **0x34: Divide (signed word: 32b) ACCUM_A by USER_DATA, Add PARAM**
  [Accum_A ÷ USER_DATA] (signed 32b) + PARAM (signed 32b) -> Accum_A (signed 32b)

---ACCUM B

- OPCODE **0x35: Divide (unsigned) ACCUM_B by USER_DATA, Add PARAM**
  [Accum_B ÷ USER_DATA] (unsigned) + PARAM (unsigned) -> Accum_B (unsigned)

- OPCODE **0x36: Divide (unsigned) ACCUM_B by USER_DATA, Subtract PARAM**
  [Accum_B ÷ USER_DATA] (unsigned) - PARAM (unsigned) -> Accum_B (unsigned)

- OPCODE **0x37: Divide (signed byte) ACCUM_B by USER_DATA, Add PARAM**
  [Accum_B ÷ USER_DATA] (signed LS-byte) + PARAM (signed LS-byte) -> Accum_B (signed LS-byte)

- OPCODE **0x38: Divide (signed halfword: 16b) ACCUM_B by USER_DATA, Add PARAM**
  [Accum_B ÷ USER_DATA] (signed LS-16b) + PARAM (signed LS-16b) -> Accum_B (signed LS-16b)

- OPCODE **0x39: Divide (signed word: 32b) ACCUM_B by USER_DATA, Add PARAM**
  [Accum_B ÷ USER_DATA] (signed 32b) + PARAM (signed 32b) -> Accum_B (signed 32b)

---Both Accumulators

- OPCODE **0x3A: Divide (unsigned) ACCUMULATORS**
  Divides the data in one accumulator from the data in the other accumulator (unsigned division)
  Accum_A (unsigned) ÷ Accum_B (unsigned) -> Accum_A (unsigned), or,
  Accum_B (unsigned) ÷ Accum_A (unsigned) -> Accum_B (unsigned)
  PARAM sets Accumulator order/destination:
  0=ACCUM_A ÷ ACCUM_B -> ACCUM_A (default)
  1=ACCUM_B ÷ ACCUM_A -> ACCUM_B
  USER_DATA: don’t care

- OPCODE **0x3B: Divide (signed) ACCUMULATORS**
  Divides the data in one accumulator from the data in the other accumulator (signed division)
  If destination is Accum_A: Accum_A (signed) ÷ ACCUM_B (signed) -> Accum_A (signed)
  If destination is Accum_B: Accum_B (signed) ÷ ACCUM_A (signed)-> Accum_B (signed)
  PARAM sets which Accumulator destination and data size:
  0=ACCUM_A, Byte (LS-Byte, signed) 0x10=ACCUM_B, Byte (LS-Byte, signed)
  1=ACCUM_A, Halfword (LS-16-bits, signed) 0x11=ACCUM_B, Halfword (LS-16-bits, signed)
  2=ACCUM_A, Word (32-bits, signed) 0x12=ACCUM_B, Word (32-bits, signed)
  USER_DATA: don’t care

3.4 BIT/LOGICAL OPERATIONS & BYTE SWAP

--- MASK, SHIFT – USER DATA

- OPCODE **0x40: ACCUM_A AND USER_DATA, Shift Right PARAM**
  (Accum A AND'd with User Data, then shifted right by number of bits indicated in PARAM)
  [Accum_A & USER_DATA] shifted right by PARAM (unsigned) -> Accum_A
  Note: If PARAM=0, no shift

- OPCODE **0x41: ACCUM_A AND USER_DATA, Shift Left PARAM**
  (Accum A AND'd with User Data, then shifted left by number of bits indicated in PARAM)
  [Accum_A & USER_DATA] shifted left by PARAM (unsigned) -> Accum_A
  Note: If PARAM=0, no shift

- OPCODE **0x42: ACCUM_A OR USER_DATA, Shift Right PARAM**
  (Accum A OR'd with User Data, then shifted right by number of bits indicated in PARAM)
  [Accum_A OR USER_DATA] shifted right by PARAM (unsigned) -> Accum_A
  Note: If PARAM=0, no shift

- OPCODE **0x43: ACCUM_A OR USER_DATA, Shift Left PARAM**
  (Accum A OR'd with User Data, then shifted left by number of bits indicated in PARAM)
  [Accum_A OR USER_DATA] shifted left by PARAM (unsigned) -> Accum_A
  Note: If PARAM=0, no shift

- OPCODE **0x44: ACCUM_A XOR USER_DATA, Shift Right PARAM**
  (Accum A Exclusive OR'd with User Data, then shifted right by number of bits indicated in PARAM)
  [Accum_A XOR USER_DATA] shifted right by PARAM (unsigned) -> Accum_A
  Note: If PARAM=0, no shift

- OPCODE **0x45: ACCUM_A XOR USER_DATA, Shift Left PARAM**
  (Accum_A Exclusive OR'd with User Data, then shifted left by number of bits indicated in PARAM)
  [Accum_A XOR USER_DATA] shifted left by PARAM (unsigned) -> Accum_A
  Note: If PARAM=0, no shift

- OPCODE **0x46: ACCUM_A AND ACCUM_B**
  Accum_A AND'd with ACCUM_B
  [Accum_A & ACCUM_B] -> Accum_A
  PARAM and USER_DATA: don't care

- OPCODE **0x47: ACCUM_A OR ACCUM_B**
  Accum_A OR'd with ACCUM_B
  [Accum_A OR ACCUM_B] -> Accum_A
  PARAM and USER_DATA: don't care

- OPCODE **0x48: ACCUM_A XOR ACCUM_B**
  Accum_A Exclusive-OR'd with ACCUM_B
  [Accum_A XOR ACCUM_B] -> Accum_A
  PARAM and USER_DATA: don't care

--- BYTE SWAP – ENDIANESS

- OPCODE **0x4C: Byte Swap 32-bits ACCUM_A**

Reverse order of all 4 bytes in ACCUM_A
Accum_A: [Byte3 | Byte2 | Byte1 | Byte 0] -> [Byte0 | Byte1 | Byte2 | Byte3]
PARAM and USER_DATA: don't care

- OPCODE **0x4D: Byte Swap 16-bits ACCUM_A**
  Reverse order highest 2 bytes and reverse order of lower 2 bytes in ACCUM_A
  Accum_A: [Byte3 | Byte2 | Byte1 | Byte 0] -> [Byte2 | Byte3 | Byte0 | Byte1]
  PARAM and USER_DATA: don't care

- OPCODE **0x4E: Halfword Swap ACCUM_A**
  Swap high 16-bits with lower 16-bits in ACCUM_A
  Accum_A: [Byte3 | Byte2 | Byte1 | Byte 0] -> [Byte1 | Byte0 | Byte3 | Byte2]
  PARAM and USER_DATA: don't care

3.5 MISC CONTROL, DELAY, LED

- OPCODE **0x50: Delay for USER_DATA x 1 mS ***
  Delay for period = USER_DATA x 1 millisecond (1/1000 sec increments)
  PARAM: don't care
  USER_DATA: Delay (number of milliseconds)

- OPCODE **0x51: LED ON for delay period: USER_DATA x 1 mS ***
  LED ON, then delay, then LED OFF. Delay specified by USER_DATA
  PARAM: don't care
  USER_DATA: Delay (number of milliseconds)

- OPCODE **0x52: LED flash: Flash frequency = USER_DATA x 1 mS, Count ***
  LED flashes at selected frequency. Flash period = USER_DATA x 1 millisecond),
  PARAM: flash count (number of flash cycles)
  USER_DATA: Flash period (number of milliseconds)
  NOTE: Minimum is 200 Milliseconds. If lower, it will be forced to 200 mSecs: 1/5 second)

*NOTE: Not a background operation, will delay subsequent processing

3.6 PROGRAM CONTROL

--- COUNTERS, REGS, RAM INDEX, FLAGS

- OPCODE **0x60: Reset Offload State Machine**
  PARAM and USER_DATA = Reserved, set=0
  Resets Accumulators, counters, flags, command instruction sequence, etc.
  The following reset actions will take place:
    Accumulators, both set = 0
    Counter1, Counter2, IO_counter, Ram Index set = 0
    I2C_Device_Adrs, I2C_byte_order, Compare flag set = 0
    The list of commands used for branching will be cleared
      (cannot branch back before this command was executed)

  NOTE: This reset also occurs when the APP first runs an offload command

--- BRANCHES

NOTE: Branching uses the value in PARAM to determine how far to jump back/forward.
That is to say, the "branch value" is the number of offload commands forward or backward that the "instruction pointer" will move to. A simple signed byte is used for branch value.
Users are expected to branch within their range of instructions.
For example: To jump 4 commands back, use -4 in PARAM with simple signed value:
Set PARAM=0x84, (0x04 with MS-bit set to indicate negative – backward jump).

- OPCODE **0x68: Branch Always**
  Always branches.
  PARAM: holds relative branch value: forward or backward, number of offload commands.
  PARAM byte is a signed value, MS-bit = 1 for negative (branch backwards).
  For example: To branch back 2 commands, set PARAM = 0x84 (0x04 with MS-bit set to go back)
  NOTE: If PARAM=0, Special Case: continues to loop on this instruction indefinitely
  USER_DATA: don't care

- OPCODE **0x69: Branch when Counter Not Zero**
  Branches when the selected counter/index is NOT equal to zero.
  PARAM: holds branch value (number of commands to jump).
  PARAM byte is signed value, MS-bit = 1 for negative (branch backwards).
  NOTE: If PARAM=0, NOP
  USER_DATA: selects counter to compare to zero:
  0: IO_Counter not=0
  1: Counter1 not=0
  2: Counter2 not=0
  3,4: Reserved
  5: RAM_Index not=0

- OPCODE **0x6A: Branch on Compare Flag Set**
  Branch when compare flag is set (number of commands to jump).
  PARAM byte is signed value, MS-bit = 1 for negative (branch backwards).
  NOTE: If PARAM=0, NOP
  USER_DATA: don't care

--- COMPARES & BRANCH-ON-COMPARE

NOTES:
For the following compares, the branching is determined are the same as above. That is to say, the "branch value" is the number of offload commands forward or backward that the "instruction pointer" will move to. A simple signed byte is used for branch value.

The "compare flag" is set TRUE or FALSE according to the compare with the following commands.

---USER_DATA & ACCUM_A

- OPCODE **0x70: Compare: USER_DATA Not Equal ACCUM_A**
  Compares USER_DATA value to ACCUM_A. Will branch when "NOT EQUAL"=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: first operand in compare.

- OPCODE **0x71: Compare: USER_DATA Equal ACCUM_A**
  Compares USER_DATA value to ACCUM_A. Will branch when "EQUAL"=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: first operand in compare.

- OPCODE **0x72: Compare: USER_DATA Greater Than ACCUM_A**
  Compares USER_DATA value to ACCUM_A. Will branch when “GREATER THAN”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: first operand in compare.

- OPCODE **0x73: Compare: USER_DATA Less Than ACCUM_A**
  Compares USER_DATA value to ACCUM_A. Will branch when “LESS THAN”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: first operand in compare.

- OPCODE **0x74: Compare: USER_DATA Greater Than Or Equal ACCUM_A**
  Compares USER_DATA value to ACCUM_A. Will branch when “GREATER THAN OR EQUAL”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: first operand in compare.

- OPCODE **0x75: Compare: USER_DATA Less Than Or Equal ACCUM_A**
  Compares USER_DATA value to ACCUM_A. Will branch when “LESS THAN OR EQUAL”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: first operand in compare.

  ---ACCUM_A and ACCUM_B

- OPCODE **0x76: Compare: ACCUM_A Not Equal ACCUM_B**
  Compares ACCUM_A to ACCUM_B contents. Will branch when “NOT EQUAL”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: don’t care

- OPCODE **0x77: Compare: ACCUM_A Equal ACCUM_B**
  Compares ACCUM_A to ACCUM_B contents. Will branch when “EQUAL”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: don’t care

- OPCODE **0x78: Compare: ACCUM_A Greater Than ACCUM_B**
  Compares ACCUM_A to ACCUM_B. Will branch when “GREATER THAN”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: don’t care

- OPCODE **0x79: Compare: ACCUM_A Less Than ACCUM_B**
  Compares ACCUM_A to ACCUM_B. Will branch when “LESS THAN”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: don’t care

- OPCODE **0x7A: Compare: ACCUM_A Greater Than or Equal ACCUM_B**
  Compares ACCUM_A to ACCUM_B contents. Will branch when “GREATER THAN OR EQUAL”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: don’t care

- OPCODE **0x7B: Compare: ACCUM_A Less Than or Equal ACCUM_B**
  Compares ACCUM_A to ACCUM_B contents. Will branch when “LESS THAN OR EQUAL”=TRUE as an option.
  PARAM: Holds the branch value (a simple signed value). 0=No branch
  USER_DATA: don’t care

3.7 INPUT-OUTPUT

--- I2C READ

- OPCODE **0x80: I2C Read bytes to ACCUM_A**
  Reads I2C byte(s) to ACCUM_A. Can read 1, 2 or 4 bytes.
  PARAM holds Register offset/counter action following read:
  0=No offset or IO_Counter change
  1=Increment I2C address register offset after transfer
  USER_DATA: holds I2C device 7-bit address, I2C device register offset,
  IO_Counter initial value (transfer count: 1, 2 or 4)
  BYTE 0 (LS-Byte): I2C device 7-bit address (this sets I2C_Adrs_Reg)
  BYTE 1 (Next LS-byte): I2C device register offset (2nd I2C byte), (Sets I2C_Reg_Offset)
  BYTE 2 (Next byte): Transfer count (IO_Counter) initial value
  BYTE 3 (MS-Byte): Reserved, set=0
  NOTES:
  MUST HAVE BYTE ORDER SET in I2C_byte_order when reading multiple bytes:
  Can use opcode 0x0A (Set/Update Vars) to set byte order.
  I2C Byte order: 0=MSB first (default), 1=LS Byte first.
  SiLabs Si7020 part (Temp/Humid) is MSB first
  Silabs Si1147 part (Light/HRM/Prox)is LSB first
  Early Odyssey FPGA examples were MSB first (but is programmable)

- OPCODE **0x81: I2C Read bytes to RAM**
  Reads I2C byte(s), up to 256 to User_RAM. (Array of 256 available)
  PARAM holds register offset/RAM_index/counter action following read:
  0=No offset or RAM_Index or IO_Counter change: single read
  1=Increment I2C register offset (I2C_Reg_Offset), Increment RAM_Index,
  Decrement IO_Counter
  2=Increment RAM_Index, Decrement IO_Counter, (No change to I2C_Reg_Offset)
  3=Decrement RAM_Index, Decrement IO_Counter, (No change to I2C_Reg_Offset)
  USER_DATA holds: I2C device 7-bit address, I2C Device register offset,
  IO_Counter initial value (transfer count) and RAM_Index initial value:
  LS-BYTE: I2C device 7-bit address (Sets I2C_Adrs_Reg)
  BYTE 1 (Next LS-byte): I2C device register offset (2nd I2C byte), (Sets I2C_Reg_Offset)
  BYTE 2 (Next byte): Transfer count (IO_Counter) initial value
  BYTE 3 (MS-Byte): RAM_Index initial value
  NOTES:
  Must set IO_delay (time between transfers), or assume reset value of 1 second.
  MUST HAVE BYTE ORDER SET in I2C_byte_order when reading multiple bytes:
  Can use opcode 0x0A (Set/Update Vars) to set byte order and/or IO_delay time.
  I2C Byte order: 0=MS Byte first (default), 1=LS Byte first
  SiLabs Si7020 part (Temp/Humid) is MSB first
  Silabs Si1147 part (Light/HRM/Prox)is LSB first
  Early Odyssey FPGA examples were MSB first (but is programmable)

--- I2C WRITE

- OPCODE **0x85: I2C Write byte from USER_DATA**
  I2C Write a single byte to device from USER_DATA (LS-Byte).
  PARAM holds I2C register offset action following read:

0=No I2C register offset
1=Increment I2C register offset after transfer
USER_DATA holds: I2C device 7-bit address, I2C device register offset,
IO_Counter (transfer count) initial value:
LS-BYTE: I2C device 7-bit address (this sets I2C_Adrs_Reg)
BYTE 1 (Next LS-byte): I2C device register offset (2nd I2C byte), Sets I2C_Reg_Offset initial value
BYTE 2 (Next byte): Data byte to be written
BYTE 3 (MS-Byte): Reserved, set=0
NOTES:
MUST HAVE BYTE ORDER SET in I2C_byte_order when reading multiple bytes:
Can use opcode 0x0A (Set/Update Vars) to set byte order.
I2C Byte order: 0=MS Byte first (default), 1=LS Byte first
SiLabs Si7020 part (Temp/Humid) is MSB first
Silabs Si1147 part (Light/HRM/Prox)is LSB first
Early Odyssey FPGA examples were MSB first (but is programmable)

- OPCODE **0x86: I2C Write byte(s) from ACCUM_A**
I2C Write bytes (1, 2 or 4) to I2C device from ACCUM_A
PARAM holds I2C register offset, counter action following read:
0=No I2C register offset or IO_Counter change
1=Increment I2C register offset (I2C_Reg_Offset)
USER_DATA holds: I2C device 7-bit address, I2C device register offset,
IO_Counter (transfer count) initial value:
LS-BYTE: I2C device 7-bit address (this sets I2C_Adrs_Reg)
BYTE 1 (Next LS-byte): I2C device register offset (2nd I2C byte), Sets I2C_Reg_Offset initial value
BYTE 2 (Next byte): Transfer byte count (1, 2 or 4 only), Sets IO_Counter initial value
BYTE 3 (MS-Byte): Reserved, set=0
NOTES:
ACCUM_A holds: Byte(s) to be written: LS-Byte, LS-halfword or all 4 bytes (LS byte first)
MUST HAVE BYTE ORDER SET in I2C_byte_order when reading multiple bytes:
Can use opcode 0x0A (Set/Update Vars) to set byte order.
I2C Byte order: 0=MS Byte first (default), 1=LS Byte first
SiLabs Si7020 part (Temp/Humid) is MSB first
Silabs Si1147 part (Light/HRM/Prox)is LSB first
Early Odyssey FPGA examples were MSB first (but is programmable)

- OPCODE **0x87: I2C Write byte(s) from RAM**
Writes I2C byte(s) to device from User_RAM. Array of 256 user RAM bytes available.
PARAM holds offset/RAM_index/counter action following write:
0=No offset or RAM_Index or IO_Counter change: single write
1=Increment I2C register offset (I2C_Reg_Offset), Increment RAM_Index,
Decrement IO_Counter
2=Increment RAM_Index, (No change to I2C_Reg_Offset)
3=Decrement RAM_Index, (No change to I2C_Reg_Offset)
USER_DATA holds: I2C device 7-bit address, I2C Device register offset,
IO_Counter (transfer count) initial value and RAM_Index initial value:
LS-BYTE: I2C device 7-bit address (Sets I2C_Adrs_Reg)
BYTE 1 (Next LS-byte): I2C device register offset (2nd I2C byte), (Sets I2C_Reg_Offset)
BYTE 2 (Next byte): Transfer count (IO_Counter) initial value
BYTE 3 (MS-Byte): RAM_Index initial value
NOTES:
Must set IO_delay (time between transfers), or assume reset value of 1 sec.
MUST HAVE BYTE ORDER SET in I2C_byte_order when reading multiple bytes:

Can use opcode 0x0A (Set/Update Vars) to set byte order and/or IO_delay.
I2C Byte order: 0=MS Byte first (default), 1=LS Byte first
SiLabs Si7020 part (Temp/Humid) is MSB first

3.8 MISCELLANEOUS / OTHER

- OPCODE **0xE1: Copy Variable to Accumulator**
  Copies a counter, RAM index, other variables to an accumulator.
  PARAM selects the destination Accumulator:
  0: ACCUM_A
  1: ACCMU_B
  USER_DATA: selects the variable to be copied

```
0: IO_counter           4: I2C_adrs_reg
1: Counter1             5: I2C_byte_order
2: Counter2             6: IO_delay
3: User_ram_index
```

3.9 TEST & DIAGNOSTICS

- OPCODE **0xF1: Setup Test OPCODE**
  Sets a value for “test OPCODE” to be used when running “Run Test Command” (opcode below).
  PARAM: don’t care
  USER_DATA (LS-byte) holds the value for the test OPCODE byte

- OPCODE **0xF2: Setup Test PARAM**
  Sets a value for “test PARAM” to be used when running “Run Test Command” (opcode below).
  PARAM: don’t care
  USER_DATA (LS-byte) holds the value for the test PARAM byte

- OPCODE **0xF3: Setup Test USER_DATA**
  Sets a value for “test USER_DATA” to be used when running “Run Test Command” (opcode below).
  PARAM: don’t care
  USER_DATA holds the value for the test USER_DATA

- OPCODE **0xF4: Run Test Command**
  This command assumes that a test opcode and test values are setup for PARAM and USER_DATA with the “Setup Test OPCODE”, “Setup Test PARAM” and “Setup Test USER_DATA” commands.
  This command will then run the offload command created by combining them.
  PARAM and USER_DATA for this opcode: don’t care

Appendix A: Offload Processing Considerations Using Math

As with any compiler, users must take care and be clear what size their data is when working with signed math. For example, signed arithmetic must use operands the same size. This is due to the position of the sign bit (and 2's complement conversions).

Likewise, when setting up a return field (for a button) in the Web App to display data for a personality, the size and type of integer must be set correctly.

Here are the sizes users may work with:

- Signed Word: 32-bit signed (INT32) – cannot combine sizes for math
- Signed Halford: 16-bit signed (INT16) – cannot combine sizes for math
- Signed Byte: 8-bit signed (INT8) – cannot combine sizes for math
- Unsigned Word: 32-bit unsigned (UINT32) – can combine sizes (not signed)
- Unsigned Halfword: 16-bit unsigned (UINT16) – can combine sizes (not signed)
- Unsigned Byte: 8-bit unsigned (UINT8) – can combine sizes (not signed)

See the opcode available for converting signed integer sizes:

OPCODE 0x06: Convert signed data size in ACCUMULATOR

An example would be getting a byte of signed data from a sensor and multiplying it by a signed word. The byte would need to be converted to a signed word first.

To make things easier for the user, a simple signed value is used for user entry. This is merely adding a sign bit to any value to make it a negative number. The state machine then converts it to a standard signed integer (2's complement) for doing math internally.

Simple signed value

User wants to load the accumulator with a byte value of -4.

Take 0x04 and add the sign bit (MS-bit) to the byte value to get 0x84.
For a 16-bit value: 0x8004
For a 32-bit word: 0x80000004

A simple signed value should be used for negative numbers for the following opcodes and inputs:

- Opcode 0x02: Load Accumulator with User Data (signed)
- All Math opcodes where User Data is an input for signed math
- All signed Multiplication and Division opcodes for generating a negative PARAM byte
- For branch values (byte) – all opcodes that have branches (usually the PARAM byte, negative for branching back)

Appendix B: New Personalities & Example Entries

Here are two new personalities created with offload commands along with tables showing the command entries: *Temp Collection* and *Custom Command Test*. Users can build and/or modify these personalities using the Web Application. Users can learn a lot by looking at the Temp Collection Personality (I2C looped collection to RAM, math conversion, timed delays, loops using branching, led flashing, auto-indexing RAM, etc).

1. Temp Collection Personality

- Loop collects several timed samples of raw temperature data via I2C, converts to Fahrenheit and stores in User RAM
- One button displays Fahrenheit temperature values. It auto-indexes the User RAM so the same button can continue to read out collected temperatures (in Fahrenheit).
- A second button converts samples to Celsius and displays the temperature. It auto-indexes the User RAM so the same button can continue to read out collected temperatures (in Celsius).
- Public example collects 8 samples, 2 seconds apart. Users can copy and edit the personality to collect over seconds/hours/days and collect up to 256 samples (256 bytes user RAM).
- LED lights briefly when sampling and for 4 seconds when done.

NOTES:

Runs immediately upon entering personality ("Setup Commands" run first).
The buttons use the same index, so if a user uses both buttons, they will index, showing different samples.
If the user wishes to enter data for timing or number of samples, they can create a separate small personality that enters and saves these values, then edit the existing personality to use the data. Data can be left unchanged between personalities (will want to eliminate the "reset all" command below)

TIP: Putting a finger on the temp sensor (white top) during sampling will help collect a variation of temperature readings.

Example Usage:

- Select "Temp Collection" personality from list (starts <u>immediately</u> and buttons are <u>disabled</u>)
  -See LED flashing every 2 seconds – at each sampling of temperature
- Gently put finger on temp sensor (white top) after the 1st or 2nd flash, until 6th flash or so
- After 8 samples/flashes, the LED stays lit for 4 seconds indicating the temperature collection is complete
  -Buttons are now enabled
- Press either button repeatedly to display the 8 samples in order (Fahrenheit or Celsius)
  -Each button uses the same samples and auto-indexes to the next, so 8 total button presses will display
  -After 8 presses, the display will show whatever is in user ram after 8th location

NOTE: See the list of commands below to understand how to do loops (using branching), LED control, I2C I/O and delays.

Temp Collection Command Entries

Values entered or selected in Web Application are highlighted

Setup Commands (15 Max)

Offload CMD type	Opcode Name or Operation	*Opcode value*	*Parameter byte*	*Write source*	*User Data word*	Notes
Command type field		***I2C device adrs*** field	***Register adrs*** field	***Fixed or Entry field***	***Fixed write data*** field	

Offload CMD	Reset state machine	0x60	0x00	Fixed	0x00	Reset all: Accum, counters, RAM index…
Offload CMD	Set vars: set Counter1	0x0a	0x03	Fixed	0x08	Counter 1 set = 0x08 (loop counter)
Offload CMD	LED on .25 secs	0x51	0x00	Fixed	0x100	LED On .25 secs (256 mS)
Offload CMD	I2C read 2 bytes to Accum A	0x80	0x00	Fixed	0x0002e340	I2C read Temp reg, I2C adrs 0x40, reg offset 0xE3, 2 bytes
Offload CMD	Mult Accum A x User Data	0x20	0x00	Fixed	0x3dc7	Accum A x 0x3DC7, + 0
Offload CMD	Divide Accum A by User Data	0x31	0x35	Fixed	0x320000	Accum A ÷ 0x320000, + 0x35
Offload CMD	Write byte to User RAM	0x0c	0x01	Fixed	0x00	Write Accum A LS-byte to RAM, increment RAM index
Offload CMD	Delay	0x50	0x00	Fixed	0x800	Delay 2 secs (2048 mS)
Offload CMD	Set vars: decrement Counter1	0x0a	0x04	Fixed	0x00	Decrement Counter 1
Offload CMD	Branch if Counter1 not=0	0x69	0x87	Fixed	0x01	Branch if Counter 1 not = 0, back 7 commands (to the LED On cmd)
Offload CMD	Set vars: reset counters, index	0x0a	0x11	Fixed	0x00	Reset Counters, RAM index = 0
Offload CMD	LED on 2 secs	0x51	0x00	Fixed	0x800	LED On 2 secs (2048 mS) – indicate DONE

Button Commands (5 Max each)

Offload CMD type	Opcode Name	*Opcode* value	*Parameter* byte	*Write source*	*User Data* word	Notes / Return field
Command type field		***I2C device adrs*** field	***Register adrs*** field	***Fixed or Entry field***	***Fixed write data*** field	
Button 1						
Button Text: Stored Temps F **Return Field Details:** ON, uint8, decimal, "F" **Entry Field Details:** OFF		**Visible:** Checked				
Offload CMD	Read User RAM: 1 byte	0x0b	0x00	Fixed	0x00	Read RAM to Accum A, increment RAM index. (Fahrenheit value)
Offload READ	N/A	N/A	N/A	N/A	N/A	Read Accum A and display* (Return field 1)
Button 2						
Control Name: Stored Temps C **Return Field Details:** ON, uint8, decimal, "C" **Entry Field Details:** OFF		**Visible:** Checked				
Offload CMD	Read User RAM: 1 byte	0x0b	0x00	Fixed	0x00	Read RAM to Accum A, increment RAM index. (Fahrenheit value)
Offload CMD	Subtract User data from Accum A	0x15	0x00	Fixed	0x20	Accum A - 32
Offload CMD	Multiply Accum A by User Data	0x20	0x00	Fixed	0x05	Accum A x 5
Offload CMD	Divide Accum A by User Data	0x30	0x00	Fixed	0x09	Accum A ÷ 9 (now Celsius)
Offload READ	N/A	N/A	N/A	N/A	N/A	Read Accum A and display* (Return field 2)

NOTE: Write source can be "Fixed" or if using user data entry, can be "Entry Field 1" thru "Entry field 8" (aligns with button number)

2. Custom Command Test Personality

- Allows user to test custom commands without creating a personality first
- Users enter Opcode, Parameter and User Data into the personality input fields and hit the "execute" button
- Other buttons allow data results to be read from the Accumulator in various formats
- Users can chain several entries to simulate a personality

Example Usage:

This example simply shows a way to enter the "*Load accumulator from User data*" opcode, which can then be read out and displayed using the "Read Accumulator" button. Users can enter any command in the opcode list (see reference in section 3 of this document).

- Select "Custom Command Test" Personality from list
- Next to the "Enter Opcode" button, enter ***01***, then press the button
  -Enters opcode 0x01 which is "Load Accumulator from User Data" (see reference in section 3)

- Next to the "Enter Param byte" button, enter ***0***, then press the button
  -Parameter byte set to 0 selects Accumulator A (default accumulator) for destination
- Next to the "Enter User Data" button, enter ***123456aa***, then press the button
  -This is a random hex value, a full word (4 bytes) so we recognize it. (Can load any hex value up to 4 bytes long)
- Press the "Read Accum hex" button to read contents of the accumulator in hex format
  -Will see 0 (or whatever is already in accumulator) because the command has not been executed
- Press the "Execute Command" button to run the "Load accumulator…" opcode (loading the entered data)
  -Runs the command, loading Accumulator A with contents 0x123456aa
- Press the "Read Accum hex" button again to see the data loaded above
  -Should now see the data we loaded because the command was executed: 0x123456aa

NOTES:
-Now you can continue to enter opcodes (with parameter bytes and user data) to operate on loaded data, I2C I/O, LED or delay functions, etc. You can read the contents of the accumulator at any time with the "Read Accum" buttons.
-The other "Read Accum…" buttons display the value in decimal formats: unsigned 32-bit, signed 8-bit (int8). The signed button will show negative byte values with a minus sign. For displaying signed integers, you must match the size correctly… for example int32 only converts 32-bit values correctly, and int8 for bytes. The buttons are setup for the return field size and type in the web application. See below data entered in the web application and Appendix A.

Custom Command Test - Command Entries

Values entered or selected in Web Application are highlighted

Setup Commands

Offload CMD type	Opcode Name or Operation	*Opcode* value	*Parameter* byte	*Write source*	*User Data* word	Notes
Command type field		*I2C device adrs* field	*Register adrs* field	*Fixed or Entry field*	*Fixed write data* field	
Offload CMD	Reset state machine	0x60	0x00	Fixed	0x00	Reset all: Accum, counters, RAM index…

Button Commands

Offload CMD type	Opcode Name or Operation	*Opcode* value	*Parameter* byte	*Write source*	*User Data* word	Notes / Return field
Command type field		*I2C device adrs* field	*Register adrs* field	*Fixed or Entry Field*	*Fixed write data* field	
Button 1						
Button Text: Enter Opcode **Return Field Details:** N/A **Entry Field Details:** ON, hex, 0..0xff		**Visible:** checked				
Offload CMD	Load test Opcode	0xf1	0x00	Entry Field 1	0	Enter test opcode
Button 2						
Button Text: Enter Param byte **Return Field Details:** N/A **Entry Field Details:** ON, hex, 0..0xff		**Visible:** checked				
Offload CMD	Load test Param byte	0xf2	0x00	Entry Field 2	0	Enter test Param byte
Button 3						
Button Text: Enter User Data **Return Field Details:** N/A **Entry Field Details:** ON, hex, 0..0xffffffff		**Visible:** checked				
Offload CMD	Load test User Data word	0xf3	0x00	Entry Field 3	0	Enter test User Data (up to word)
Button 4						
Button Text: Execute Command		**Visible:** checked				

Return Field Details: N/A **Entry Field Details:** N/A						
Offload CMD	Execute Test Command	0xf4	0x00	Fixed	0	Run the test command
Button 5						
Button Text: Read Accum hex **Return Field Details:** ON, UINT32, hex **Entry Field Details:** OFF		**Visible:** checked				
Offload READ	Read Accumulator to App	N/A	N/A	N/A	N/A	Read result in Accum to Return Field 5
Button 6						
Button Text: Read Accum dec **Return Field Details:** ON, UINT32, decimal **Entry Field Details:** OFF		**Visible:** checked				
Offload READ	Read Accumulator to App	N/A	N/A	N/A	N/A	Read result in Accum to Return Field 6
Button 7						
Button Text: Read Accum int8 **Return Field Details:** ON, INT8, decimal **Entry Field Details:** OFF		**Visible:** checked				
Offload READ	Read Accumulator to App	N/A	N/A	N/A	N/A	Read result in Accum to Return Field 7

Appendix C: Offload Processing Personality Creation Template

ODYSSEY OFFLOAD PROCESSING PERSONALITY – COMMANDS TEMPLATE

Use to plan command entry for creating offload processing personalities. See example filled in at bottom of file.

Setup Commands (15 Max)

Offload CMD type	Opcode Name or Operation	*Opcode value*	*Parameter byte*	*Write source*	*User Data* word	Notes
Command type field		***I2C device adrs*** field	***Register adrs*** field	***Fixed or Entry field***	***Fixed write data*** field	

Background Commands – Autonomous Field (5 Max)

Offload CMD type	Opcode Name or Operation	*Opcode value*	*Parameter byte*	*Write source*	*User Data* word	Notes
Command type field		***I2C device adrs*** field	***Register adrs*** field	***Fixed or Entry field***	***Fixed write data*** field	

Button Commands (5 Max each)

Offload CMD type	Opcode Name or Operation	*Opcode value*	*Parameter byte*	*Write source*	*User Data* word	Notes
Command type field		***I2C device adrs*** field	***Register adrs*** field	***Fixed or Entry Field***	***Fixed write data*** field	
Button 1						
Button Text: **Return Field Details:** **Entry Field Details:**		**Visible:**				
Button 2						
Button Text:		**Visible:**				

Return Field Details: **Entry Field Details:**						
Button 3						
Button Text: **Return Field Details:** **Entry Field Details:**		**Visible:**				
Button 4						
Button Text: **Return Field Details:** **Entry Field Details:**		**Visible:**				
Button 5						
Button Text: **Return Field Details:** **Entry Field Details:**		**Visible:**				
Button 6						
Button Text: **Return Field Details:** **Entry Field Details:**		**Visible:**				
Button 7						
Button Text: **Return Field Details:** **Entry Field Details:**		**Visible:**				
Button 8						
Button Text: **Return Field Details:** **Entry Field Details:**		**Visible:**				

EXAMPLE – filled in

Values entered or selected in Web Application are highlighted

Setup Commands (15 Max)

Offload CMD type	Opcode Name or Operation	*Opcode value*	*Parameter byte*	*Write source*	*User Data word*	Notes
Command type field		***I2C device adrs*** field	***Register adrs*** field	***Fixed or Entry field***	***Fixed write data*** field	
Offload CMD	Reset state machine	0x60	0x00	Fixed	0x00	Reset all: Accum, counters, RAM index…
Offload CMD	Set vars: set Counter1	0x0a	0x03	Fixed	0x08	Counter 1 set = 0x08 (loop counter)
Offload CMD	LED on .25 secs	0x51	0x00	Fixed	0x100	LED On .25 secs (256 mS)
Offload CMD	I2C read 2 bytes to Accum A	0x80	0x00	Fixed	0x0002e340	I2C read Temp reg, I2C adrs 0x40, reg offset 0xE3, 2 bytes
Offload CMD	Mult Accum A x User Data	0x20	0x00	Fixed	0x3dc7	Accum A x 0x3DC7, + 0
Offload CMD	Divide Accum A by User Data	0x31	0x35	Fixed	0x320000	Accum A ÷ 0x320000, + 0x35
Offload CMD	Write byte to User RAM	0x0c	0x01	Fixed	0x00	Write Accum A LS-byte to RAM, increment RAM index
Offload CMD	Delay	0x50	0x00	Fixed	0x800	Delay 2 secs (2048 mS)
Offload CMD	Set vars: decrement Counter1	0x0a	0x04	Fixed	0x00	Decrement Counter 1
Offload CMD	Branch if Counter1 not=0	0x69	0x87	Fixed	0x01	Branch if Counter 1 not = 0, back 7 commands (LED On cmd)
Offload CMD	Set vars: reset counters, index	0x0a	0x11	Fixed	0x00	Reset Counters, RAM index = 0
Offload CMD	LED on 2 secs	0x51	0x00	Fixed	0x800	LED On 2 secs (2048 mS) – indicate DONE

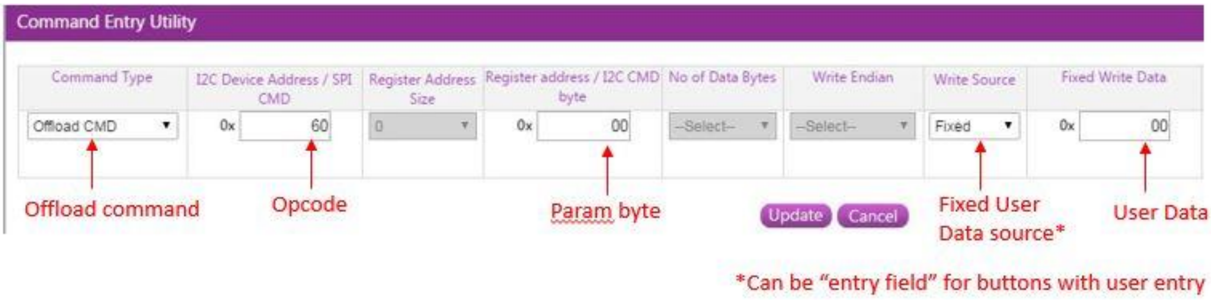
Button Commands (5 Max each)

Offload CMD type	Opcode Name	*Opcode value*	*Parameter byte*	*Write source*	*User Data word*	Notes / Return field
Command type field		***I2C device adrs*** field	***Register adrs*** field	***Fixed or Entry field***	***Fixed write data*** field	
Button 1						
Button Text: Stored Temps F **Return Field Details:** ON, uint8, decimal, "F" **Entry Field Details:** OFF		**Visible:** Checked				
Offload CMD	Read User RAM: 1 byte	0x0b	0x00	Fixed	0x00	Read RAM to Accum A, increment RAM index. (Fahrenheit value)
Offload READ	N/A	N/A	N/A	N/A	N/A	Read Accum A and display* (Return field 1)
Button 2						
Control Name: Stored Temps C **Return Field Details:** ON, uint8, decimal, "C" **Entry Field Details:** OFF		**Visible:** Checked				
Offload CMD	Read User RAM: 1 byte	0x0b	0x00	Fixed	0x00	Read RAM to Accum A, increment RAM index. (Fahrenheit value)
Offload CMD	Subtract User data from Accum A	0x15	0x00	Fixed	0x20	Accum A - 32
Offload CMD	Multiply Accum A by User Data	0x20	0x00	Fixed	0x05	Accum A x 5
Offload CMD	Divide Accum A by User Data	0x30	0x00	Fixed	0x09	Accum A ÷ 9 (now Celsius)
Offload READ	N/A	N/A	N/A	N/A	N/A	Read Accum A and display* (Return field 2)

NOTE: Write source can be "Fixed" or if using user data entry, can be "Entry Field 1" thru "Entry field 8" (aligns with button number)

Example Web Application Entry of the 1st command in the 1st example table above (Reset state machine):

Opcode: 0x60 Param byte: 0x00 User Data: 0x00



Appendix D: Odyssey I2C Addresses & Byte Order

Here are the I2C addresses and standard byte order for existing I2C interfaces on the Odyssey kit. Note that the address and order for the MAX10 FPGA are from designs that were included in the original personalities and are programmable (subject to the specific FPGA design).
NOTE: The Accelerometer is on a SPI bus that is unavailable to the Offload Processing state machine.

Device with I2C Interface	I2C Device Address	Byte order
Silabs Si7020 Temp/Humidity Sensor	0x40	MSB first
Silabs Si1147 Light/Proximity/HRM sensor	0x60	LSB first
MAX10 FPGA	0x30*	MSB first

*Original FPGA design personalities, subject to FPGA design

The default (power-on) byte order for the Odyssey Offload Processing state machine is MSB first.

<u>Temperature calculation Notes – SiLabs Si7020</u>

C = ([16-bit reading] * 17572 / 6553600) – 47 (Celsius)

F = ([16-bit reading] * 158148 / 32768000) – 53 (Fahrenheit)

Document Revision History

Date	Revision	Changes
February 15, 2016	1.0	Initial Release - for v2.0 Odyssey Firmware
March 9, 2016	1.1	Miscellaneous cleanup, clarifications
March 30, 2016	1.2	New personality descriptions enhanced, usage added