www.parallax.com/P2 ▪ sales@parallax.com ▪ support@parallax.com ▪ +1 888-512-1024

Parallax Propeller 2 (P2X8C4M64P) Datasheet

The Propeller 2 is a multicore microcontroller for embedded systems that delivers high-speed parallel processing with low current consumption in a small package. The Propeller's multiple processors enjoy full command of I/O pins, the flexibility to change clock speed, the power to start and stop at will, and the ability to perform simultaneous tasks in an independent or cooperative manner.

The Propeller 2 P2X8C4M64P microcontroller consists of 8 identical 32-bit processors (called cogs), each with their own RAM, which connect to a common hub. The hub provides 512 KB of shared RAM, a CORDIC math solver, and housekeeping facilities. The architecture supports 64 smart I/O pins, each capable of many autonomous analog and digital functions. The Propeller 2's assembly language (PASM2) features per-instruction conditional execution, special looping mechanisms, and pattern-based instruction skipping to encourage fast, compact code.

Part Number Legend			
P2X	**8C**	**4M**	**64P**
Propeller 2	8 cogs (processors)	4 Mbit Main RAM (512 KB)	64 smart I/O pins

There are three memory regions: Register RAM, Lookup RAM, and Hub RAM. Each cog has its own Register RAM and Lookup RAM (collectively called Cog RAM), while the Hub RAM is shared by all cogs.

Propeller 2 (P2X8C4M64P) RAM Memory Configuration				
Region	**Depth**	**Width**	**Program Counter Address Range (Hex)**	**PASM Instruction D/S Address Range (Hex)**
Cog "Register" RAM	512	32 bits	$00000..$001FF	$000..$1FF
Cog "Lookup" RAM	512	32 bits	$00200..$003FF	$000..$1FF
Hub RAM	524,288	8 bits	$00400..$7FFFF	$00000..$7FFFF

FEATURES

Eight powerful 32-bit processors, each with:

- Access to all I/O pins, plus four fast DAC output channels and four fast ADC input channels
- 512 longs of dual-port Register RAM for code and fast variables
- 512 longs of dual-port Lookup RAM for code, streamer lookup, and variables
- Ability to execute code directly from Register RAM, Lookup RAM, and Hub RAM
- ~350 unique instructions for math, logic, timing, and control operations
- 2-clock execution for all math and logic instructions, including 16 x 16 multiply

- 6-clock custom-bytecode executor for interpreted languages
- Ability to stream Hub RAM and/or Lookup RAM to DACs and pins or HDMI modulator
- Ability to stream pins and/or ADCs to Hub RAM
- Live colorspace conversion using a 3 x 3 matrix with 8-bit signed/unsigned coefficients
- Pixel blending instructions for 8:8:8:8 data
- 16 unique event trackers that can be polled and waited upon
- 3 prioritized interrupts that trigger on selectable events
- Cog-to-cog attention signals for swift coordination
- Hidden debug interrupt for single-stepping, breakpoint, and polling
- 8-level hardware stack for fastest subroutine calls/returns and push/pop operations
- Carry and Zero flag

Central hub serving the processors with:

- 512 KB of contiguous RAM in a 20-bit address space
    - 32-bits-per-clock sequential read/write for all cogs, simultaneously
    - readable and writable as bytes, words, or longs in little-endian format
    - last 16 KB of RAM is write-protectable
- 32-bit, pipelined CORDIC solver with scale-factor correction
    - 32-bit x 32-bit unsigned multiply with 64-bit result
    - 64-bit / 32-bit unsigned divide with 32-bit quotient and 32-bit remainder
    - 64-bit → 32-bit square root
    - Rotate (X32, Y32) by Theta32 → (X32, Y32)
    - (Rho32, Theta32) → (X32, Y32) polar-to-cartesian
    - (X32, Y32) → (Rho32, Theta32) cartesian-to-polar
    - 32 → 5.27 unsigned-to-logarithm
    - 5.27 → 32 logarithm-to-unsigned
    - Cogs can start CORDIC operations every 8 clocks and get results 55 clocks later
- 16 semaphore bits with atomic read-modify-write operations
- 64-bit free-running counter which increments every clock, cleared on reset
- High-quality pseudo-random number generator (Xoroshiro128**), true-random seeded at start-up, updates every clock, provides unique data to each cog and pin
- Mechanisms for starting, polling, and stopping cogs
- 16KB boot ROM
    - Loads into last 16 KB of Hub RAM on boot-up
    - SPI loader for automatic startup from 8-pin flash or SD card
    - Serial loader for startup from host
        - Hex and Base64 download protocols
        - Interactive terminal P2 Monitor
        - Interactive terminal TAQOZ Forth

64 Smart I/O pins, each with:

- 8-bit, 120-ohm (3ns) and 1k-ohm DACs with 16-bit oversampling, noise, and high/low digital modes
- Delta-sigma ADC with 5 ranges, 2 sources, and VIO/GIO calibration
- Several ADC sampling modes: automatic 2n SINC2, adjustable SINC2/SINC3, oscilloscope
- Logic, Schmitt, pin-to-pin-comparator, and 8-bit-level-comparator input modes
- 2/3/5/8-bit-unanimous input filtering with selectable sample rate
- Incorporation of inputs from relative pins, -3 to +3
- Negative or positive local feedback, with or without clocking
- Externally powered in blocks of 4 for clean analog Vdd reference

- Separate drive modes for high and low output: logic / 1.5 k / 15 k / 150 k / 1 mA / 100 µA / 10 µA / float
- Programmable 32-bit clock output, transition output, NCO/duty output
- Triangle/sawtooth/SMPS PWM output, 16-bit frame with 16-bit prescaler
- Quadrature decoding with 32-bit counter, both position and velocity modes
- 16 different 32-bit measurements involving one or two signals
- USB full-speed and low-speed (via odd/even pin pairs)
- Synchronous serial transmit and receive, 1 to 32 bits, up to clock/2 baud rate
- Asynchronous serial transmit and receive, 1 to 32 bits, up to clock/3 baud rate

Six clock modes, all under software control with glitch-free switching between sources:

- Internal 20M Hz+ RC oscillator, nominally 25 MHz, used as initial clock source
- Crystal oscillator with internal loading caps for 7.5 pF/15 pF crystals, can feed PLL
- Clock input, can feed PLL
- Fractional PLL with 1..64 crystal divider --> 1..1024 VCO multiplier --> optional (1..15)*2 VCO post-divider
- Internal ~20 kHz RC oscillator for low-power operation (130 µA)
- Clock can be stopped for lowest power until reset (100 µA, due to leakage)

Power requirements:

- Core: 1.8 VDC, powered via VDD pins
- Smart I/O pins: 3.3 VDC, powered in groups of 4 via VIO pins

Physical characteristics:

- Package type: Exposed-pad TQFP-100
- Dimensions: 14 x 14 mm
- Operating temperature range: AEC-Q100 Level 2 (-40 to +221 °F, -40 to +105 °C)
- Moisture Sensitivity Level (MSL) 3 (168 hours)

Table of Contents

PREFACE

This datasheet provides a condensed description of the concepts, features, and hardware of the Propeller 2 multicore microcontroller. It serves as a feature reference beyond that of the single-page spec sheet.

For additional documentation and resources, including programming tools, visit www.parallax.com/P2. The latest version of this datasheet, along with links to a commentable Google Doc version, are available from the Documentation section. In addition, there are links to more in-depth references for the Propeller 2 and its Spin2 and PASM2 languages, which may include commentable Google Docs.

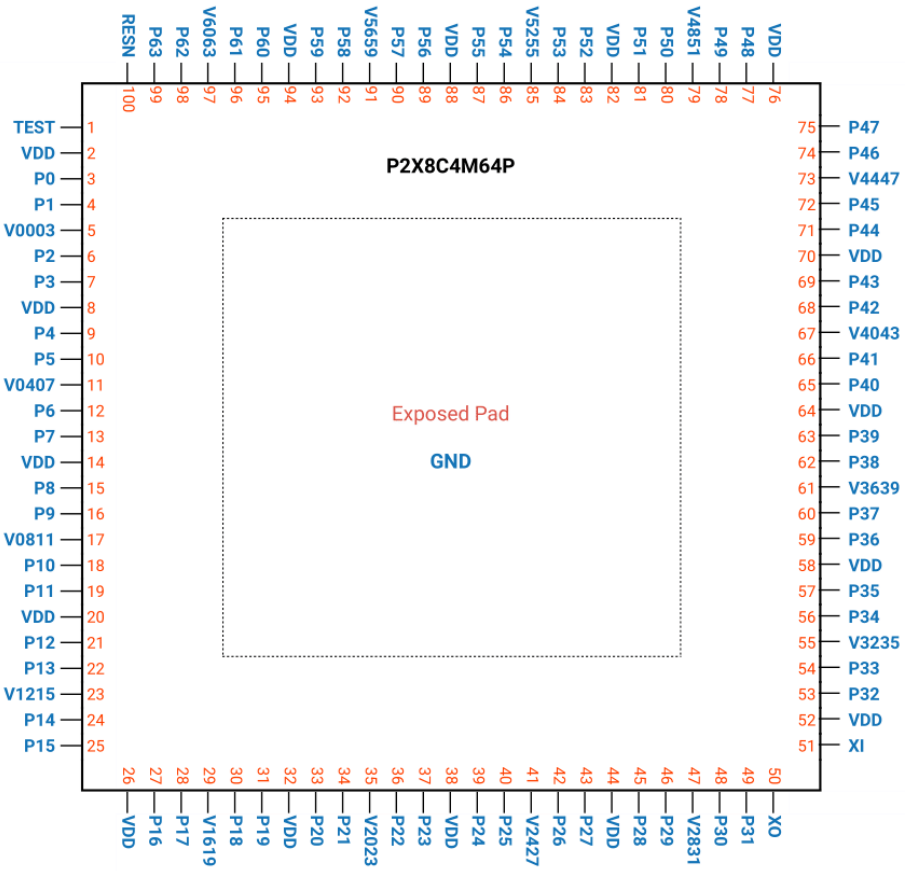
HARDWARE

The Propeller 2 microcontroller's pin layout and simplified connections are described here for conceptual reference. For most prototyping uses, Parallax recommends a pre-made Propeller 2 circuit, like the P2 Edge Module (#P2-EC) or Propeller 2 Evaluation Board (#64000) that includes all the recommended connections and layout design rules. Visit the Propeller 2 section of the Parallax online store for chips, evaluation boards, accessories, and developer starter kit bundles.

Pin Descriptions

The Propeller 2 (P2X8C4M64P) features notable power and thermal considerations in its pinout:

- A large exposed under-side pad serves both as a ground reference (GND) for the core and I/O pins as well as a heatsink to cool the silicon
- Multiple power pins (VDD) spaced regularly around the chip allow clean core power references across the entire chip
- Multiple I/O power pins (Vxxyy) spaced regularly around the chip provide flexibility for the isolated, ultra-stable power references required for clean DAC and ADC I/O operations

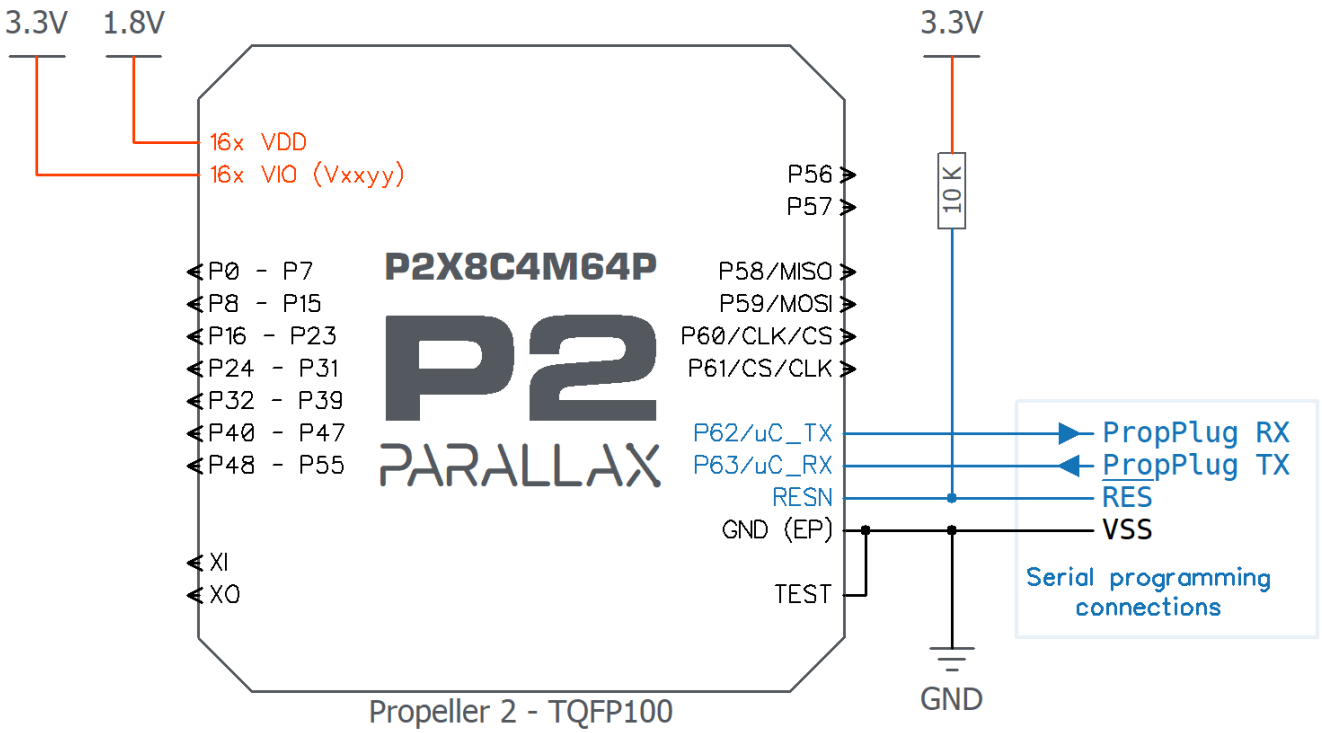


Pin Descriptions			
Pin Name	**Direction**	**V (typ)**	**Description**
GND	-	0	Exposed Pad (underside of chip); ground for core and smart pins – internally connected to exposed pad. Connect to ground plane for thermal dissipation.
TEST	I	0	Tied to ground
VDD	-	1.8	Core power
P0-63	I/O	0 to 3.3	Smart pins. P58-P63 serve in the boot process, then general purpose after.
Vxxyy	-	3.3	Power for smart pins in groups of 4: Pxx through Pyy
XO	O	-	Crystal Output. Provides feedback for an external crystal, or may be left disconnected depending on CLK Register settings. No external resistors or capacitors are required.
XI	I	-	Crystal Input. Can be connected to the output of crystal/oscillator pack (with XO left disconnected), or to one leg of crystal (with XO connected to the other leg of crystal or resonator) depending on CLK Register settings. No external resistors or capacitors are required.
RESN	I	0	Reset (active low). When low, resets the Propeller: all cogs disabled and I/O pins floating. Propeller restarts 3 ms after RESn transitions from low to high. Connect to a resistor to pull up to 3.3 V.

Hardware Connections

Minimal Connections

The Propeller 2 is programmed via four wires and may optionally include an external crystal, reset switch, SPI Flash and/or microSD memory.



Minimal Propeller 2 Connections

- All VDD pins must be connected to a single 1.8 V supply.
- All Vxxyy pins must be connected to 3.3 V. The Vxxyy pins can share a common 3.3 V supply, or be split across multiple supplies. Typically those Vxxyy pins powering analog and digital functions would have different supplies.
- All VDD and all Vxxyy pins must have closely-located bypass caps to GND (not shown).
- The common GND (EP) pad under the chip is also used for thermal dissipation. It is recommended to connect the GND pad to a solid ground plane with multiple vias.
- TEST pin must always be connected to GND.
- RESN (reset) pin must always have a pullup resistor to 3.3 V (typically 10 KΩ).
- Programming and debugging is achieved with a serial interface, such as the Parallax PropPlug #32201.
- Minimal connections assume no external clock source connected to XI/XO– the internal clock must be used (*adjust source code to match*). Refer to the AC Characteristics table for further information.

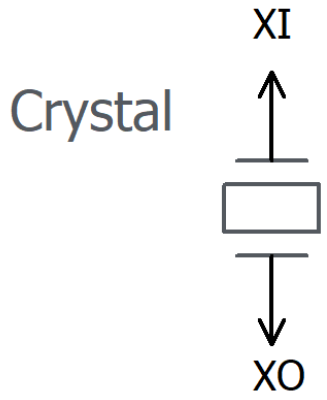
External Crystal

The internal clock reference is good for very low power applications, but is also fairly low accuracy.

For applications that require higher accuracy (ex: handling high speed data) an external clock source is connected to XI/XO.

Typically a crystal would be connected between XI and XO, but an external clock source could also be connected to XI only, with common options including a clock source generator, oscillator, or MEMS resonator.

Refer to the AC Characteristics table for further information.

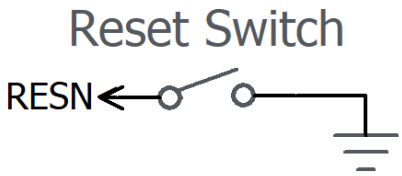


Reset Switch

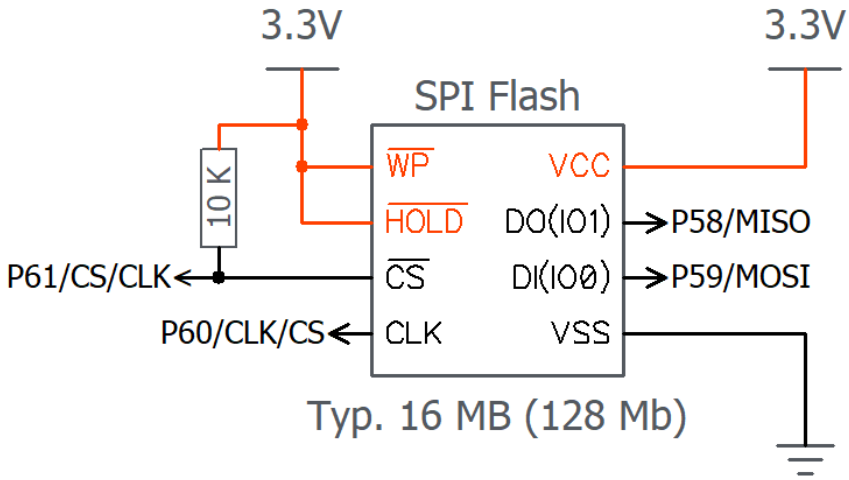
Reset Switch can be optionally included and is a convenient way to restart the Propeller 2 during development or in a final product.

The switch drives the Propeller 2 reset pin (RESN) to ground, and while held low, the Propeller 2 remains in a dormant, low-power state.

Note that the RESN pin must always be pulled high (to 3.3 V) with an external resistor, which is shown in the Minimal Connections diagram.



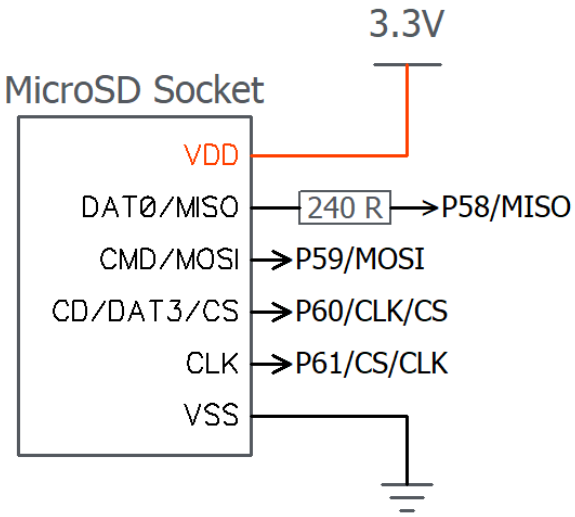
SPI Flash Boot Memory



SPI Flash Boot Memory Connections

- Refer to all requirements and recommendations for Minimal Connections
- When the Propeller 2 starts up (or is reset) with this circuit, there will be a serial programming window of 100 ms, then automatic boot from SPI flash. If SPI flash boot fails, then a further serial window of 60 seconds will be followed by shutdown.
- Each Propeller 2 firmware image requires up to 512 KB. A single SPI Flash chip could hold multiple firmware images or code snippets, and/or be used for user data. The SPI Flash chip is fully available to the user program as a general memory area.

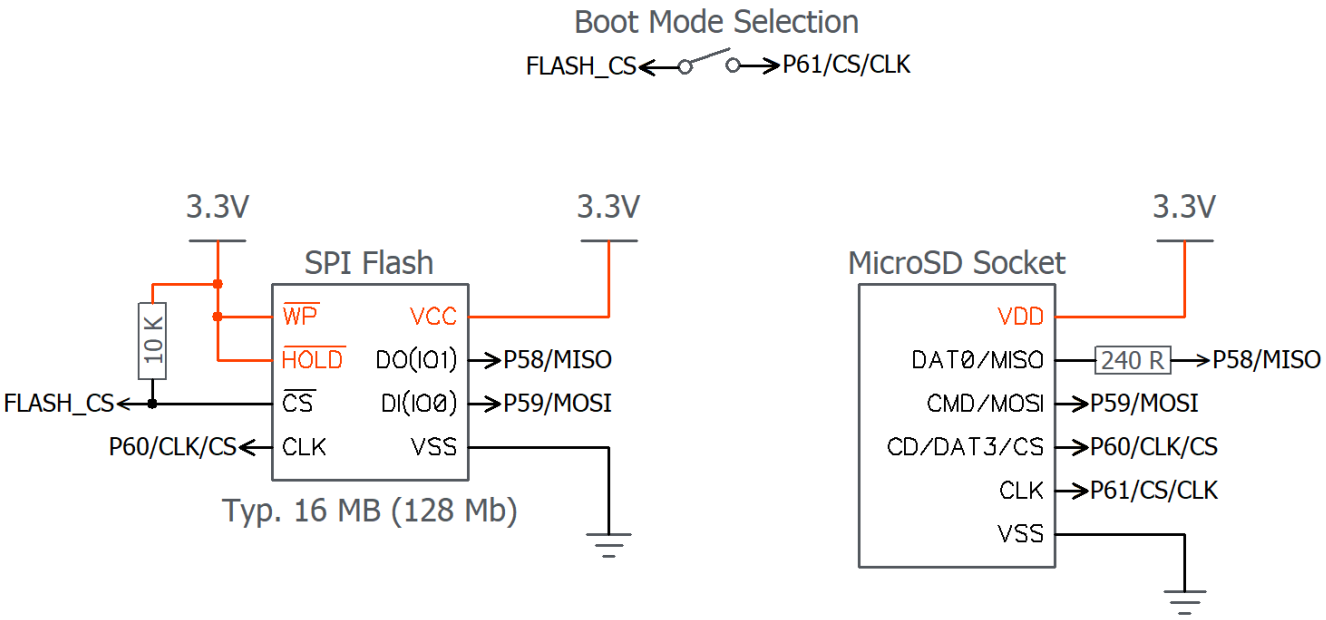
MicroSD Boot Memory



MicroSD Boot Memory Connections

- Refer to all requirements and recommendations for Minimal Connections
- When the Propeller 2 starts up (or is reset) with this circuit, it will automatically boot from firmware saved on the microSD card. If microSD boot fails, a further serial window of 60 seconds will be followed by shutdown.
- The microSD card must be formatted as FAT32, and the boot firmware file must be saved in the root of the microSD card with the special filename: `_P2_BOOT.BIX`
- Each Propeller 2 firmware image requires up to 512 KB. A single microSD card could hold multiple firmware images or code snippets, and/or be used for user data. The microSD card is fully available to the user program as a general memory area.

Dual Boot Memory



Dual Boot Memory Connections

- Refer to all requirements and recommendations for Minimal Connections
- The Boot Mode Selection switch determines the active boot device; either SPI Flash or microSD.
- Switch closed (on) = SPI Flash boot mode : When the Propeller 2 starts up (or is reset) with this circuit, there will be a serial programming window of 100 ms, then automatic boot from firmware stored in the SPI flash memory. If SPI flash boot fails, a further serial window of 60 seconds will be followed by shutdown.
- Switch open (off) = microSD boot mode : When the Propeller 2 starts up (or is reset) with this circuit, it will automatically boot from firmware saved on the microSD card. If microSD boot fails then a further serial window of 60 seconds will be followed by shutdown.

OPERATION

Assuming a stable power supply and "high" RESn pin, the Propeller 2 boots up using the following procedure:

1. Within 5 ms, the Propeller 2 loads its Bootloader into cog 0.
2. The Bootloader checks the Boot Pattern (configuration) on pins P59-P61 and either communicates with a host over serial, or loads (boots) an application from a connected flash chip or SD card.
3. All further activity is defined by the application itself. The application, and thus the developer, has complete control over modifying internal clock speed, I/O pin usage and behavior, what cogs are running at any given time, and more.
4. The Propeller continues running until all cogs shut each other (or themselves) down, the RESn pin goes low, or the application requests a reboot.

HOST COMMUNICATION

In typical operation (above), the Propeller 2 will boot up with a user's pre-written Propeller application; however, the same process also allows for loading new applications or interacting with the built-in systems. Most boot patterns (pins P59-P61) trigger a serial communication window in which a host computer can talk with the Propeller 2 serially over pins P62 and P63.

To enter interactive mode from a host computer:

- Run serial terminal software (like Parallax Serial Terminal, TeraTerm, or RealTerm)
- Disable character echo ("Echo On" in Parallax Serial Terminal)
- Set to any baud rate from 9600 Bd to 2 MBd (recommended), 8 data bits, 1 stop bit, no parity
- Press and release the Propeller 2 development board's Reset button
- Type "> " (greater than followed by a space), then either Ctrl+D or the ESC key to enter P2 Monitor or TAQOZ mode, respectively

P2 Monitor

The P2 Monitor is a built-in interactive system that allows for viewing and manipulating memory and running code. Use the P2 Monitor to explore and change current RAM contents or load and run code from microSD memory. After power-up or reset (and while preventing autorun of a flash/microSD-resident application), invoke the P2 Monitor from a terminal by typing: "> " (greater than followed by a space), then Ctrl+D.

Here is an example of listing the first 16 longs of Register RAM (in long format), by typing "`000-010L`":

```
*000-010L
  000: FF800800 FC0C003F F606C832 FCDC041F  '.......?...2....'
  004: FD747E40 F0A6CA01 F426CA1F FD62CA00  '.t~@.....&...b..'
  008: FB6EC9FA FC0C003F FD64C428 FF0007E0  '.n.....?.d.(....'
  00C: FB06012C FD655229 FF0007E1 FB0420B8  '...,.eR)...... .'
```

Here is a list of the first 16 bytes of Hub RAM (in long format), typing "`0000-0010L`":

```
*0000-0010L
00000: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  '................'
```

For more information, see the P2 Monitor link on the Propeller 2 Documentation Page at www.parallax.com/p2.

To switch to TAQOZ while in P2 Monitor, type ESC followed by the Enter key.

TAQOZ

TAQOZ is a built-in interactive Forth language engine, based on Tachyon Forth. Use TAQOZ to explore "what ifs" and quickly exercise P2 hardware for testing or debugging. After power-up or reset (and while preventing autorun of a flash/microSD-resident application), invoke TAQOZ from a terminal by typing: "> " ESC (greater than followed by a space), then the Escape key.

Toggle pin 56 (ex: blink an LED on P56) by typing:

```
56 blink                                                    (type "56 mute" to stop toggling)
```

...or by typing:

```
begin 56 high 250 ms 56 low 250 ms key until    (press any key to stop toggling)
```

For more information, see the TAQOZ links on the Propeller 2 Documentation Page at www.parallax.com/p2.

To switch to P2 Monitor while in TAQOZ, type Ctrl+D.

SYSTEM ORGANIZATION

The Propeller 2 includes the following subsystems.

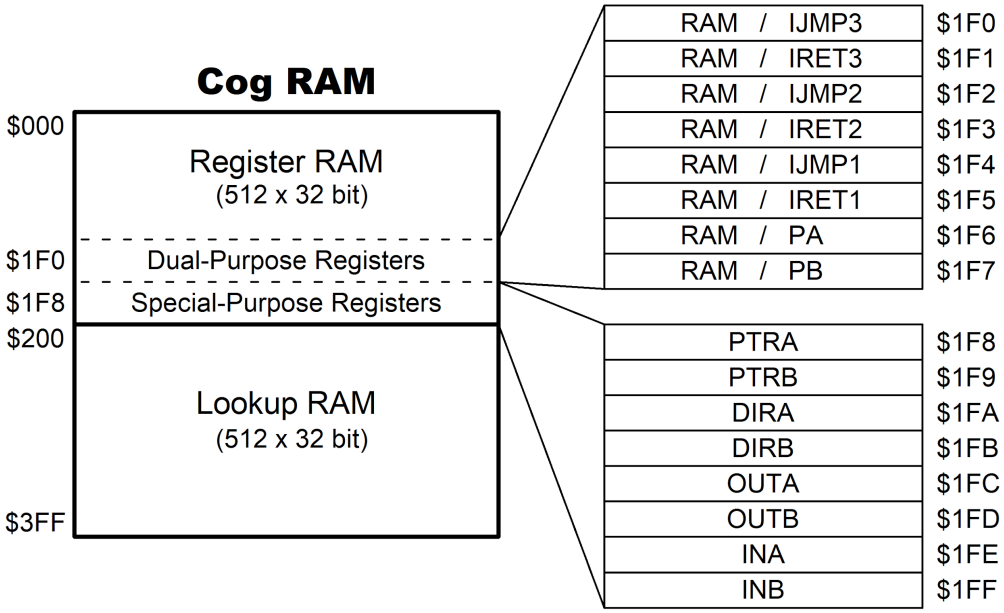
- Cogs (processors) - independent 32-bit processing units
  - Register RAM - private memory for cog to execute code and swiftly manipulate data
  - Lookup RAM - semi-private memory for cog to execute code and manipulate data tables/streams
- Hub - access manager for exclusive shared resources
  - Hub RAM - shared memory for all cogs to execute code and manipulate data
  - System Clock - clock source for all internal components
  - Locks - 16 semaphore bits to coordinate exclusive access of shared resources
  - CORDIC Solver - pipelined calculator for math functions
- Smart I/O Pins - I/O pins with optional Smart circuit for autonomous functions

Cogs

The Propeller contains multiple processors, called cogs. Each cog has its own RAM and can start, stop, and execute instructions independently of one another. All active cogs share the same System Clock, Hub RAM, and I/O pins.

Cog RAM

Each cog's RAM is made of two blocks of 512 longs (512 x 32), called Register RAM and Lookup RAM, organized as shown here.



Register RAM

Each cog's primary 512 x 32-bit dual-port Register RAM (Reg RAM for short) provides for code execution, fast direct register access, and special use. It is read and written as longs (4 bytes) and contains general purpose, dual-purpose, and special-purpose registers.

General Purpose Registers

RAM registers $000 through $1EF are general-purpose registers for code and data usage.

Dual-purpose Registers

RAM registers $1F0 through $1F7 may either be used as general-purpose registers, or may be used as special-purpose registers if their associated functions are enabled.

Address	Name	Purpose
$1F0	RAM / IJMP3	Interrupt call address for INT3
$1F1	RAM / IRET3	Interrupt return address for INT3
$1F2	RAM / IJMP2	Interrupt call address for INT2
$1F3	RAM / IRET2	Interrupt return address for INT2
$1F4	RAM / IJMP1	Interrupt call address for INT1
$1F5	RAM / IRET1	Interrupt return address for INT1
$1F6	RAM / PA	CALLD-imm return, CALLPA parameter, or LOC address
$1F7	RAM / PB	CALLD-imm return, CALLPB parameter, or LOC address

Special-purpose Registers

RAM registers $1F8 through $1FF give mapped access to eight special-purpose functions. In general, when specifying an address between $1F8 and $1FF, the PASM instruction accesses a special-purpose register, *not* just the underlying RAM.

Address	Name	Purpose
$1F8	PTRA	Pointer A to Hub RAM
$1F9	PTRB	Pointer B to Hub RAM
$1FA	DIRA	Output enables for P31..P0
$1FB	DIRB	Output enables for P63..P32
$1FC	OUTA	Output states for P31..P0
$1FD	OUTB	Output states for P63..P32
$1FE	INA[1]	Input states for P31..P0
$1FF	INB[2]	Input states for P63..P32

[1] Also debug interrupt call address

[2] Also debug interrupt return address

Lookup RAM

Each cog's secondary 512 x 32-bit dual-port Lookup RAM (LUT RAM for short) is read and written as longs (4 bytes). It is useful for:

- Scratch space
- Streamer access
- Bytecode execution lookup table
- Smart pin data source
- Paired-Cog communication mechanism
- Code execution

Scratch Space

In contrast to Register RAM, the cog cannot directly reference Lookup RAM locations in the majority of its PASM instructions. Instead, the desired location(s) must be read or written between Lookup RAM and Register RAM using the RDLUT and WRLUT instructions, respectively. This is synonymous with other hardware architecture's scratch storage using "LOAD" and "STORE" instructions. When using the RDLUT and WRLUT instructions, the Lookup RAM's locations $200..$3FF are addressable as $000..$1FF.

Paired-Cog Communication Mechanism

Adjacent cogs whose ID numbers differ by only the LSB (cogs 0 and 1, 2 and 3, etc.) can allow their Lookup RAMs to be written by the other cog via its local Lookup RAM writes. This allows adjacent cogs to share data quickly through their Lookup RAMs. Use the SETLUTS instruction to enable/disable this feature and SETSE1..4 to facilitate handshaking if necessary. Note that this adjacent cog access is implemented on the Lookup RAM's 2nd port, which is also used by the streamer in DDS/LUT modes; these are not intended to be used simultaneously.

Execution

Cogs employ a five-stage pipelined execution architecture. When the execution pipeline is full, each PASM2 instruction effectively takes as little as two clock cycles to execute. If an instruction stalls for additional clock cycles, all following instructions in the pipeline are also stalled. Any instruction that is conditionally cancelled will still move through the pipeline without stalling or executing. Branch instructions cause the pipeline to be flushed, so the first instruction following the branch will take at least five clock cycles.

Cogs use 20-bit addresses for their program counters (PC); the upper bit is a "don't care" bit - this affords an execution space of up to 512 KB. Depending on the value of a cog's PC, an instruction will be fetched from either its Register RAM, its Lookup RAM, or the Hub RAM.

PASM2 Execution Regions			
PC Address	**Instruction Source**	**Memory Width**	**PC Increment**
$00000..$001FF	Cog Register RAM	32 bits	1
$00200..$003FF	Cog Lookup RAM	32 bits	1
$00400..$7FFFF	Hub RAM	8 bits	4

Register Execution
When the PC is in the range of $00000 to $001FF, the cog fetches instructions from Cog Register RAM. This is referred to as "cog execution." There are no special considerations when branching to a cog register address.

Lookup Execution
When the PC is in the range of $00200 to $003FF, the cog fetches instructions from Cog Lookup RAM. This is referred to as "lut execution." There are no special considerations when branching to a cog lookup address.

Hub Execution
When the PC is in the range of $00400 to $7FFFF, the cog fetches instructions from Hub RAM. This is referred to as "hub execution mode." Special considerations are involved with hub execution.

1. The PC rolling beyond $003FF will not initiate hub execution (it will just wrap back to $00000); a branch must occur to get from register or lookup execution to hub execution.
2. Branching to a hub address takes a minimum of 13 clock cycles. If the instruction being branched to is not long-aligned, one additional clock cycle is required.
3. When executing from Hub RAM, the cog employs the FIFO hardware to spool up instructions so that a stream of instructions will be available for continuous execution. This means the FIFO cannot be used for anything else. So, during hub execution these instructions cannot be used:

   `RDFAST` / `WRFAST` / `FBLOCK`
   `RFBYTE` / `RFWORD` / `RFLONG` / `RFVAR` / `RFVARS`
   `WFBYTE` / `WFWORD` / `WFLONG`
   `XINIT` / `XZERO` / `XCONT` - when the streamer mode engages the FIFO

It is not possible to execute code from hub addresses $00000 through $003FF, as the cog will instead read instructions from the cog's Register RAM or Lookup RAM as indicated above.

Starting and Stopping Cogs

Any cog can start or stop any other cog, or restart or stop itself. Each cog has a unique ID which can be used to start or stop it. It's also possible to start free (stopped or never started) cogs, without needing to know their IDs. This way, applications can simply start free cogs, as needed, and as those cogs retire by stopping themselves or getting stopped by others, they return to the pool of free cogs to become available again for restarting.

PASM2 code can ID its own cog (or get the running status of other cogs) with the COGID instruction, can start a cog with COGINIT, and can stop a cog with COGSTOP. Using a SETQ instruction before a COGINIT instruction sets the target cog's PTRA register to a 32-bit value; useful for pointing the new cog to runtime data or delivering a single startup value.

Cog Attention

Each cog can request the attention of other cogs by using the **COGATN** instruction. One or more of the D operand's lower 8 bits may be set high (1) to signal the corresponding cog or cogs. For each high bit, the matching cog sees an 'attention' event for **POLLATN** / **WAITATN** / **JATN** / **JNATN** and interrupt use. The attention strobe outputs from all cogs are OR'd together to form a composite set of 8 strobes from which each cog receives its particular strobe.

In the intended use case, the cog receiving an attention request knows which other cog is strobing it and how to respond. In cases where multiple cogs may request the attention of a single cog, some messaging structure may need to be implemented in Hub RAM to differentiate requests.

Hub

Hub RAM

The globally-accessible Hub RAM can be read and written as bytes, words, and longs, in little-endian format. Hub addresses are always byte-oriented. There are no special alignment rules for words and longs in Hub RAM. Cogs can read and write bytes, words, and longs starting at any hub address, as well as execute PASM2 instructions (longs) from any hub address starting at $400.

The last 16 KB of Hub RAM is normally addressable at both its normal address range, as well as at $FC000..$FFFFF. This provides a stable address space (regardless of future Propeller 2 variations) for the 16 KB of internal ROM which gets cached into the last 16 KB of Hub RAM on startup. This upper 16 KB mapping is also used by the cog debugging scheme.

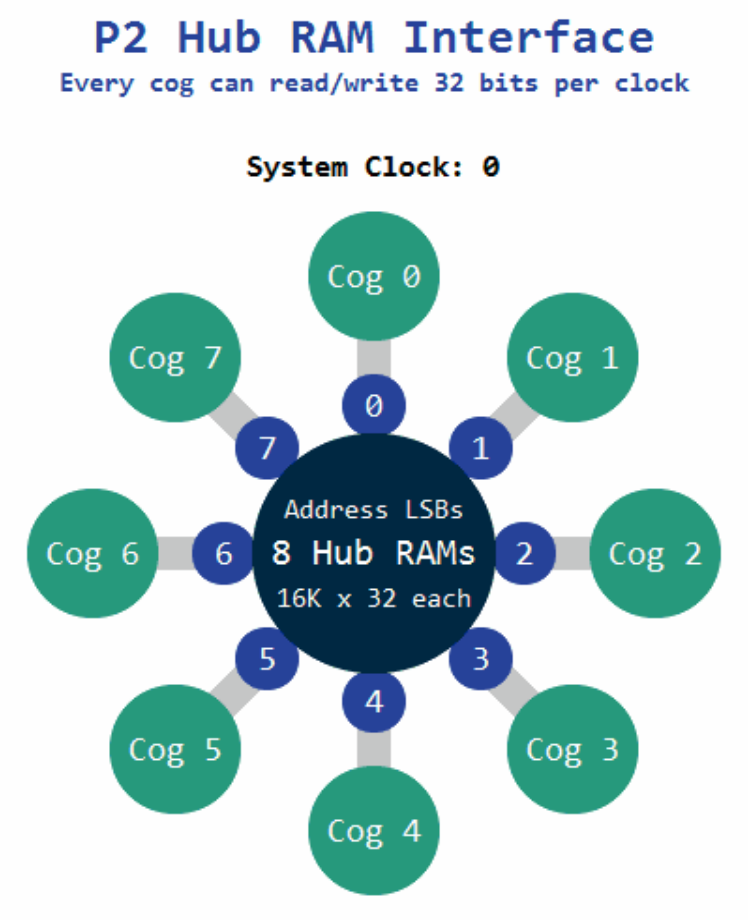
The last 16 KB of RAM can be hidden from its normal address range and made read-only at $FC000..$FFFFF. This is useful for making the last 16 KB of RAM persistent, like ROM. It is also how debugging is realized, as the RAM mapped to $FC000..$FFFFF can still be written to while executing code from within debug interrupt service routines, permitting the otherwise-protected RAM to be used as debugger-application space and cog-register swap buffers for debug interrupts.

Cog-to-Hub RAM Interface

Hub RAM consists of 32-bit-wide single-port RAMs with byte-level write controls. This RAM is split into slices (one per cog) that are multiplexed among all cogs. On the Propeller 2 (P2X8C4M64P), each RAM slice holds every 8th long in the composite Hub RAM. Upon every clock cycle, each cog can access the "next" RAM slice, allowing for continuous bidirectional streaming of sequential Hub RAM longs. The Hub RAM Interface diagram illustrates this process conceptually as the collective of RAM slices rotates around, each facing a new cog every clock cycle.

When a cog wants to read or write the Hub RAM, it must wait up to #cogs-1 clocks to access the initial RAM slice of interest. Once that occurs, subsequent locations (slices) can be accessed on every clock, thereafter, for continuous reading or writing of 32-bit longs.

Normally, if the cog chooses not to access the next available location upon the next clock, it must once again wait up to 7 clocks to re-align with the desired slice. However, each cog has an optional hub FIFO interface that smooths out data flow for less than 32-bits-per-clock access. This hub FIFO interface can be set for hub-RAM-read or hub-RAM-write operation to allow Hub RAM to be either sequentially read or sequentially written in any combination of bytes, words, or longs, at any rate, up to one long per clock. Regardless of the transfer frequency or the word size, the FIFO will ensure that the cog's reads or writes are all properly conducted from/to the composite Hub RAM.



Cogs can access Hub RAM either via the sequential FIFO interface, or by waiting for RAM slices of interest, while yielding to the FIFO. If the FIFO is not busy (which is soon the case if data is not being read from or written to it) random accesses will have full opportunity to access the composite Hub RAM.

There are three ways the hub FIFO interface can be used, and it can only be used for one of these at a time:

- Hub execution (when the PC is $00400..$FFFFF)
- Streamer usage (background transfers from Hub RAM → pins/DACs, or from pins/ADCs → Hub RAM)
- Software usage (fast sequential-reading or sequential-writing instructions)

For streamer or software usage, FIFO operation must be established by a RDFAST or WRFAST instruction executed from Cog RAM (Register/Lookup, $00000..$003FF). After that, and while remaining in Cog RAM, the streamer can be enabled to begin moving data in the background, or the two-clock RFxxxx/WFxxxx instructions can be used to manually read and write sequential data.

The FIFO contains (#cogs+11) stages. When in read mode, the FIFO loads continuously whenever less than (#cogs+7) stages are filled, after which point, up to 5 more longs may stream in, potentially filling all stages. These metrics ensure that the FIFO never underflows, under all potential reading scenarios.

System Clock

The system clock is the time base for all internal components and can be configured in several ways.

- Direct from internal slow clock (RCSLOW); a ~20 kHz oscillator is intended for low-power operation
- Direct from internal fast clock (RCFAST); a 20 MHz+ oscillator designed for minimum 20 MHz operation
- Direct from XI pin; driven externally via a clock oscillator or a crystal oscillator
- PLL-modified XI pin; driven externally via a clock oscillator or a crystal oscillator and the signal internally modified by the PLL (phase-locked loop), usually to multiple to a much higher frequency

The system clock is configured by the running Propeller 2 application using the `HUBSET` instruction in this format:

```
HUBSET  ##%0000_000E_DDDD_DDMM_MMMM_MMMM_PPPP_CCSS     'set clock mode
```

The bit fields (E, D, M, P, C, and S) are described in the following tables.

PLL Setting	Value	Effect	Notes
%E	0/1	PLL off/on	XI input must be enabled by %CC. Allow 10ms for crystal+PLL to stabilize before switching over to PLL clock source.
%DDDDDD	0..63	1..64 division of XI pin frequency	This divided XI frequency feeds into the phase-frequency comparator's 'reference' input.
%MMMMMMMMMM	0..1023	1..1024 division of VCO frequency	This divided VCO frequency feeds into the phase-frequency comparator's 'feedback' input. This frequency division has the effect of *multiplying* the divided XI frequency (per %DDDDDD) inside the VCO. The VCO frequency should be kept within 100MHz to 350MHz.
%PPPP	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	VCO / 2 VCO / 4 VCO / 6 VCO / 8 VCO / 10 VCO / 12 VCO / 14 VCO / 16 VCO / 18 VCO / 20 VCO / 22 VCO / 24 VCO / 26 VCO / 28 VCO / 30 VCO / 1	This divided VCO frequency is selectable as the system clock when SS = %11.

%CC	XI status	XO status	XI / XO impedance	XI / XO loading caps
%00	ignored	float	Hi-Z	OFF
%01	input	600-ohm drive	1M-ohm	OFF
%10	input	600-ohm drive	1M-ohm	15pF per pin
%11	input	600-ohm drive	1M-ohm	30pF per pin

%SS	Clock Source	Notes
%11	PLL	CC != %00 and E=1, allow 10ms for crystal+PLL to stabilize before switching to PLL
%10	XI	CC != %00, allow 5ms for crystal to stabilize before switching to XI pin
%01	RCSLOW	~20 kHz, can be switched to at any time, low-power
%00	RCFAST	20 MHz+, can be switched to at any time, used on boot-up.

WARNING: Incorrectly switching away from the PLL setting (%SS = %11) can cause a glitch which will hang the clock circuit. In order to safely switch, always start by switching to an internal oscillator using either HUBSET #$F0 (for RCFAST) or HUBSET #$F1 (for RCSLOW).

PLL Example

The PLL divides the XI pin frequency from 1 to 64, then multiplies the resulting frequency from 1 to 1024 in the VCO. The VCO frequency can be used directly, or divided by 2, 4, 6, ...30, to get the final PLL clock frequency which can be used as the system clock.

The PLL's VCO is designed to run between 100 MHz and 200 MHz and should be kept within that range.

$$VCO = \frac{Freq(XI) \times (\%MMMMMMMMMM + 1)}{(\%DDDDDD + 1)}$$
$$PLL = if(\%PPPP = 15) \Rightarrow VCO$$
$$PLL = if(\%PPPP \neq 15) \Rightarrow \frac{VCO}{(\%PPPP + 1) \times 2}$$

Let's say you have a 20 MHz crystal attached to XI and XO and you want to run the Prop2 at 148.5 MHz. You could divide the crystal by 40 (%DDDDDD = 39) to get a 500 kHz reference, then multiply that by 297 (%MMMMMMMMMM = 296) in the VCO to get 148.5 MHz. You would set %PPPP to %1111 to use the VCO output directly. The configuration value would be %1_100111_0100101000_1111_10_11. The last two 2-bit fields select 15 pf crystal mode and the PLL. In order to realize this clock setting, though, it must be done over a few steps:

```
HUBSET  #$F0                                'set 20 MHz+ (RCFAST) mode
HUBSET  ##%1_100111_0100101000_1111_10_00   'enable crystal+PLL, stay in RCFAST mode
WAITX   ##20_000_000/100                    'wait ~10ms for crystal+PLL to stabilize
HUBSET  ##%1_100111_0100101000_1111_10_11   'now switch to PLL running at 148.5 MHz
```

The clock selector controlled by the %SS bits has a deglitching circuit which waits for a positive edge on the old clock source before disengaging, holding its output high, and then waiting for a positive edge on the new clock source before switching over to it. It is necessary to select mode %00 or %01 while waiting for the crystal and/or PLL to settle into operation, before switching over to either.

Locks

For application-defined cog coordination, the hub provides a pool of 16 semaphore bits, called locks. Cogs may use locks, for example, to manage exclusive access of a resource or to represent an exclusive state, shared among multiple cogs. What a lock represents is completely up to the application using it; they are a means of allowing one cog at a time the exclusive status of 'owner' of a particular lock ID. In order to be useful, all participant cogs must agree on a lock's ID and what purpose it serves.

The LOCK instructions are:

```
LOCKNEW    D {WC}
LOCKRET {#}D
LOCKTRY {#}D {WC}
LOCKREL {#}D {WC}
```

Lock Usage

In order to use a lock, one cog must first allocate a lock with LOCKNEW and communicate that lock's ID with other cooperative cogs. Cooperative cogs then use LOCKTRY and LOCKREL to respectively take or release ownership of the state which that lock represents. If the lock is no longer needed by the application, it may be returned to the unallocated lock pool by executing LOCKRET. A cog may allocate more than one lock.

At any time, a cog may attempt to own a lock (ie: the state that lock represents) by using LOCKTRY. The Hub grants or denies ownership in response, ensuring that, at most, one cog owns the lock at any time. If a cog is granted ownership, it can perform the task defined for that lock and then use LOCKRET to release ownership, allowing any other cog to attempt ownership. Only the cog that has taken ownership of the lock can release it; however, a lock will also be implicitly released if the owner cog is stopped (COGSTOP) or restarted (COGINIT).

CORDIC Solver

The Hub contains a 54-stage pipelined CORDIC solver (Coordinate Rotation Digital Computer) that can compute the following functions for all cogs:

- 32 x 32 unsigned multiply with 64-bit product
- 64 / 32 unsigned divide with 32-bit quotient and 32-bit remainder
- Square root of 64-bit unsigned value with 32-bit result
- 32-bit signed (X, Y) rotation around (0, 0) by a 32-bit angle with 32-bit signed (X, Y) results
- 32-bit signed (X, Y) to 32-bit (length, angle) cartesian to polar operation
- 32-bit (length, angle) to 32-bit signed (X, Y) polar to cartesian operation
- 32-bit unsigned integer to 5:27-bit logarithm
- 5:27-bit logarithm to 32-bit unsigned integer

Each cog can issue one CORDIC instruction per its hub access window (which occurs once every eight clocks) and retrieve the result 55 clocks later via the GETQX and GETQY instructions. For faster throughput, cogs can take advantage of the hub access window and CORDIC pipeline relationship to issue a stream of CORDIC instructions interleaved with retrieving corresponding results, achieving up to one CORDIC result every eight clocks. Each cog's active CORDIC instructions and forthcoming results are completely isolated from each other, as well as from other cogs; however, each result must be retrieved on time else it will be overwritten by the following result, if any.

Multiply

Use the QMUL instruction to multiply two unsigned 32-bit numbers together and retrieve the CORDIC result with the GETQX and GETQY instructions (for lower and upper long, respectively).

Divide

Use the QDIV or QFRAC instruction (either with optional preceding SETQ instruction) to divide a 64-bit numerator by a 32-bit denominator, then retrieve the CORDIC results with the GETQX and GETQY instructions (for quotient and remainder, respectively).

Square Root
Use the QSQRT instruction on a 64-bit number and retrieve the square root CORDIC result with the GETQX instruction.

(X, Y) Rotation
Use the SETQ instruction followed by the QROTATE instruction to rotate a 32-bit signed Y and X point pair by an unsigned 32-bit angle and retrieve the CORDIC results with the GETQX and GETQY instructions for X and Y, respectively.

(X, Y) to (length, angle)
Use the QVECTOR instruction to convert a (X, Y) cartesian coordinate into (length, angle) polar coordinate and retrieve the CORDIC results with the GETQX and GETQY instructions (for length and angle, respectively).

(length, angle) to (X, Y)
Use the QROTATE instruction to convert a (length, angle) polar coordinate into (X, Y) cartesian coordinate and retrieve the CORDIC results with the GETQX and GETQY instructions (for X and Y, respectively).

Logarithm
Use the QLOG instruction on an unsigned 32-bit integer and retrieve the 5:27-bit logarithm CORDIC result (5-bit exponent and 27-bit mantissa) with the GETQX instruction.

Exponent
Use the QEXP instruction on a 5:27-bit logarithm and retrieve the unsigned 32-bit integer CORDIC result with the GETQX instruction.

Smart I/O Pins

Every I/O pin features versatile digital and analog capabilities as well as autonomous state machine functions that would otherwise require processor time to perform. The combination of *pin* modes and *smart* modes provides adept functionality for application design, increasing the Propeller 2 potential beyond what multicore architecture alone provides. There are 24 low-level pin modes and 34 high-level smart modes.

Each I/O pin's behavior is described by the combination of four settings: 1) direction (input/output), 2) state (output drive / input sense), 3) pin mode, and 4) smart mode (optional).

Direction and State

In simplest form, I/O pins are controlled via dedicated cog registers and the instructions that affect them.

I/O Pin Registers		
Register	**Cog Address**	**Purpose**
`DIRA`	$1FA	Output enable bits for P0..P31 (active high)
`DIRB`	$1FB	Output enable bits for P32..P63 (active high)
`OUTA`	$1FC	Output state bits for P0..P31 (corresponding DIRA bit must be high to enable output)
`OUTB`	$1FD	Output state bits for P32..P63 (corresponding DIRB bit must be high to enable output)
`INA`	$1FE	Input state bits for P0..P31
`INB`	$1FF	Input state bits for P32..P63

General-purpose and special pin instructions can write to `DIRA` / `DIRB` / `OUTA` / `OUTB` to affect pin input/output behavior and can read from `INA` / `INB` to retrieve pin states. General-purpose instructions operate on the entire 32-bit register (all pins) while the special pin instructions operate on a single bit (pin) within them.

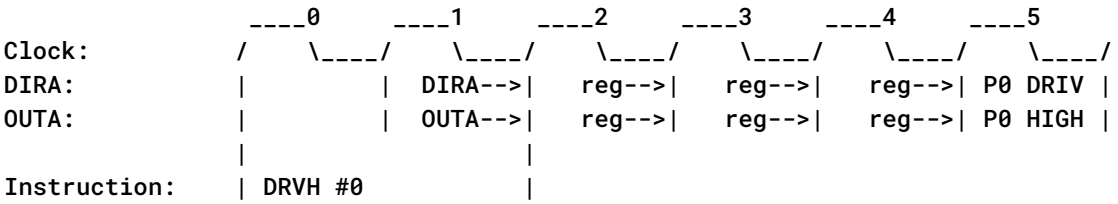
Special Pin Instructions	
Instructions	**Purpose**
DIRL/DIRH/DIRC/DIRNC/DIRZ/DIRNZ/DIRRND/DIRNOT {#}D	Affect pin D bit in DIRx
OUTL/OUTH/OUTC/OUTNC/OUTZ/OUTNZ/OUTRND/OUTNOT {#}D	Affect pin D bit in OUTx
FLTL/FLTH/FLTC/FLTNC/FLTZ/FLTNZ/FLTRND/FLTNOT {#}D	Affect pin D bit in OUTx, clear bit in DIRx
DRVL/DRVH/DRVC/DRVNC/DRVZ/DRVNZ/DRVRND/DRVNOT {#}D	Affect pin D bit in OUTx, set bit in DIRx
TESTP {#}D WC/WZ/ANDC/ANDZ/ORC/ORZ/XORC/XORZ	Read pin D bit in INx and affect C or Z
TESTPN {#}D WC/WZ/ANDC/ANDZ/ORC/ORZ/XORC/XORZ	Read pin D bit in !INx and affect C or Z

The selected pin mode and smart mode (if other than the default) may override some of the above, as described in their respective sections, later.

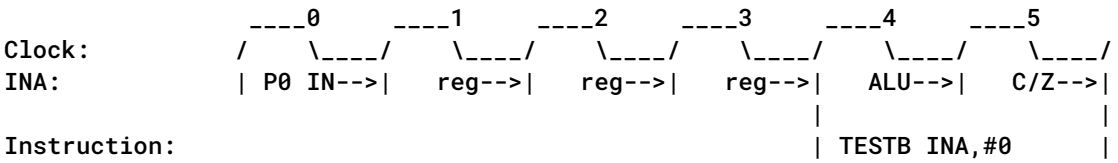
I/O Pin Timing

Between each physical I/O pin and the cog(s) controlling them, there is a chain of three single-bit registers (reg). The live signal (input or output) traverses through this chain on the way to its destination as described below.

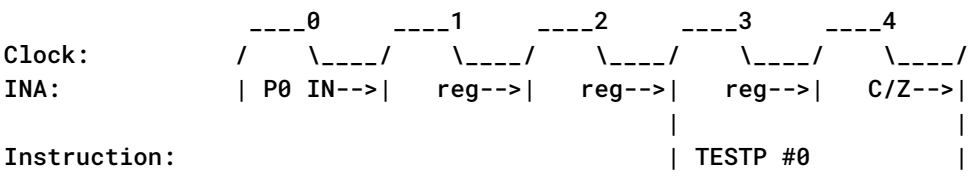
When a **DIR**x/**OUT**x bit is changed by any instruction, it takes *three* additional clocks after the instruction before the pin starts transitioning to the new state. Here this delay is demonstrated using **DRVH**:



When an **IN**x register is read by an instruction, it will reflect the state of the pins registered *three* clocks before the start of the instruction. Here this delay is demonstrated using **TESTB**:



When a **TESTP**/**TESTPN** instruction is used to read a pin, the value read will reflect the state of the pin registered *two* clocks before the start of the instruction. Effectively, **TESTP**/**TESTPN** get fresher **IN**x data than is available via the **IN**x registers:



Pin Modes

Each I/O pin has 13 low-level pin mode configuration bits which determine the mode of operation (1 of 24) for its 3.3 V circuit. The pin mode is set using the **WRPIN** instruction, where the 13 'M' bits within the instruction's D operand specifies the pin mode configuration. Note that in some smart pin modes, the configuration bits are partially overwritten to set things like DAC values.

The format of the **WRPIN**'s D operand value is:

```
%AAAA_BBBB_FFF_MMMMMMMMMMMMM_TT_SSSSS_0
```

- A = PIN input selector
- B = ADJ input selector
- F = PIN and ADJ input logic/filtering (applied to result of PIN and ADJ input selectors)
- M = pin mode
- T = pin **DIR**/**OUT** control (default = %00)
- S = smart mode

(A) PIN or (B) ADJ Input Selector	
%AAAA %BBBB	**Selection**
0xxx	true (default)
1xxx	inverted
x000	this pin's read state (default)
x001	relative +1 pin's read state
x010	relative +2 pin's read state
x011	relative +3 pin's read state
x100	this pin's OUT bit from cogs
x101	relative -3 pin's read state
x110	relative -2 pin's read state
x111	relative -1 pin's read state

(F) PIN and ADJ Logic/Filtering	
%FFF	**Logic/Filter**
000	A, B (default)
001	A AND B, B
010	A OR B, B
011	A XOR B, B
100	A, B, both filtered using global filt0 settings
101	A, B, both filtered using global filt1 settings
110	A, B, both filtered using global filt2 settings
111	A, B, both filtered using global filt3 settings

The resultant 'A' will drive the IN signal in non-smart-pin modes.

(M) Pin Mode

WRPIN D[20:8] Configuration			Resulting Internal Configuration					
M[12:0]	**Input**	**Pin Output[1]**	**CIOHHHLLL**	**OE[2]**	**DAC**	**ADC**	**ADC Mode**	**Comparator**
0000_CIOHHHLLL	Pin Logic	**OUT**	CIOHHHLLL	**DIR**	0	0		0
0001_CIOHHHLLL	Pin Logic	Input	CIOHHHLLL	**DIR**	0	0		0
0010_CIOHHHLLL	Adj Logic	Input	CIOHHHLLL	**DIR**	0	0		0
0011_CIOHHHLLL	Pin Schmitt	**OUT**	CIOHHHLLL	**DIR**	0	0		0
0100_CIOHHHLLL	Pin Schmitt	Input	CIOHHHLLL	**DIR**	0	0		0
0101_CIOHHHLLL	Adj Schmitt	Input	CIOHHHLLL	**DIR**	0	0		0
0110_CIOHHHLLL	Pin > Adj	**OUT**	CIOHHHLLL	**DIR**	0	0		Pin > Adj
0111_CIOHHHLLL	Pin > Adj	Input	CIOHHHLLL	**DIR**	0	0		Pin > Adj
100000_0HHHLLL	ADC, GND	**OUT**	10OHHHLLL	**DIR**	0	1	000	0
100001_0HHHLLL	ADC, Vxxyy	**OUT**	10OHHHLLL	**DIR**	0	1	001	0
100010_0HHHLLL	ADC, float	**OUT**	10OHHHLLL	**DIR**	0	1	010	0
100011_0HHHLLL	ADC, Pin 1x	**OUT**	10OHHHLLL	**DIR**	0	1	011	0
100100_0HHHLLL	ADC, Pin 3.16x	**OUT**	10OHHHLLL	**DIR**	0	1	100	0
100101_0HHHLLL	ADC, Pin 10x	**OUT**	10OHHHLLL	**DIR**	0	1	101	0
100110_0HHHLLL	ADC, Pin 31.6x	**OUT**	10OHHHLLL	**DIR**	0	1	110	0
100111_0HHHLLL	ADC, Pin 100x	**OUT**	10OHHHLLL	**DIR**	0	1	111	0
10100_DDDDDDDD	ADC, Pin 1x [3]	DAC 990 Ω, 3.3 V	10xxxxxxx	0	**DIR**	**OUT**	011	0
10101_DDDDDDDD	ADC, Pin 1x [3]	DAC 600 Ω, 2.0 V	10xxxxxxx	0	**DIR**	**OUT**	011	0
10110_DDDDDDDD	ADC, Pin 1x [3]	DAC 123.75 Ω, 3.3 V	10xxxxxxx	0	**DIR**	**OUT**	011	0
10111_DDDDDDDD	ADC, Pin 1x [3]	DAC 75 Ω, 2.0 V	10xxxxxxx	0	**DIR**	**OUT**	011	0
1100_CDDDDDDDD	Pin > D	**OUT**, 1.5 kΩ	C00001001	**DIR**	0	0		Pin > D
1101_CDDDDDDDD	Pin > D	!Input, 1.5 kΩ	C01001001	**DIR**	0	0		Pin > D
1110_CDDDDDDDD	Adj > D	Input, 1.5 kΩ	C00001001	**DIR**	0	0		Adj > D
1111_CDDDDDDDD	Adj > D	!Input, 1.5 kΩ	C01001001	**DIR**	0	0		Adj > D

[1] **OUT** means output latch bit drives output; Input means the 'Input' column's item drives output

[2] OE is digital logic output enable only; analog output is indicated in the DAC column

[3] if **OUT** bit = 1

Pin Mode Legend

C	IN/OUT
0	Live [1]
1	Clocked [2]

I	IN
0	True
1	Not (inverted)

O	Output
0	True
1	Not (inverted)

HHH LLL	Drive
000	Fast
001	1.5 kΩ
010	15 kΩ
011	150 kΩ
100	1 mA
101	100 µA
110	10 µA
111	Float

OE = digital output enable (when **DIR** bit high)
DAC = digital to analog converter enable (when **DIR** bit high)
ADC = analog to digital converter enable (fixed, or when **OUT** bit high)
OUT = output latch bit; 0: low, 1: high. Exception: DAC modes use **OUT** as 0: disable, 1: enable.
DIR = direction bit; 0: input (float), 1: output (drive) Exception: DAC modes use **DIR** as 0: disable, 1: enable.
DDDDDDDD and D = DAC Level

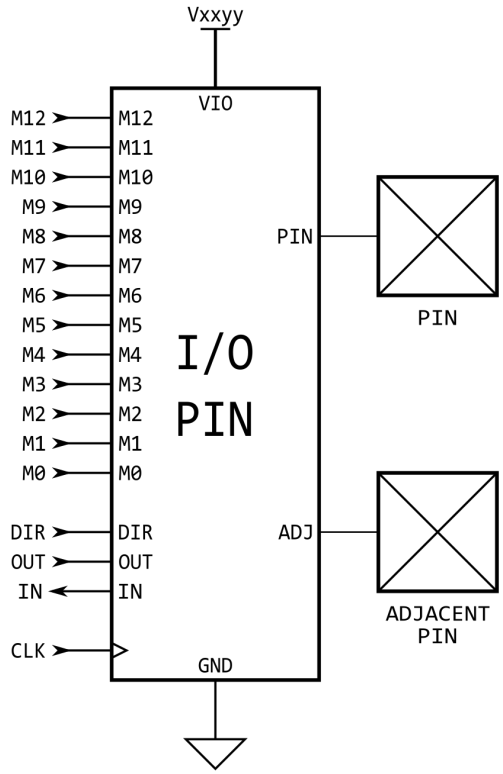
[1] used for feedback operations; provides continuous (non-clocked) signal

[2] signal updates on clock edge only

(T) Pin DIR/OUT Control		
Default (%TT = 00)		
for odd pins	'OTHER' = even pin's NOT (inverted) output state (diff source)	
for even pins	'OTHER' = unique pseudo-random bit (noise source)	
for all pins	'SMART' = smart pin output which overrides **OUT**/OTHER	
'DAC_MODE' is enabled when M[12:10] = %101		
'BIT_DAC' outputs {2{M[7:4]}} for 'high' or {2{M[3:0]}} for 'low' in DAC_MODE		
for smart pin mode "off" (%SSSSS = %00000)		
	DIR enables output	
	for non-DAC_MODE	
	0x	**OUT** drives output
	1x	OTHER drives output
	for DAC_MODE	
	00	**DIR** enables DAC, M[7:0] sets DAC level
	01	**OUT** enables ADC, M[3:0] selects cog DAC channel
	10	**OUT** drives BIT_DAC
	11	OTHER drives BIT_DAC
for smart pin mode "on" (%SSSSS > %00000)		
	x0	output disabled, regardless of **DIR**
	x1	output enabled, regardless of **DIR**
for DAC smart pin modes (%SSSSS = %00001..%00011)		
	0x	**OUT** enables DAC in DAC_MODE, M[7:0] overridden
	1x	OTHER enables DAC in DAC_MODE, M[7:0] overridden
for non-DAC smart pin modes (%SSSSS = %00100..%11111)		
	0x	SMART/**OUT** drives output, or BIT_DAC if DAC_MODE
	1x	SMART/OTHER drives output, or BIT_DAC if DAC_MODE

I/O Pin Circuit

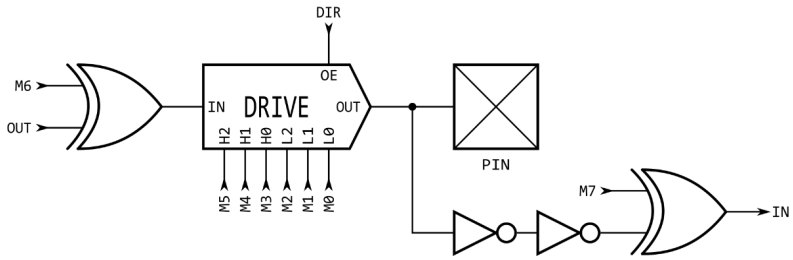
Below is a diagram of a single I/O pin circuit which is powered from its local 3.3V supply pin (Vxxyy). It connects to its own physical pin (PIN), as well as its adjacent odd or even pin (ADJ). I/O Pins P0 and P1 see each other as adjacent pins, as do P2 and P3, etc.



P0..P63
(64 Instances)

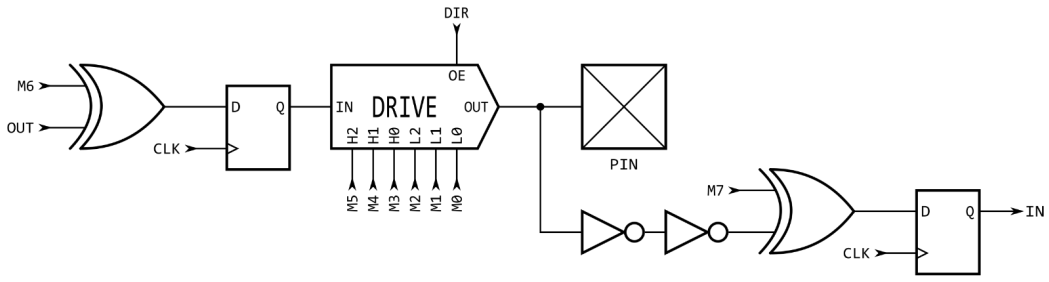
Equivalent Schematics for Each Unique I/O Pin Configuration

%00000MMMMMMMM - Logic

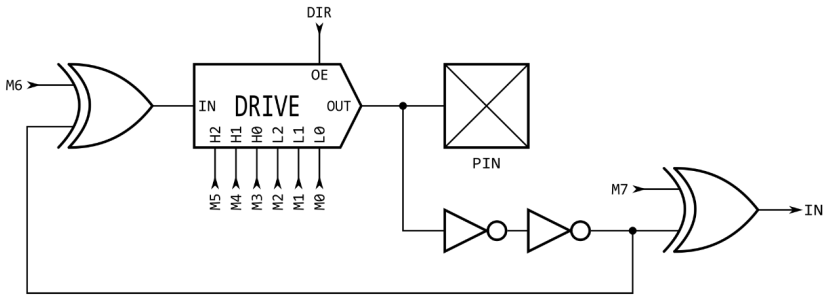


H/L	DRIVE
000	Digital
001	1.5k
010	15k
011	150k
100	1mA
101	100uA
110	10uA
111	Float

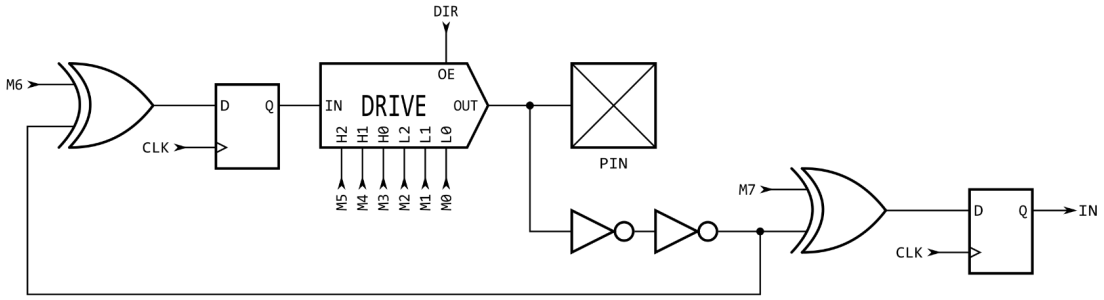
%00001MMMMMMMM - Logic, Clocked



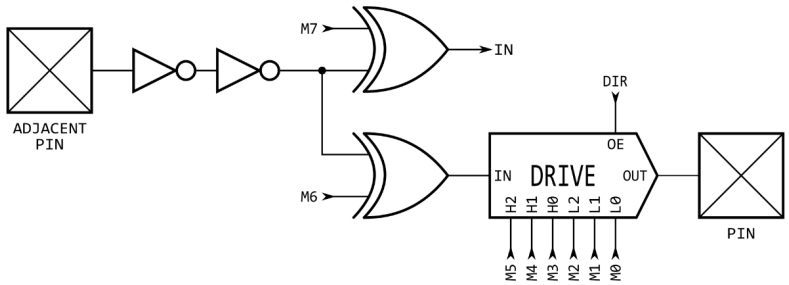
%00010MMMMMMMM - Logic with Feedback



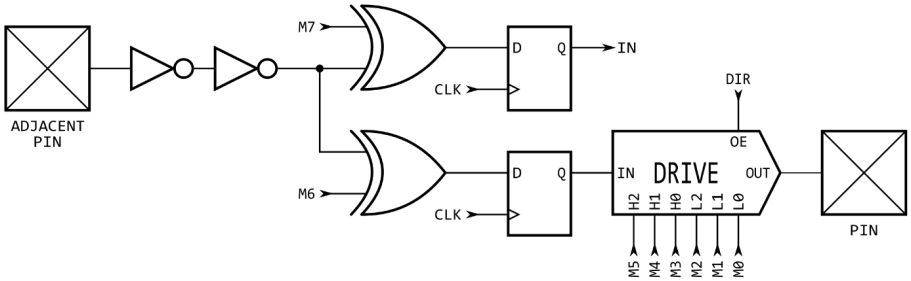
%00011MMMMMMMM - Logic with Feedback, Clocked



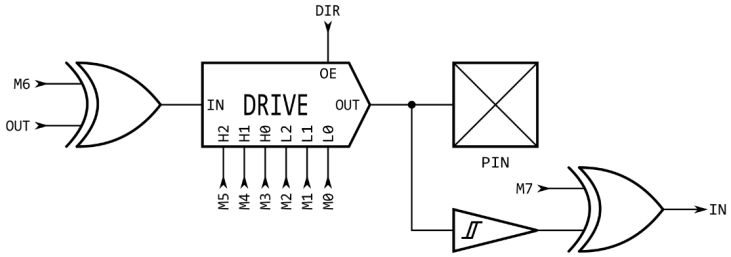
%00100MMMMMMMM - Logic with Adjacent-Pin Feedback



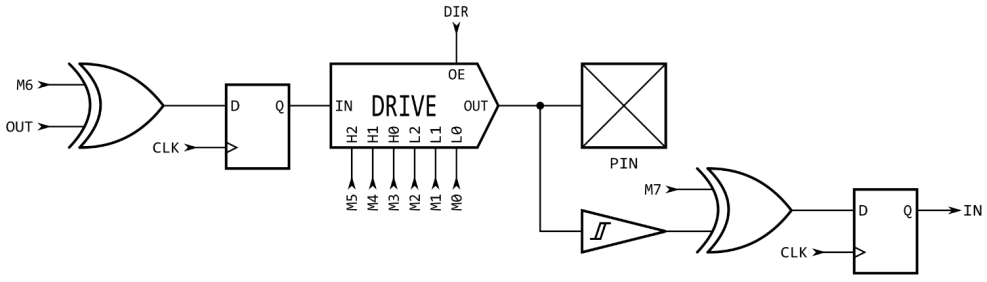
%00101MMMMMMMM - Logic with Adjacent-Pin Feedback, Clocked



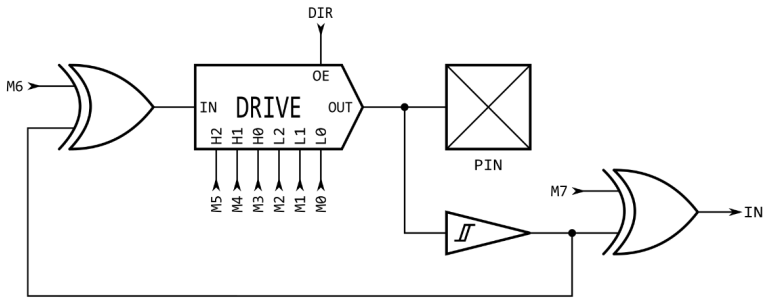
%00110MMMMMMMM - Schmitt



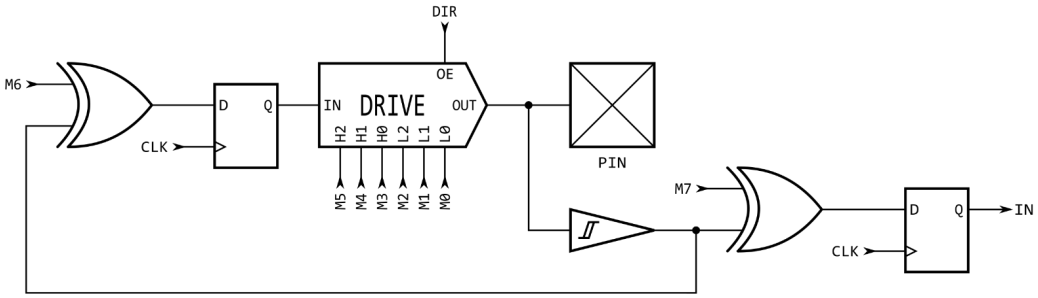
%00111MMMMMMMM - Schmitt, Clocked



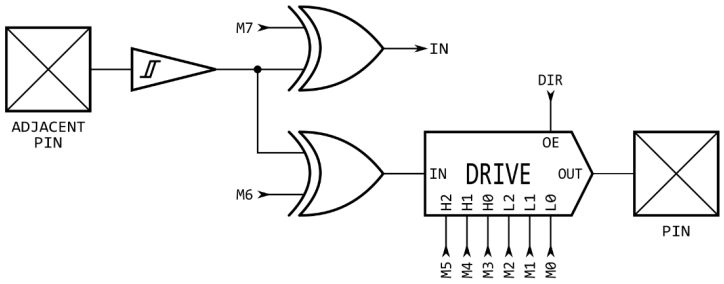
%01000MMMMMMMM - Schmitt with Feedback



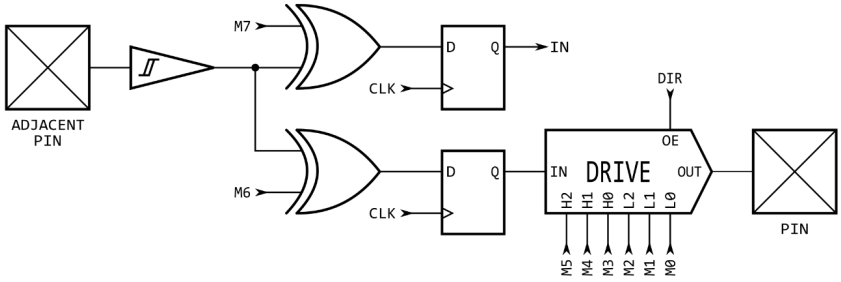
%01001MMMMMMMM - Schmitt with Feedback, Clocked



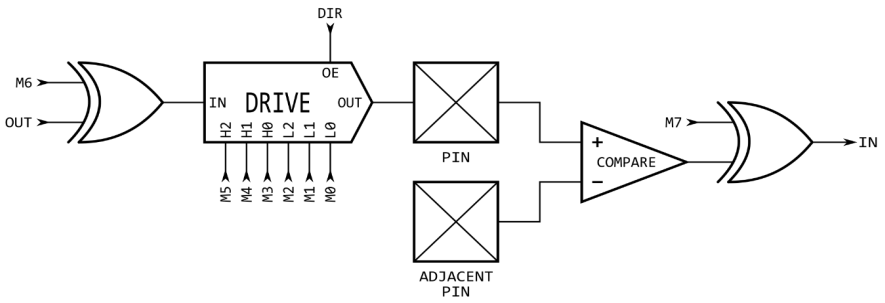
%01010MMMMMMMM - Schmitt with Adjacent-Pin Feedback



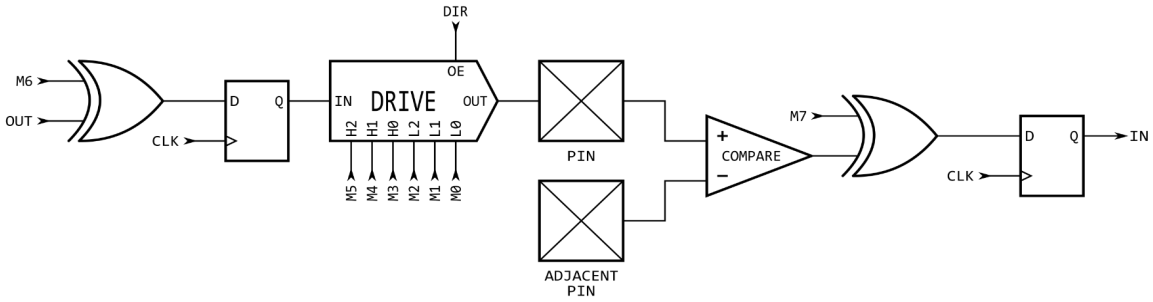
%01011MMMMMMMM - Schmitt with Adjacent-Pin Feedback, Clocked



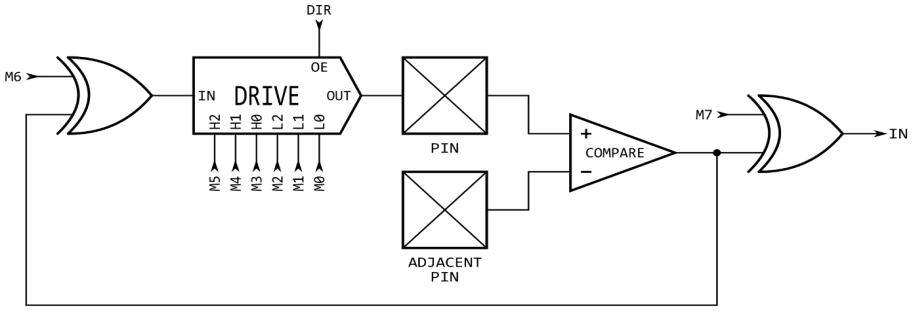
%01100MMMMMMMM - Comparator



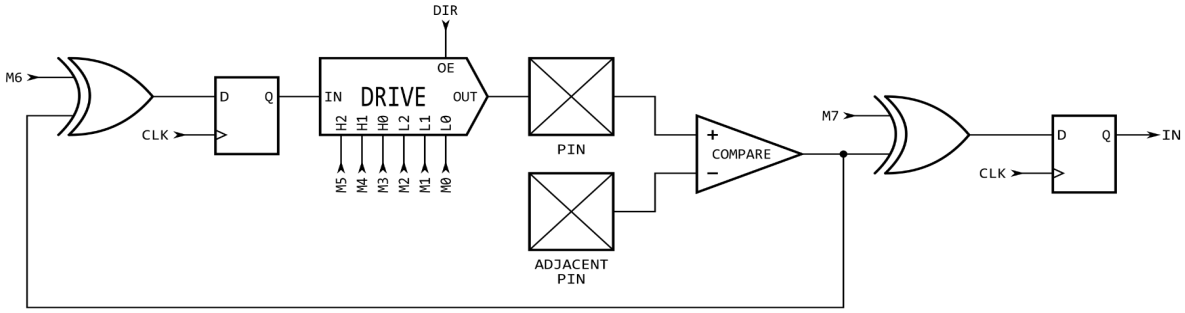
%01101MMMMMMMM - Comparator, Clocked



%01110MMMMMMMM - Comparator with Feedback

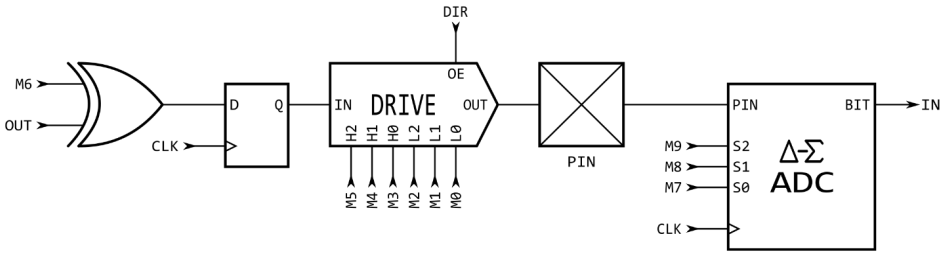


%01111MMMMMMMM - Comparator with Feedback, Clocked



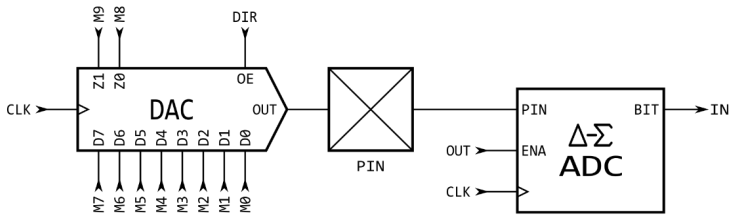
%100MMMMMMMMMM - ADC with Optional Drive

SSS	ADC
000	GND
001	VIO
010	Float
011	1x
100	3.2x
101	10x
110	32x
111	100x

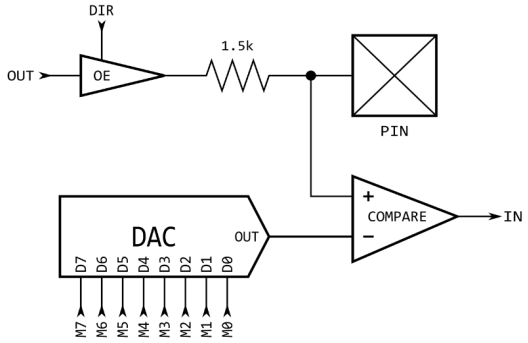


%101MMMMMMMMMM - DAC with Optional ADC

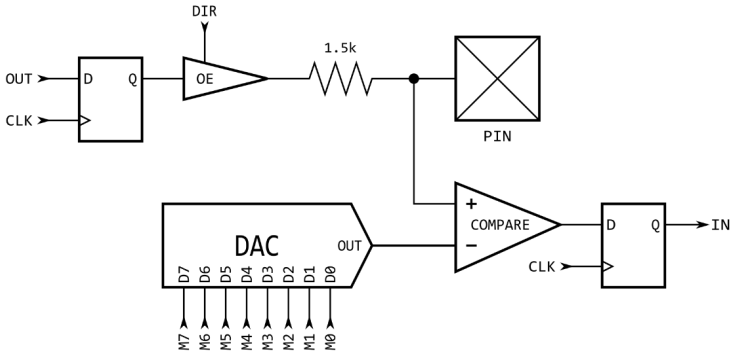
ZZ	DAC
00	990 ohm 3.3V
01	600 ohm 2.0V
10	124 ohm 3.3V
11	75 ohm 2.0V



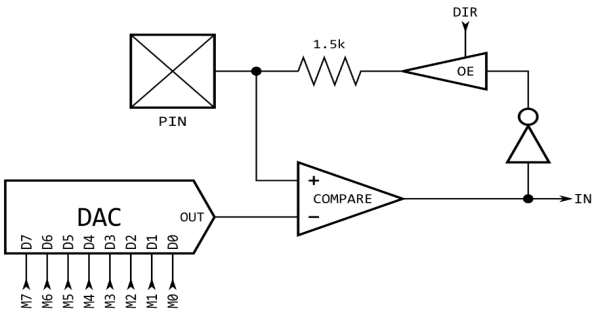
%11000MMMMMMMM - Level Comparator with 1.5k Output



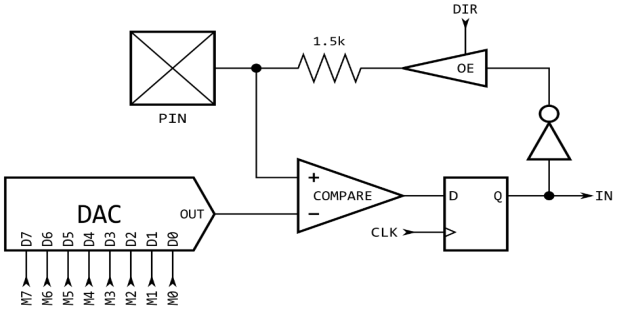
%11001MMMMMMMM - Level Comparator with 1.5k Output, Clocked



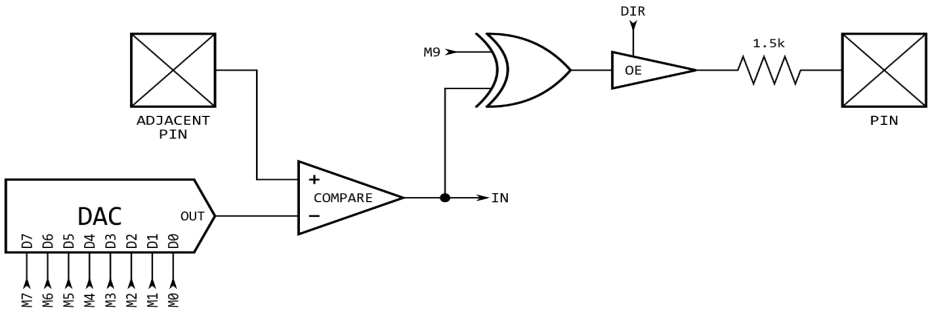
%11010MMMMMMMM - Level Comparator with Local Feedback



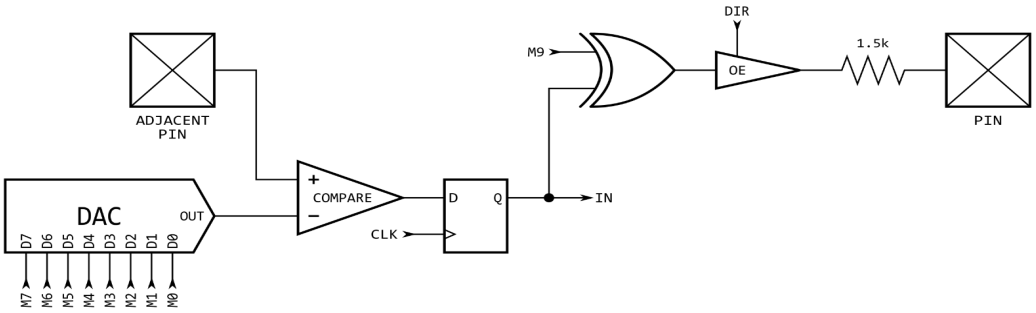
%11011MMMMMMMM - Level Comparator with Local Feedback, Clocked



%111M0MMMMMMMM - Level Comparator with Separate Feedback



%111M1MMMMMMMM - Level Comparator with Separate Feedback, Clocked



Smart Modes

Each I/O pin has built-in 'smart pin' circuitry which (when enabled) performs one of 34 different autonomous functions on the pin. Smart pins free the cogs from the need to micromanage many I/O operations by providing high-bandwidth concurrent hardware functions that cogs could otherwise not perform as well through I/O pin manipulating instructions.

In normal operation, an I/O pin's output enable is controlled by its **DIR** bit, its output state is controlled by its **OUT** bit, and its **IN** bit returns the pin's read state. With smart pin mode enabled, its **DIR** bit is used as an active-low reset signal to the smart pin circuitry, while the output enable state is controlled by a configuration bit. In some modes, the smart pin circuit takes over driving the output state, in which case the **OUT** bit gets ignored. Its **IN** bit serves as a flag to indicate to the cog(s) that the smart pin has completed some function or an event has occurred, and acknowledgment is perhaps needed.

To configure a smart pin, first set its **DIR** bit to low (holding it in reset) then use **WRPIN**, **WXPIN**, and **WYPIN** to establish the mode and related parameters. Once configured, **DIR** can be raised high and the smart pin will begin operating. After that, depending on the mode, you may feed it new data via **WXPIN**/**WYPIN** or retrieve results using **RDPIN**/**RQPIN**. These activities are usually coordinated with the IN signal going high; explained later.

Note that while a smart pin is configured, the %TT bits (of the **WRPIN** instruction's D operand) will govern the pin's output enable, regardless of the **DIR** state.

Smart pins have four 32-bit registers inside of them:

Smart Pin Registers	
32-bit Register	**Purpose**
Mode	smart pin mode, as well as low-level I/O pin mode (write-only)
X	mode-specific parameter (write-only)
Y	mode-specific parameter (write-only)
Z	mode-specific result (read-only)

These four registers are written and read via the following 2-clock instructions, in which S/# is used to select the pin number (0..63) and D/# is the 32-bit data conduit:

```
WRPIN   D/#,S/#         - Set smart pin S/# mode to D/#, ack pin
WXPIN   D/#,S/#         - Set smart pin S/# parameter X to D/#, ack pin
WYPIN   D/#,S/#         - Set smart pin S/# parameter Y to D/#, ack pin
RDPIN   D,S/# {WC}      - Get smart pin S/# result Z into D, flag into C, ack pin
RQPIN   D,S/# {WC}      - Get smart pin S/# result Z into D, flag into C, don't ack pin
AKPIN   S/#             - Acknowledge pin S/#
```

Each smart pin has a 34-bit input bus and a 33-bit output bus that connect it to the cogs.

To configure and control smart pins, each cog writes data and acknowledgement signals to the smart pin input bus. Each smart pin OR's all incoming 34-bit buses from the collective of cogs in the same way **DIR** and **OUT** bits are OR'd before going to the pins. Therefore, if you intend to have multiple cogs execute **WRPIN** / **WXPIN** / **WYPIN** / **RDPIN** / **AKPIN** instructions on the same smart pin, you must be sure that they do so at different times, in order to avoid clobbering each other's bus data. Reading a smart pin with **RDPIN** can cause the same conflict; however,

any number of cogs can read a smart pin simultaneously without bus conflict by using **RQPIN** ('read quiet'), since it does not utilize the smart pin input bus for acknowledgement signalling (like **RDPIN** does).

Each smart pin writes to it's output bus to convey its Z result and a special flag. The **RDPIN** and **RQPIN** multiplex and read these buses, so that a pin's Z result is read into D and its special flag can be read into C. C will be either a mode-related flag or the MSB of the Z result.

When a mode-related event occurs in a smart pin, it raises its **IN** signal to alert the cog(s) that new data is ready, new data can be loaded, or some process has finished. A cog can test for this signal via the **TESTP** instruction and can acknowledge a smart pin by executing a **WRPIN**, **WXPIN**, **WYPIN**, **RDPIN**, or **AKPIN** instruction for it. This acknowledgement causes the smart pin to lower its **IN** signal so that it can be raised again on the next event. After a **WRPIN** / **WXPIN** / **WYPIN** / **RDPIN** / **AKPIN**, it takes two clocks for **IN** to drop, before it can be polled again.

A smart pin can be reset at any time, without the need to reconfigure it, by clearing and then setting its **DIR** bit.

To return a pin to normal mode, do a '**WRPIN #0,pin**'.

(S) Smart Pin Modes		
%SSSSS	**Mode**	**Note**
00000	smart pin off (default)	
00001	long repository	M[12:10] != %101 (not DAC_MODE)
00010	long repository	M[12:10] != %101 (not DAC_MODE)
00011	long repository	M[12:10] != %101 (not DAC_MODE)
00001	DAC noise	M[12:10] = %101 (DAC_MODE)
00010	DAC 16-bit dither, noise	M[12:10] = %101 (DAC_MODE)
00011	DAC 16-bit dither, PWM	M[12:10] = %101 (DAC_MODE)
00100[1]	pulse/cycle output	
00101[1]	transition output	
00110[1]	NCO frequency	
00111[1]	NCO duty	
01000[1]	PWM triangle	
01001[1]	PWM sawtooth	
01010[1]	PWM switch-mode power supply, V and I feedback	
01011	periodic/continuous: A-B quadrature encoder	
01100	periodic/continuous: inc on A-rise & B-high	
01101	periodic/continuous: inc on A-rise & B-high / dec on A-rise & B-low	
01110	periodic/continuous: inc on A-rise {/ dec on B-rise}	
01111	periodic/continuous: inc on A-high {/ dec on B-high}	
10000	time A-states	
10001	time A-highs	
10010	time X A-highs/rises/edges -or- timeout on X A-high/rise/edge	

10011	for X periods, count time	
10100	for X periods, count states	
10101	for periods in X+ clocks, count time	
10110	for periods in X+ clocks, count states	
10111	for periods in X+ clocks, count periods	
11000	ADC sample/filter/capture, internally clocked	
11001	ADC sample/filter/capture, externally clocked	
11010	ADC scope with trigger	
11011[1]	USB host/device	even/odd pin pair = DM/DP
11100[1]	sync serial transmit	A-data, B-clock
11101	sync serial receive	A-data, B-clock
11110[1]	async serial transmit	baud rate
11111	async serial receive	baud rate

[1] **OUT** signal overridden

Rebooting

While normally powered, the Propeller 2 reboots if it receives a low pulse on the RESn pin or executes a **HUBSET ##$1000_0000** instruction. Both reset methods, external (via RESn pin) and internal (via **HUBSET**), behave the same; however, the internal reset is not detectable externally using the RESn pin.

PROPELLER 2 ASSEMBLY LANGUAGE (PASM2) IN BRIEF

Math and Logic Instructions				
Instruction			Description	Clocks Reg, LUT, & Hub
ABS	D	{WC/WZ/WCZ}	Get absolute value of D into D. D = ABS(D). C = D[31]. *	2
ABS	D,{#}S	{WC/WZ/WCZ}	Get absolute value of S into D. D = ABS(S). C = S[31]. *	2
ADD	D,{#}S	{WC/WZ/WCZ}	Add S into D. D = D + S. C = carry of (D + S). *	2
ADDS	D,{#}S	{WC/WZ/WCZ}	Add S into D, signed. D = D + S. C = correct sign of (D + S). *	2
ADDSX	D,{#}S	{WC/WZ/WCZ}	Add (S + C) into D, signed and extended. D = D + S + C. C = correct sign of (D + S + C). Z = Z AND (result == 0).	2
ADDX	D,{#}S	{WC/WZ/WCZ}	Add (S + C) into D, extended. D = D + S + C. C = carry of (D + S + C). Z = Z AND (result == 0).	2
AND	D,{#}S	{WC/WZ/WCZ}	AND S into D. D = D & S. C = parity of result. *	2
ANDN	D,{#}S	{WC/WZ/WCZ}	AND !S into D. D = D & !S. C = parity of result. *	2
BITC	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = C. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].	2
BITH	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = 1. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].	2
BITL	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = 0. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].	2
BITNC	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = !C. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].	2
BITNOT	D,{#}S	{WCZ}	Toggle bits D[S[9:5]+S[4:0]:S[4:0]]. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].	2
BITNZ	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = !Z. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].	2
BITRND	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = RNDs. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].	2
BITZ	D,{#}S	{WCZ}	Bits D[S[9:5]+S[4:0]:S[4:0]] = Z. Other bits unaffected. Prior SETQ overrides S[9:5]. C,Z = original D[S[4:0]].	2
BMASK	D		Get LSB-justified bit mask of size (D[4:0] + 1) into D. D = ($0000_0002 << D[4:0]) - 1.	2
BMASK	D,{#}S		Get LSB-justified bit mask of size (S[4:0] + 1) into D. D = ($0000_0002 << S[4:0]) - 1.	2
CMP	D,{#}S	{WC/WZ/WCZ}	Compare D to S. C = borrow of (D - S). Z = (D == S).	2
CMPM	D,{#}S	{WC/WZ/WCZ}	Compare D to S, get MSB of difference into C. C = MSB of (D - S). Z = (D == S).	2
CMPR	D,{#}S	{WC/WZ/WCZ}	Compare S to D (reverse). C = borrow of (S - D). Z = (D == S).	2
CMPS	D,{#}S	{WC/WZ/WCZ}	Compare D to S, signed. C = correct sign of (D - S). Z = (D == S).	2
CMPSUB	D,{#}S	{WC/WZ/WCZ}	Compare and subtract S from D if D >= S. If D => S then D = D - S and C = 1, else D same and C = 0. *	2
CMPSX	D,{#}S	{WC/WZ/WCZ}	Compare D to (S + C), signed and extended. C = correct sign of (D - (S + C)). Z = Z AND (D == S + C).	2
CMPX	D,{#}S	{WC/WZ/WCZ}	Compare D to (S + C), extended. C = borrow of (D - (S + C)). Z = Z AND (D == S + C).	2
CRCBIT	D,{#}S		Iterate CRC value in D using C and polynomial in S. If (C XOR D[0]) then D = (D >> 1) XOR S, else D = (D >> 1).	2
CRCNIB	D,{#}S		Iterate CRC value in D using Q[31:28] and polynomial in S. Like CRCBIT x 4. Q = Q << 4. Use 'REP #n,#1'+SETQ+CRCNIB+CRCNIB+CRCNIB...	2
DECMOD	D,{#}S	{WC/WZ/WCZ}	Decrement with modulus. If D = 0 then D = S and C = 1, else D = D - 1 and C = 0. *	2
DECOD	D		Decode D[4:0] into D. D = 1 << D[4:0].	2
DECOD	D,{#}S		Decode S[4:0] into D. D = 1 << S[4:0].	2
ENCOD	D	{WC/WZ/WCZ}	Get bit position of top-most '1' in D into D. D = position of top '1' in S (0..31). C = (S != 0). *	2
ENCOD	D,{#}S	{WC/WZ/WCZ}	Get bit position of top-most '1' in S into D. D = position of top '1' in S (0..31). C = (S != 0). *	2
FGE	D,{#}S	{WC/WZ/WCZ}	Force D >= S. If D < S then D = S and C = 1, else D same and C = 0. *	2
FGES	D,{#}S	{WC/WZ/WCZ}	Force D >= S, signed. If D < S then D = S and C = 1, else D same and C = 0. *	2
FLE	D,{#}S	{WC/WZ/WCZ}	Force D <= S. If D > S then D = S and C = 1, else D same and C = 0. *	2
FLES	D,{#}S	{WC/WZ/WCZ}	Force D <= S, signed. If D > S then D = S and C = 1, else D same and C = 0. *	2
GETBYTE	D		Get byte established by prior ALTGB instruction into D.	2
GETBYTE	D,{#}S,#N		Get byte N of S into D. D = {24'b0, S.BYTE[N]).	2
GETNIB	D		Get nibble established by prior ALTGN instruction into D.	2
GETNIB	D,{#}S,#N		Get nibble N of S into D. D = {28'b0, S.NIBBLE[N]).	2
GETWORD	D		Get word established by prior ALTGW instruction into D.	2
GETWORD	D,{#}S,#N		Get word N of S into D. D = {16'b0, S.WORD[N]).	2
INCMOD	D,{#}S	{WC/WZ/WCZ}	Increment with modulus. If D = S then D = 0 and C = 1, else D = D + 1 and C = 0. *	2

LOC	PA/PB/PTRA/PTRB,#{\}A		Get {12'b0, address[19:0]} into PA/PB/PTRA/PTRB (per W). If R = 1, address = PC + A, else address = A. "\" forces R = 0.	2
MERGEB	D		Merge bits of bytes in D. D = {D[31], D[23], D[15], D[7], ...D[24], D[16], D[8], D[0]}.	2
MERGEW	D		Merge bits of words in D. D = {D[31], D[15], D[30], D[14], ...D[17], D[1], D[16], D[0]}.	2
MODC	c	{WC}	Modify C according to cccc. C = cccc[{C,Z}].	2
MODCZ	c,z	{WC/WZ/WCZ}	Modify C and Z according to cccc and zzzz. C = cccc[{C,Z}], Z = zzzz[{C,Z}].	2
MODZ	z	{WZ}	Modify Z according to zzzz. Z = zzzz[{C,Z}].	2
MOV	D,{#}S	{WC/WZ/WCZ}	Move S into D. D = S. C = S[31]. *	2
MOVBYTS	D,{#}S		Move bytes within D, per S. D = {D.BYTE[S[7:6]], D.BYTE[S[5:4]], D.BYTE[S[3:2]], D.BYTE[S[1:0]]}.	2
MUL	D,{#}S	{WZ}	D = unsigned (D[15:0] * S[15:0]). Z = (S == 0) | (D == 0).	2
MULS	D,{#}S	{WZ}	D = signed (D[15:0] * S[15:0]). Z = (S == 0) | (D == 0).	2
MUXC	D,{#}S	{WC/WZ/WCZ}	Mux C into each D bit that is '1' in S. D = (!S & D) | (S & {32{ C}}). C = parity of result. *	2
MUXNC	D,{#}S	{WC/WZ/WCZ}	Mux !C into each D bit that is '1' in S. D = (!S & D) | (S & {32{!C}}). C = parity of result. *	2
MUXNIBS	D,{#}S		For each non-zero nibble in S, copy that nibble into the corresponding D nibble, else leave that D nibble the same.	2
MUXNITS	D,{#}S		For each non-zero bit pair in S, copy that bit pair into the corresponding D bits, else leave that D bit pair the same.	2
MUXNZ	D,{#}S	{WC/WZ/WCZ}	Mux !Z into each D bit that is '1' in S. D = (!S & D) | (S & {32{!Z}}). C = parity of result. *	2
MUXQ	D,{#}S		Used after SETQ. For each '1' bit in Q, copy the corresponding bit in S into D. D = (D & !Q) | (S & Q).	2
MUXZ	D,{#}S	{WC/WZ/WCZ}	Mux Z into each D bit that is '1' in S. D = (!S & D) | (S & {32{ Z}}). C = parity of result. *	2
NEG	D	{WC/WZ/WCZ}	Negate D. D = -D. C = MSB of result. *	2
NEG	D,{#}S	{WC/WZ/WCZ}	Negate S into D. D = -S. C = MSB of result. *	2
NEGC	D	{WC/WZ/WCZ}	Negate D by C. If C = 1 then D = -D, else D = D. C = MSB of result. *	2
NEGC	D,{#}S	{WC/WZ/WCZ}	Negate S by C into D. If C = 1 then D = -S, else D = S. C = MSB of result. *	2
NEGNC	D	{WC/WZ/WCZ}	Negate D by !C. If C = 0 then D = -D, else D = D. C = MSB of result. *	2
NEGNC	D,{#}S	{WC/WZ/WCZ}	Negate S by !C into D. If C = 0 then D = -S, else D = S. C = MSB of result. *	2
NEGNZ	D	{WC/WZ/WCZ}	Negate D by !Z. If Z = 0 then D = -D, else D = D. C = MSB of result. *	2
NEGNZ	D,{#}S	{WC/WZ/WCZ}	Negate S by !Z into D. If Z = 0 then D = -S, else D = S. C = MSB of result. *	2
NEGZ	D	{WC/WZ/WCZ}	Negate D by Z. If Z = 1 then D = -D, else D = D. C = MSB of result. *	2
NEGZ	D,{#}S	{WC/WZ/WCZ}	Negate S by Z into D. If Z = 1 then D = -S, else D = S. C = MSB of result. *	2
NOT	D	{WC/WZ/WCZ}	Get !D into D. D = !D. C = !D[31]. *	2
NOT	D,{#}S	{WC/WZ/WCZ}	Get !S into D. D = !S. C = !S[31]. *	2
ONES	D	{WC/WZ/WCZ}	Get number of '1's in D into D. D = number of '1's in S (0..32). C = LSB of result. *	2
ONES	D,{#}S	{WC/WZ/WCZ}	Get number of '1's in S into D. D = number of '1's in S (0..32). C = LSB of result. *	2
OR	D,{#}S	{WC/WZ/WCZ}	OR S into D. D = D | S. C = parity of result. *	2
RCL	D,{#}S	{WC/WZ/WCZ}	Rotate carry left. D = [63:32] of ({D[31:0], {32{C}}} << S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[31]. *	2
RCR	D,{#}S	{WC/WZ/WCZ}	Rotate carry right. D = [31:0] of ({{32{C}}, D[31:0]} >> S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[0]. *	2
RCZL	D	{WC/WZ/WCZ}	Rotate C,Z left through D. D = {D[29:0], C, Z}. C = D[31], Z = D[30].	2
RCZR	D	{WC/WZ/WCZ}	Rotate C,Z right through D. D = {C, Z, D[31:2]}. C = D[1], Z = D[0].	2
REV	D		Reverse D bits. D = D[0:31].	2
RGBEXP	D		Expand 5:6:5 RGB value in D[15:0] into 8:8:8 value in D[31:8]. D = {D[15:11,15:13], D[10:5,10:9], D[4:0,4:2], 8'b0}.	2
RGBSQZ	D		Squeeze 8:8:8 RGB value in D[31:8] into 5:6:5 value in D[15:0]. D = {15'b0, D[31:27], D[23:18], D[15:11]}.	2
ROL	D,{#}S	{WC/WZ/WCZ}	Rotate left. D = [63:32] of ({D[31:0], D[31:0]} << S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[31]. *	2
ROLBYTE	D		Rotate-left byte established by prior ALTGB instruction into D.	2
ROLBYTE	D,{#}S,#N		Rotate-left byte N of S into D. D = {D[23:0], S.BYTE[N]).	2
ROLNIB	D		Rotate-left nibble established by prior ALTGN instruction into D.	2
ROLNIB	D,{#}S,#N		Rotate-left nibble N of S into D. D = {D[27:0], S.NIBBLE[N]).	2
ROLWORD	D		Rotate-left word established by prior ALTGW instruction into D.	2
ROLWORD	D,{#}S,#N		Rotate-left word N of S into D. D = {D[15:0], S.WORD[N]).	2
ROR	D,{#}S	{WC/WZ/WCZ}	Rotate right. D = [31:0] of ({D[31:0], D[31:0]} >> S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[0]. *	2
SAL	D,{#}S	{WC/WZ/WCZ}	Shift arithmetic left. D = [63:32] of ({D[31:0], {32{D[0]}}} << S[4:0]). C = last bit shifted out if S[4:0] > 0, else	2

		D[31]. *	
SAR	D,{#}S {WC/WZ/WCZ}	Shift arithmetic right. D = [31:0] of ({{32{D[31]}}, D[31:0]} >> S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[0]. *	2
SCA	D,{#}S {WZ}	Next instruction's S value = unsigned (D[15:0] * S[15:0]) >> 16. *	2
SCAS	D,{#}S {WZ}	Next instruction's S value = signed (D[15:0] * S[15:0]) >> 14. In this scheme, $4000 = 1.0 and $C000 = -1.0. *	2
SETBYTE	{#}S	Set S[7:0] into byte established by prior ALTSB instruction.	2
SETBYTE	D,{#}S,#N	Set S[7:0] into byte N in D, keeping rest of D same.	2
SETD	D,{#}S	Set D field of D to S[8:0]. D = {D[31:18], S[8:0], D[8:0]}.	2
SETNIB	{#}S	Set S[3:0] into nibble established by prior ALTSN instruction.	2
SETNIB	D,{#}S,#N	Set S[3:0] into nibble N in D, keeping rest of D same.	2
SETR	D,{#}S	Set R field of D to S[8:0]. D = {D[31:28], S[8:0], D[18:0]}.	2
SETS	D,{#}S	Set S field of D to S[8:0]. D = {D[31:9], S[8:0]}.	2
SETWORD	{#}S	Set S[15:0] into word established by prior ALTSW instruction.	2
SETWORD	D,{#}S,#N	Set S[15:0] into word N in D, keeping rest of D same.	2
SEUSSF	D	Relocate and periodically invert bits within D. Returns to original value on 32nd iteration. Forward pattern.	2
SEUSSR	D	Relocate and periodically invert bits within D. Returns to original value on 32nd iteration. Reverse pattern.	2
SHL	D,{#}S {WC/WZ/WCZ}	Shift left. D = [63:32] of ({D[31:0], 32'b0} << S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[31]. *	2
SHR	D,{#}S {WC/WZ/WCZ}	Shift right. D = [31:0] of ({32'b0, D[31:0]} >> S[4:0]). C = last bit shifted out if S[4:0] > 0, else D[0]. *	2
SIGNX	D,{#}S {WC/WZ/WCZ}	Sign-extend D from bit S[4:0]. C = MSB of result. *	2
SPLITB	D	Split every 4th bit of D into bytes. D = {D[31], D[27], D[23], D[19], ...D[12], D[8], D[4], D[0]}.	2
SPLITW	D	Split odd/even bits of D into words. D = {D[31], D[29], D[27], D[25], ...D[6], D[4], D[2], D[0]}.	2
SUB	D,{#}S {WC/WZ/WCZ}	Subtract S from D. D = D - S. C = borrow of (D - S). *	2
SUBR	D,{#}S {WC/WZ/WCZ}	Subtract D from S (reverse). D = S - D. C = borrow of (S - D). *	2
SUBS	D,{#}S {WC/WZ/WCZ}	Subtract S from D, signed. D = D - S. C = correct sign of (D - S). *	2
SUBSX	D,{#}S {WC/WZ/WCZ}	Subtract (S + C) from D, signed and extended. D = D - (S + C). C = correct sign of (D - (S + C)). Z = Z AND (result == 0).	2
SUBX	D,{#}S {WC/WZ/WCZ}	Subtract (S + C) from D, extended. D = D - (S + C). C = borrow of (D - (S + C)). Z = Z AND (result == 0).	2
SUMC	D,{#}S {WC/WZ/WCZ}	Sum +/-S into D by C. If C = 1 then D = D - S, else D = D + S. C = correct sign of (D +/- S). *	2
SUMNC	D,{#}S {WC/WZ/WCZ}	Sum +/-S into D by !C. If C = 0 then D = D - S, else D = D + S. C = correct sign of (D +/- S). *	2
SUMNZ	D,{#}S {WC/WZ/WCZ}	Sum +/-S into D by !Z. If Z = 0 then D = D - S, else D = D + S. C = correct sign of (D +/- S). *	2
SUMZ	D,{#}S {WC/WZ/WCZ}	Sum +/-S into D by Z. If Z = 1 then D = D - S, else D = D + S. C = correct sign of (D +/- S). *	2
TEST	D {WC/WZ/WCZ}	Test D. C = parity of D. Z = (D == 0).	2
TEST	D,{#}S {WC/WZ/WCZ}	Test D with S. C = parity of (D & S). Z = ((D & S) == 0).	2
TESTB	D,{#}S WC/WZ	Test bit S[4:0] of D, write to C/Z. C/Z = D[S[4:0]].	2
TESTB	D,{#}S ORC/ORZ	Test bit S[4:0] of D, OR into C/Z. C/Z = C/Z OR D[S[4:0]].	2
TESTB	D,{#}S ANDC/ANDZ	Test bit S[4:0] of D, AND into C/Z. C/Z = C/Z AND D[S[4:0]].	2
TESTB	D,{#}S XORC/XORZ	Test bit S[4:0] of D, XOR into C/Z. C/Z = C/Z XOR D[S[4:0]].	2
TESTBN	D,{#}S WC/WZ	Test bit S[4:0] of !D, write to C/Z. C/Z = !D[S[4:0]].	2
TESTBN	D,{#}S ORC/ORZ	Test bit S[4:0] of !D, OR into C/Z. C/Z = C/Z OR !D[S[4:0]].	2
TESTBN	D,{#}S ANDC/ANDZ	Test bit S[4:0] of !D, AND into C/Z. C/Z = C/Z AND !D[S[4:0]].	2
TESTBN	D,{#}S XORC/XORZ	Test bit S[4:0] of !D, XOR into C/Z. C/Z = C/Z XOR !D[S[4:0]].	2
TESTN	D,{#}S {WC/WZ/WCZ}	Test D with !S. C = parity of (D & !S). Z = ((D & !S) == 0).	2
WRC	D	Write 0 or 1 to D, according to C. D = {31'b0, C).	2
WRNC	D	Write 0 or 1 to D, according to !C. D = {31'b0, !C).	2
WRNZ	D	Write 0 or 1 to D, according to !Z. D = {31'b0, !Z).	2
WRZ	D	Write 0 or 1 to D, according to Z. D = {31'b0, Z).	2
XOR	D,{#}S {WC/WZ/WCZ}	XOR S into D. D = D ^ S. C = parity of result. *	2
XORO32	D	Iterate D with xoroshiro32+ PRNG algorithm and put PRNG result into next instruction's S.	2
ZEROX	D,{#}S {WC/WZ/WCZ}	Zero-extend D above bit S[4:0]. C = MSB of result. *	2

Pin & Smart Pin Instructions				
Instruction			Description	Clocks Cog, LUT & Hub
Pin				
DIRC	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = C. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.	2
DIRH	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = 1. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.	2
DIRL	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = 0. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.	2
DIRNC	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = !C. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.	2
DIRNOT	{#}D	{WCZ}	Toggle DIR bits of pins D[10:6]+D[5:0]..D[5:0]. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.	2
DIRNZ	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = !Z. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.	2
DIRRND	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = RNDs. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.	2
DIRZ	{#}D	{WCZ}	DIR bits of pins D[10:6]+D[5:0]..D[5:0] = Z. Wraps within DIRA/DIRB. Prior SETQ overrides D[10:6]. C,Z = DIR bit.	2
DRVC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = C. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
DRVH	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 1. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
DRVL	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 0. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
DRVNC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !C. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
DRVNOT	{#}D	{WCZ}	Toggle OUT bits of pins D[10:6]+D[5:0]..D[5:0]. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
DRVNZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !Z. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
DRVRND	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = RNDs. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
DRVZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = Z. DIR bits = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
FLTC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = C. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
FLTH	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 1. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
FLTL	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 0. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
FLTNC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !C. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
FLTNOT	{#}D	{WCZ}	Toggle OUT bits of pins D[10:6]+D[5:0]..D[5:0]. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
FLTNZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !Z. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
FLTRND	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = RNDs. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
FLTZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = Z. DIR bits = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
OUTC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = C. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
OUTH	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 1. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
OUTL	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = 0. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
OUTNC	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !C. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
OUTNOT	{#}D	{WCZ}	Toggle OUT bits of pins D[10:6]+D[5:0]..D[5:0]. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z =	2

			OUT bit.	
OUTNZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = !Z. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
OUTRND	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = RNDs. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
OUTZ	{#}D	{WCZ}	OUT bits of pins D[10:6]+D[5:0]..D[5:0] = Z. Wraps within OUTA/OUTB. Prior SETQ overrides D[10:6]. C,Z = OUT bit.	2
TESTP	{#}D	WC/WZ	Test IN bit of pin D[5:0], write to C/Z. C/Z = IN[D[5:0]].	2
TESTP	{#}D	ORC/ORZ	Test IN bit of pin D[5:0], OR into C/Z. C/Z = C/Z OR IN[D[5:0]].	2
TESTP	{#}D	ANDC/ANDZ	Test IN bit of pin D[5:0], AND into C/Z. C/Z = C/Z AND IN[D[5:0]].	2
TESTP	{#}D	XORC/XORZ	Test IN bit of pin D[5:0], XOR into C/Z. C/Z = C/Z XOR IN[D[5:0]].	2
TESTPN	{#}D	WC/WZ	Test !IN bit of pin D[5:0], write to C/Z. C/Z = !IN[D[5:0]].	2
TESTPN	{#}D	ORC/ORZ	Test !IN bit of pin D[5:0], OR into C/Z. C/Z = C/Z OR !IN[D[5:0]].	2
TESTPN	{#}D	ANDC/ANDZ	Test !IN bit of pin D[5:0], AND into C/Z. C/Z = C/Z AND !IN[D[5:0]].	2
TESTPN	{#}D	XORC/XORZ	Test !IN bit of pin D[5:0], XOR into C/Z. C/Z = C/Z XOR !IN[D[5:0]].	2
Smart Pin				
AKPIN	{#}S		Acknowledge smart pins S[10:6]+S[5:0]..S[5:0]. Wraps within A/B pins. Prior SETQ overrides S[10:6].	2
GETSCP	D		Get four-channel oscilloscope samples into D. D = {ch3[7:0],ch2[7:0],ch1[7:0],ch0[7:0]}.	2
RDPIN	D,{#}S	{WC}	Read smart pin S[5:0] result "Z" into D, acknowledge smart pin. C = modal result.	2
RQPIN	D,{#}S	{WC}	Read smart pin S[5:0] result "Z" into D, don't acknowledge smart pin ("Q" in RQPIN means "quiet"). C = modal result.	2
SETDACS	{#}D		DAC3 = D[31:24], DAC2 = D[23:16], DAC1 = D[15:8], DAC0 = D[7:0].	2
SETSCP	{#}D		Set four-channel oscilloscope enable to D[6] and set input pin base to D[5:2].	2
WRPIN	{#}D,{#}S		Set mode of smart pins S[10:6]+S[5:0]..S[5:0] to D, acknowledge smart pins. Wraps within A/B pins. Prior SETQ overrides S[10:6].	2
WXPIN	{#}D,{#}S		Set "X" of smart pins S[10:6]+S[5:0]..S[5:0] to D, acknowledge smart pins. Wraps within A/B pins. Prior SETQ overrides S[10:6].	2
WYPIN	{#}D,{#}S		Set "Y" of smart pins S[10:6]+S[5:0]..S[5:0] to D, acknowledge smart pins. Wraps within A/B pins. Prior SETQ overrides S[10:6].	2

Branch Instructions

Instruction			Description	Clocks Cog & LUT / Hub
CALL	#{\}A		Call to A by pushing {C, Z, 10'b0, PC[19:0]} onto stack. If R = 1 then PC += A, else PC = A. "\" forces R = 0.	4 / 13...20
CALL	D	{WC/WZ/WCZ}	Call to D by pushing {C, Z, 10'b0, PC[19:0]} onto stack. C = D[31], Z = D[30], PC = D[19:0].	4 / 13...20
CALLA	#{\}A		Call to A by writing {C, Z, 10'b0, PC[19:0]} to hub long at PTRA++. If R = 1 then PC += A, else PC = A. "\" forces R = 0.	5...12 [1] / 14...32 [1]
CALLA	D	{WC/WZ/WCZ}	Call to D by writing {C, Z, 10'b0, PC[19:0]} to hub long at PTRA++. C = D[31], Z = D[30], PC = D[19:0].	5...12 [1] / 14...32 [1]
CALLB	#{\}A		Call to A by writing {C, Z, 10'b0, PC[19:0]} to hub long at PTRB++. If R = 1 then PC += A, else PC = A. "\" forces R = 0.	5...12 [1] / 14...32 [1]
CALLB	D	{WC/WZ/WCZ}	Call to D by writing {C, Z, 10'b0, PC[19:0]} to hub long at PTRB++. C = D[31], Z = D[30], PC = D[19:0].	5...12 [1] / 14...32 [1]
CALLD	D,{#}S	{WC/WZ/WCZ}	Call to S** by writing {C, Z, 10'b0, PC[19:0]} to D. C = S[31], Z = S[30].	4 / 13...20
CALLD	PA/PB/PTRA/PTRB,#{\}A		Call to A by writing {C, Z, 10'b0, PC[19:0]} to PA/PB/PTRA/PTRB (per W). If R = 1 then PC += A, else PC = A. "\" forces R = 0.	4 / 13...20
CALLPA	{#}D,{#}S		Call to S** by pushing {C, Z, 10'b0, PC[19:0]} onto stack, copy D to PA.	4 / 13...20
CALLPB	{#}D,{#}S		Call to S** by pushing {C, Z, 10'b0, PC[19:0]} onto stack, copy D to PB.	4 / 13...20
DJF	D,{#}S		Decrement D and jump to S** if result is $FFFF_FFFF.	2 or 4 / 2 or 13...20
DJNF	D,{#}S		Decrement D and jump to S** if result is not $FFFF_FFFF.	2 or 4 / 2 or 13...20
DJNZ	D,{#}S		Decrement D and jump to S** if result is not zero.	2 or 4 / 2 or 13...20
DJZ	D,{#}S		Decrement D and jump to S** if result is zero.	2 or 4 / 2 or 13...20
EXECF	{#}D		Jump to D[9:0] in cog/LUT and set SKIPF pattern to D[31:10]. PC = {10'b0, D[9:0]}.	4 / 4

IJNZ	D,{#}S		Increment D and jump to S** if result is not zero.	2 or 4 / 2 or 13...20
IJZ	D,{#}S		Increment D and jump to S** if result is zero.	2 or 4 / 2 or 13...20
JMP	#{\}A		Jump to A. If R = 1 then PC += A, else PC = A. "\" forces R = 0.	4 / 13...20
JMP	D	{WC/WZ/WCZ}	Jump to D. C = D[31], Z = D[30], PC = D[19:0].	4 / 13...20
JMPREL	{#}D		Jump ahead/back by D instructions. For cogex, PC += D[19:0]. For hubex, PC += D[17:0] << 2.	4 / 13...20
REP	{#}D,{#}S		Execute next D[8:0] instructions S times. If S = 0, repeat instructions infinitely. If D[8:0] = 0, nothing repeats.	2 / 2
RESI0			Resume from INT0. (CALLD $1FE,$1FF WCZ)	4 / 13...20
RESI1			Resume from INT1. (CALLD $1F4,$1F5 WCZ)	4 / 13...20
RESI2			Resume from INT2. (CALLD $1F2,$1F3 WCZ)	4 / 13...20
RESI3			Resume from INT3. (CALLD $1F0,$1F1 WCZ)	4 / 13...20
RET		{WC/WZ/WCZ}	Return by popping stack (K). C = K[31], Z = K[30], PC = K[19:0].	4 / 13...20
RETA		{WC/WZ/WCZ}	Return by reading hub long (L) at --PTRA. C = L[31], Z = L[30], PC = L[19:0].	11...18 [1] / 20...40 [1]
RETB		{WC/WZ/WCZ}	Return by reading hub long (L) at --PTRB. C = L[31], Z = L[30], PC = L[19:0].	11...18 [1] / 20...40 [1]
RETI0			Return from INT0. (CALLD $1FF,$1FF WCZ)	4 / 13...20
RETI1			Return from INT1. (CALLD $1FF,$1F5 WCZ)	4 / 13...20
RETI2			Return from INT2. (CALLD $1FF,$1F3 WCZ)	4 / 13...20
RETI3			Return from INT3. (CALLD $1FF,$1F1 WCZ)	4 / 13...20
SKIP	{#}D		Skip instructions per D. Subsequent instructions 0..31 get cancelled for each '1' bit in D[0]..D[31].	2 / 2
SKIPF	{#}D		Skip cog/LUT instructions fast per D. Like SKIP, but instead of cancelling instructions, the PC leaps over them.	2 / ILLEGAL
TJF	D,{#}S		Test D and jump to S** if D is full (D = $FFFF_FFFF).	2 or 4 / 2 or 13...20
TJNF	D,{#}S		Test D and jump to S** if D is not full (D != $FFFF_FFFF).	2 or 4 / 2 or 13...20
TJNS	D,{#}S		Test D and jump to S** if D is not signed (D[31] = 0).	2 or 4 / 2 or 13...20
TJNZ	D,{#}S		Test D and jump to S** if D is not zero.	2 or 4 / 2 or 13...20
TJS	D,{#}S		Test D and jump to S** if D is signed (D[31] = 1).	2 or 4 / 2 or 13...20
TJV	D,{#}S		Test D and jump to S** if D overflowed (D[31] != C, C = 'correct sign' from last addition/subtraction).	2 or 4 / 2 or 13...20
TJZ	D,{#}S		Test D and jump to S** if D is zero.	2 or 4 / 2 or 13...20

[1] +1 if crosses hub long

Hub Control, FIFO, & RAM Instructions				
Instruction			**Description**	**Clocks Cog & LUT / Hub**
Hub Control				
COGID	{#}D	{WC}	If D is register and no WC, get cog ID (0 to 15) into D. If WC, check status of cog D[3:0], C = 1 if on.	2...9, +2 if result / same
COGINIT	{#}D,{#}S	{WC}	Start cog selected by D. S[19:0] sets hub startup address and PTRB of cog. Prior SETQ sets PTRA of cog.	2...9, +2 if result / same
COGSTOP	{#}D		Stop cog D[3:0].	2...9 / same
LOCKNEW	D	{WC}	Request a LOCK. D will be written with the LOCK number (0 to 15). C = 1 if no LOCK available.	4...11 / same
LOCKREL	{#}D	{WC}	Release LOCK D[3:0]. If D is a register and WC, get current/last cog id of LOCK owner into D and LOCK status into C.	2...9, +2 if result / same
LOCKRET	{#}D		Return LOCK D[3:0] for reallocation.	2...9 / same
LOCKTRY	{#}D	{WC}	Try to get LOCK D[3:0]. C = 1 if got LOCK. LOCKREL releases LOCK. LOCK is also released if owner cog stops or restarts.	2...9, +2 if result / same
HUBSET	{#}D		Set hub configuration to D.	2...9 / same
Hub FIFO				
GETPTR	D		Get current FIFO hub pointer into D.	2 / FIFO IN USE
FBLOCK	{#}D,{#}S		Set next block for when block wraps. D[13:0] = block size in 64-byte units (0 = max), S[19:0] = block start address.	2 / FIFO IN USE
RDFAST	{#}D,{#}S		Begin new fast hub read via FIFO. D[31] = no wait, D[13:0] = block size in 64-byte units (0 = max), S[19:0] = block start address.	2 or WRFAST finish + 10...17 / FIFO IN USE
WRFAST	{#}D,{#}S		Begin new fast hub write via FIFO. D[31] = no wait, D[13:0] = block size in 64-byte units (0 = max), S[19:0] = block start address.	2 or WRFAST finish + 3 / FIFO IN USE
RFBYTE	D	{WC/WZ/WCZ}	Used after RDFAST. Read zero-extended byte from FIFO into D. C = MSB of byte. *	2 / FIFO IN USE

RFLONG	D	{WC/WZ/WCZ}	Used after RDFAST. Read long from FIFO into D. C = MSB of long. *	2 / FIFO IN USE
RFVAR	D	{WC/WZ/WCZ}	Used after RDFAST. Read zero-extended 1..4-byte value from FIFO into D. C = 0. *	2 / FIFO IN USE
RFVARS	D	{WC/WZ/WCZ}	Used after RDFAST. Read sign-extended 1..4-byte value from FIFO into D. C = MSB of value. *	2 / FIFO IN USE
RFWORD	D	{WC/WZ/WCZ}	Used after RDFAST. Read zero-extended word from FIFO into D. C = MSB of word. *	2 / FIFO IN USE
WFBYTE	{#}D		Used after WRFAST. Write byte in D[7:0] into FIFO.	2 / FIFO IN USE
WFLONG	{#}D		Used after WRFAST. Write long in D[31:0] into FIFO.	2 / FIFO IN USE
WFWORD	{#}D		Used after WRFAST. Write word in D[15:0] into FIFO.	2 / FIFO IN USE
Hub RAM				
POPA	D	{WC/WZ/WCZ}	Read long from hub address --PTRA into D. C = MSB of long. *	9...16 [1] / 9...26 [1]
POPB	D	{WC/WZ/WCZ}	Read long from hub address --PTRB into D. C = MSB of long. *	9...16 [1] / 9...26 [1]
RDBYTE	D,{#}S/P	{WC/WZ/WCZ}	Read zero-extended byte from hub address {#}S/PTRx into D. C = MSB of byte. *	9...16 / 9...26
RDLONG	D,{#}S/P	{WC/WZ/WCZ}	Read long from hub address {#}S/PTRx into D. C = MSB of long. * Prior SETQ/SETQ2 invokes cog/LUT block transfer.	9...16 [1] / 9...26 [1]
RDWORD	D,{#}S/P	{WC/WZ/WCZ}	Read zero-extended word from hub address {#}S/PTRx into D. C = MSB of word. *	9...16 [1] / 9...26 [1]
PUSHA	{#}D		Write long in D[31:0] to hub address PTRA++.	3...10 [1] / 3...20 [1]
PUSHB	{#}D		Write long in D[31:0] to hub address PTRB++.	3...10 [1] / 3...20 [1]
WMLONG	D,{#}S/P		Write only non-SOO bytes in D[31:0] to hub address {#}S/PTRx. Prior SETQ/SETQ2 invokes cog/LUT block transfer.	3...10 [1] / 3...20 [1]
WRBYTE	{#}D,{#}S/P		Write byte in D[7:0] to hub address {#}S/PTRx.	3...10 / 3...20
WRLONG	{#}D,{#}S/P		Write long in D[31:0] to hub address {#}S/PTRx. Prior SETQ/SETQ2 invokes cog/LUT block transfer.	3...10 [1] / 3...20 [1]
WRWORD	{#}D,{#}S/P		Write word in D[15:0] to hub address {#}S/PTRx.	3...10 [1] / 3...20 [1]

[1] +1 if crosses hub long

Event Instructions			
Instruction		**Description**	**Clocks Cog & LUT / Hub**
ADDCT1	D,{#}S	Set CT1 event to trigger on CT = D + S. Adds S into D.	2
ADDCT2	D,{#}S	Set CT2 event to trigger on CT = D + S. Adds S into D.	2
ADDCT3	D,{#}S	Set CT3 event to trigger on CT = D + S. Adds S into D.	2
COGATN	{#}D	Strobe "attention" of all cogs whose corresponding bits are high in D[15:0].	2
JATN	{#}S	Jump to S** if ATN event flag is set.	2 or 4 / 2 or 13...20
JCT1	{#}S	Jump to S** if CT1 event flag is set.	2 or 4 / 2 or 13...20
JCT2	{#}S	Jump to S** if CT2 event flag is set.	2 or 4 / 2 or 13...20
JCT3	{#}S	Jump to S** if CT3 event flag is set.	2 or 4 / 2 or 13...20
JFBW	{#}S	Jump to S** if FBW event flag is set.	2 or 4 / 2 or 13...20
JINT	{#}S	Jump to S** if INT event flag is set.	2 or 4 / 2 or 13...20
JNATN	{#}S	Jump to S** if ATN event flag is clear.	2 or 4 / 2 or 13...20
JNCT1	{#}S	Jump to S** if CT1 event flag is clear.	2 or 4 / 2 or 13...20
JNCT2	{#}S	Jump to S** if CT2 event flag is clear.	2 or 4 / 2 or 13...20
JNCT3	{#}S	Jump to S** if CT3 event flag is clear.	2 or 4 / 2 or 13...20
JNFBW	{#}S	Jump to S** if FBW event flag is clear.	2 or 4 / 2 or 13...20
JNINT	{#}S	Jump to S** if INT event flag is clear.	2 or 4 / 2 or 13...20
JNPAT	{#}S	Jump to S** if PAT event flag is clear.	2 or 4 / 2 or 13...20
JNQMT	{#}S	Jump to S** if QMT event flag is clear.	2 or 4 / 2 or 13...20
JNSE1	{#}S	Jump to S** if SE1 event flag is clear.	2 or 4 / 2 or 13...20
JNSE2	{#}S	Jump to S** if SE2 event flag is clear.	2 or 4 / 2 or 13...20
JNSE3	{#}S	Jump to S** if SE3 event flag is clear.	2 or 4 / 2 or 13...20
JNSE4	{#}S	Jump to S** if SE4 event flag is clear.	2 or 4 / 2 or 13...20
JNXFI	{#}S	Jump to S** if XFI event flag is clear.	2 or 4 / 2 or 13...20
JNXMT	{#}S	Jump to S** if XMT event flag is clear.	2 or 4 / 2 or 13...20

JNXRL {#}S	Jump to S** if XRL event flag is clear.	2 or 4 / 2 or 13...20
JNXRO {#}S	Jump to S** if XRO event flag is clear.	2 or 4 / 2 or 13...20
JPAT {#}S	Jump to S** if PAT event flag is set.	2 or 4 / 2 or 13...20
JQMT {#}S	Jump to S** if QMT event flag is set.	2 or 4 / 2 or 13...20
JSE1 {#}S	Jump to S** if SE1 event flag is set.	2 or 4 / 2 or 13...20
JSE2 {#}S	Jump to S** if SE2 event flag is set.	2 or 4 / 2 or 13...20
JSE3 {#}S	Jump to S** if SE3 event flag is set.	2 or 4 / 2 or 13...20
JSE4 {#}S	Jump to S** if SE4 event flag is set.	2 or 4 / 2 or 13...20
JXFI {#}S	Jump to S** if XFI event flag is set.	2 or 4 / 2 or 13...20
JXMT {#}S	Jump to S** if XMT event flag is set.	2 or 4 / 2 or 13...20
JXRL {#}S	Jump to S** if XRL event flag is set.	2 or 4 / 2 or 13...20
JXRO {#}S	Jump to S** if XRO event flag is set.	2 or 4 / 2 or 13...20
POLLATN {WC/WZ/WCZ}	Get ATN event flag into C/Z, then clear it.	2
POLLCT1 {WC/WZ/WCZ}	Get CT1 event flag into C/Z, then clear it.	2
POLLCT2 {WC/WZ/WCZ}	Get CT2 event flag into C/Z, then clear it.	2
POLLCT3 {WC/WZ/WCZ}	Get CT3 event flag into C/Z, then clear it.	2
POLLFBW {WC/WZ/WCZ}	Get FBW event flag into C/Z, then clear it.	2
POLLINT {WC/WZ/WCZ}	Get INT event flag into C/Z, then clear it.	2
POLLPAT {WC/WZ/WCZ}	Get PAT event flag into C/Z, then clear it.	2
POLLQMT {WC/WZ/WCZ}	Get QMT event flag into C/Z, then clear it.	2
POLLSE1 {WC/WZ/WCZ}	Get SE1 event flag into C/Z, then clear it.	2
POLLSE2 {WC/WZ/WCZ}	Get SE2 event flag into C/Z, then clear it.	2
POLLSE3 {WC/WZ/WCZ}	Get SE3 event flag into C/Z, then clear it.	2
POLLSE4 {WC/WZ/WCZ}	Get SE4 event flag into C/Z, then clear it.	2
POLLXFI {WC/WZ/WCZ}	Get XFI event flag into C/Z, then clear it.	2
POLLXMT {WC/WZ/WCZ}	Get XMT event flag into C/Z, then clear it.	2
POLLXRL {WC/WZ/WCZ}	Get XRL event flag into C/Z, then clear it.	2
POLLXRO {WC/WZ/WCZ}	Get XRO event flag into C/Z, then clear it.	2
SETPAT {#}D,{#}S	Set pin pattern for PAT event. C selects INA/INB, Z selects =/!=, D provides mask value, S provides match value.	2
SETSE1 {#}D	Set SE1 event configuration to D[8:0].	2
SETSE2 {#}D	Set SE2 event configuration to D[8:0].	2
SETSE3 {#}D	Set SE3 event configuration to D[8:0].	2
SETSE4 {#}D	Set SE4 event configuration to D[8:0].	2
WAITATN {WC/WZ/WCZ}	Wait for ATN event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITCT1 {WC/WZ/WCZ}	Wait for CT1 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITCT2 {WC/WZ/WCZ}	Wait for CT2 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITCT3 {WC/WZ/WCZ}	Wait for CT3 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITFBW {WC/WZ/WCZ}	Wait for FBW event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITINT {WC/WZ/WCZ}	Wait for INT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITPAT {WC/WZ/WCZ}	Wait for PAT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITSE1 {WC/WZ/WCZ}	Wait for SE1 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITSE2 {WC/WZ/WCZ}	Wait for SE2 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITSE3 {WC/WZ/WCZ}	Wait for SE3 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITSE4 {WC/WZ/WCZ}	Wait for SE4 event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITXFI {WC/WZ/WCZ}	Wait for XFI event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITXMT {WC/WZ/WCZ}	Wait for XMT event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITXRL {WC/WZ/WCZ}	Wait for XRL event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+
WAITXRO {WC/WZ/WCZ}	Wait for XRO event flag, then clear it. Prior SETQ sets optional CT timeout value. C/Z = timeout.	2+

Interrupt Instructions		
Instruction	**Description**	**Clocks Cog, LUT & Hub**
ALLOWI	Allow interrupts (default).	2
BRK {#}D	If in debug ISR, set next break condition to D. Else, set BRK code to D[7:0] and unconditionally trigger BRK interrupt, if enabled.	2
COGBRK {#}D	If in debug ISR, trigger asynchronous breakpoint in cog D[3:0]. Cog D[3:0] must have asynchronous breakpoint enabled.	2
GETBRK D WC/WZ/WCZ	Get breakpoint/cog status into D according to WC/WZ/WCZ. See documentation for details.	2
NIXINT1	Cancel INT1.	2
NIXINT2	Cancel INT2.	2
NIXINT3	Cancel INT3.	2
SETINT1 {#}D	Set INT1 source to D[3:0].	2
SETINT2 {#}D	Set INT2 source to D[3:0].	2
SETINT3 {#}D	Set INT3 source to D[3:0].	2
STALLI	Stall Interrupts.	2
TRGINT1	Trigger INT1, regardless of STALLI mode.	2
TRGINT2	Trigger INT2, regardless of STALLI mode.	2
TRGINT3	Trigger INT3, regardless of STALLI mode.	2

Register Indirection Instructions		
Instruction	**Description**	**Clocks Cog & LUT / Hub**
ALTB D,{#}S	Alter D field of next instruction to D[13:5].	2
ALTB D,{#}S	Alter D field of next instruction to (D[13:5] + S) & $1FF. D += sign-extended S[17:9].	2
ALTD D	Alter D field of next instruction to D[8:0].	2
ALTD D,{#}S	Alter D field of next instruction to (D + S) & $1FF. D += sign-extended S[17:9].	2
ALTGB D	Alter subsequent GETBYTE/ROLBYTE instruction. Next S field = D[10:2], N field = D[1:0].	2
ALTGB D,{#}S	Alter subsequent GETBYTE/ROLBYTE instruction. Next S field = (D[10:2] + S) & $1FF, N field = D[1:0]. D += sign-extended S[17:9].	2
ALTGN D	Alter subsequent GETNIB/ROLNIB instruction. Next S field = D[11:3], N field = D[2:0].	2
ALTGN D,{#}S	Alter subsequent GETNIB/ROLNIB instruction. Next S field = (D[11:3] + S) & $1FF, N field = D[2:0]. D += sign-extended S[17:9].	2
ALTGW D	Alter subsequent GETWORD/ROLWORD instruction. Next S field = D[9:1], N field = D[0].	2
ALTGW D,{#}S	Alter subsequent GETWORD/ROLWORD instruction. Next S field = ((D[9:1] + S) & $1FF), N field = D[0]. D += sign-extended S[17:9].	2
ALTI D	Execute D in place of next instruction. D stays same.	2
ALTI D,{#}S	Substitute next instruction's I/R/D/S fields with fields from D, per S. Modify D per S.	2
ALTR D	Alter result register address (normally D field) of next instruction to D[8:0].	2
ALTR D,{#}S	Alter result register address (normally D field) of next instruction to (D + S) & $1FF. D += sign-extended S[17:9].	2
ALTS D	Alter S field of next instruction to D[8:0].	2
ALTS D,{#}S	Alter S field of next instruction to (D + S) & $1FF. D += sign-extended S[17:9].	2
ALTSB D	Alter subsequent SETBYTE instruction. Next D field = D[10:2], N field = D[1:0].	2
ALTSB D,{#}S	Alter subsequent SETBYTE instruction. Next D field = (D[10:2] + S) & $1FF, N field = D[1:0]. D += sign-extended S[17:9].	2
ALTSN D	Alter subsequent SETNIB instruction. Next D field = D[11:3], N field = D[2:0].	2
ALTSN D,{#}S	Alter subsequent SETNIB instruction. Next D field = (D[11:3] + S) & $1FF, N field = D[2:0]. D += sign-extended S[17:9].	2
ALTSW D	Alter subsequent SETWORD instruction. Next D field = D[9:1], N field = D[0].	2
ALTSW D,{#}S	Alter subsequent SETWORD instruction. Next D field = (D[9:1] + S) & $1FF, N field = D[0]. D += sign-extended S[17:9].	2

CORDIC Solver Instructions		
Instruction	**Description**	**Clocks Cog, LUT & Hub**
GETQX D {WC/WZ/WCZ}	Retrieve CORDIC result X into D. Waits, in case result not ready. C = X[31]. [1]	2...58
GETQY D {WC/WZ/WCZ}	Retrieve CORDIC result Y into D. Waits, in case result not ready. C = Y[31]. [1]	2...58
QDIV {#}D,{#}S	Begin CORDIC unsigned division of {SETQ value or 32'b0, D} / S. GETQX/GETQY retrieves quotient/remainder.	2...9
QEXP {#}D	Begin CORDIC logarithm-to-number conversion of D. GETQX retrieves number.	2...9
QFRAC {#}D,{#}S	Begin CORDIC unsigned division of {D, SETQ value or 32'b0} / S. GETQX/GETQY retrieves quotient/remainder.	2...9
QLOG {#}D	Begin CORDIC number-to-logarithm conversion of D. GETQX retrieves log {5'whole_exponent, 27'fractional_exponent}.	2...9
QMUL {#}D,{#}S	Begin CORDIC unsigned multiplication of D * S. GETQX/GETQY retrieves lower/upper product.	2...9
QROTATE {#}D,{#}S	Begin CORDIC rotation of point (D, SETQ value or 32'b0) by angle S. GETQX/GETQY retrieves X/Y.	2...9
QSQRT {#}D,{#}S	Begin CORDIC square root of {S, D}. GETQX retrieves root.	2...9
QVECTOR {#}D,{#}S	Begin CORDIC vectoring of point (D, S). GETQX/GETQY retrieves length/angle.	2...9

[1] Z = (result == 0)

Color Space Converter and Pixel Mixer Instructions		
Instruction	**Description**	**Clocks Cog, LUT & Hub**
Color Space Converter		
SETCFRQ {#}D	Set the colorspace converter "CFRQ" parameter to D[31:0].	2
SETCI {#}D	Set the colorspace converter "CI" parameter to D[31:0].	2
SETCMOD {#}D	Set the colorspace converter "CMOD" parameter to D[8:0].	2
SETCQ {#}D	Set the colorspace converter "CQ" parameter to D[31:0].	2
SETCY {#}D	Set the colorspace converter "CY" parameter to D[31:0].	2
Pixel Mixer		
ADDPIX D,{#}S	Add bytes of S into bytes of D, with $FF saturation.	7
BLNPIX D,{#}S	Alpha-blend bytes of S into bytes of D, using SETPIV value.	7
MIXPIX D,{#}S	Mix bytes of S into bytes of D, using SETPIX and SETPIV values.	7
MULPIX D,{#}S	Multiply bytes of S into bytes of D, where $FF = 1.0 and $00 = 0.0.	7
SETPIV {#}D	Set BLNPIX/MIXPIX blend factor to D[7:0].	2
SETPIX {#}D	Set MIXPIX mode to D[5:0].	2

Lookup Table, Streamer, and Misc Instructions		
Instruction	**Description**	**Clocks Cog & LUT / Hub**
Lookup Table		
RDLUT D,{#}S/P {WC/WZ/WCZ}	Read data from LUT address {#}S/PTRx into D. C = MSB of data. *	3
SETLUTS {#}D	If D[0] = 1 then enable LUT sharing, where LUT writes within the adjacent odd/even companion cog are copied to this cog's LUT.	2
WRLUT {#}D,{#}S/P	Write D to LUT address {#}S/PTRx.	2
Streamer		
GETXACC D	Get the streamer's Goertzel X accumulator into D and the Y accumulator into the next instruction's S, clear accumulators.	2
SETXFRQ {#}D	Set streamer NCO frequency to D.	2
XCONT {#}D,{#}S	Buffer new streamer command to be issued on final NCO rollover of current command, continuing phase.	2+
XINIT {#}D,{#}S	Issue streamer command immediately, zeroing phase.	2
XSTOP	Stop streamer immediately.	2
XZERO {#}D,{#}S	Buffer new streamer command to be issued on final NCO rollover of current command, zeroing phase.	2+
Miscellaneous		
AUGD #n	Queue #n to be used as upper 23 bits for next #D occurrence, so that the next 9-bit #D will be augmented to 32 bits.	2
AUGS #n	Queue #n to be used as upper 23 bits for next #S occurrence, so that the next 9-bit #S will be augmented to 32 bits.	2
GETCT D {WC}	Get CT[31:0] or CT[63:32] if WC into D. GETCT WC + GETCT gets full CT. CT=0 on reset, CT++ on every clock. C = same.	2
GETRND WC/WZ/WCZ	Get RND into C/Z. C = RND[31], Z = RND[30], unique per cog.	2
GETRND D {WC/WZ/WCZ}	Get RND into D/C/Z. RND is the PRNG that updates on every clock. D = RND[31:0], C = RND[31], Z = RND[30], unique per cog.	2
NOP	No operation.	2
POP D {WC/WZ/WCZ}	Pop stack (K). D = K. C = K[31]. *	2
PUSH {#}D	Push D onto stack.	2
SETQ {#}D	Set Q to D. Use before RDLONG/WRLONG/WMLONG to set block transfer. Also used before MUXQ/COGINIT/QDIV/QFRAC/QROTATE/WAITxxx.	2
SETQ2 {#}D	Set Q to D. Use before RDLONG/WRLONG/WMLONG to set LUT block transfer.	2
WAITX {#}D {WC/WZ/WCZ}	Wait 2 + D clocks if no WC/WZ/WCZ. If WC/WZ/WCZ, wait 2 + (D & RND) clocks. C/Z = 0.	2 + D

SYSTEM CHARACTERISTICS

Absolute Maximum Electrical Ratings

Stresses in excess of the absolute maximum ratings can cause permanent damage to the device. These are absolute stress ratings only. Functional operation of the device is not implied at these or any other conditions in excess of those given. Exposure to absolute maximum ratings for extended periods can adversely affect device reliability.

Absolute Maximum Ratings	
Ambient temperature under bias	-40 °C to +125 °C
Storage temperature	-40 °C to +150 °C
Voltage on VDD with respect to GND	-0.3 V to +2.2 V
Voltage on Vxxyy with respect to GND	-0.3 V to +4.0 V
Voltage on all other pins with respect to GND[1]	-0.3 V to (Vxxyy + 0.3 V)
Total power dissipation	2.5 W
Max. current out of GND	4 A
Max. current into VDD pins	120 mA per pin
Max. current into Vxxyy pins	120 mA per pin
Max DC current into an input pin with internal protection diode forward biased	±10 mA
Max. allowable current per I/O pin	±30 mA
ESD Human Body Model (JS-001)	4 kV
ESD Charged Device Model (JS-002)	1 kV

[1] Note: I/O pin voltages in respect to GND may be exceeded if the internal protection diode forward bias current is not exceeded.

DC Characteristics

Operating temperature range: -40 °C to +105 °C unless otherwise noted.

DC Characteristics						
Symbol	**Parameter**	**Conditions**	**Min**	**Typ[1]**	**Max**	**Units**
V_{dd}	Core Supply Voltage		1.7	1.8	1.9	V
V_{xxyy}	VIO Supply Voltage		3.15	3.3	3.45	V
V_{ih}	Input Logic Threshold		V_{xxyy} * 0.3	V_{xxyy} * 0.5	V_{xxyy} * 0.7	V
I_{il}	Input Leakage Current	IO pin V_{in} = GND or V_{io}		±0.1	±10	µA
V_{ol}	Output Low Voltage (relative to GND)	VDD=3.3V, sinking 1mA VDD=3.3V, sinking 10mA VDD=3.3V, sinking 30mA		15 160 510		mV
Voh	Output High Voltage (relative to Vxxyy)	VDD=3.3V, sourcing 1mA VDD=3.3V, sourcing 10mA VDD=3.3V, sourcing 30mA		-6 -170 -580		mV
Iq V_{dd}	VDD Quiescent Current	RESn = TEST = P0..P64 = 0V, Vxxyy = 3.3V, VDD = 1.8V		40		µA
Iq V_{xxyy}	Vxxyy Quiescent Current	RESn = TEST = P0..P64 = 0V, Vxxyy = 3.3V, VDD = 1.8V		0.5		µA

[1] Note: Data in the Typical (“Typ”) column is T = 25 °C unless otherwise stated.

AC Characteristics

Operating temperature range: -40 °C to +105 °C unless otherwise noted.

AC Characteristics						
Symbol	Parameter	Conditions	Min	Typ[1]	Max	Units
Freq	Oscillator Frequency	RCSLOW (internal) RCFAST (internal) Direct drive (into XI) Crystal (between XI and XO) PLL (fed by direct drive or crystal)	12 20 DC 1 3.33	20 24 - - -	30 30 200 50 320	kHz MHz MHz MHz MHz
C_{in}	XI and XO pin Capacitance	Mode 0 : Disabled (1MΩ feedback resistor off) Mode 1 : Direct drive Mode 2 : Crystal ≥ 16MHz Mode 3 : Crystal < 16MHz		2 2 15 30		pF pF pF pF

[1] Note: Data in the Typical (“Typ”) column is T = 25 °C unless otherwise stated.

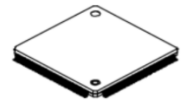
PACKAGING

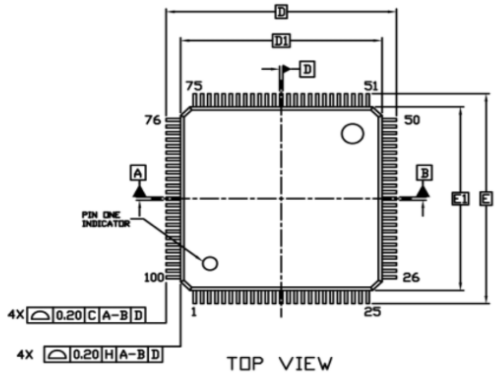
MECHANICAL CASE OUTLINE
PACKAGE DIMENSIONS

ON Semiconductor®

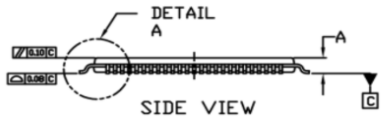
TQFP100 14x14, 0.5P
CASE 932BR
ISSUE O

DATE 03 JUL 2018





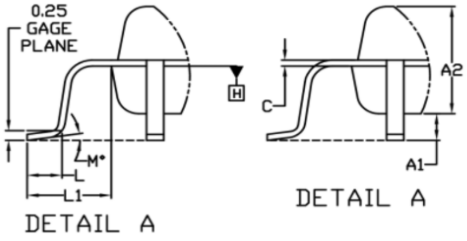
TOP VIEW



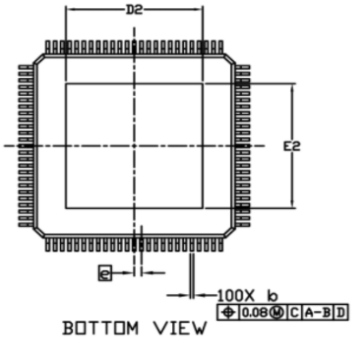
SIDE VIEW

NOTES:
1. DIMENSIONING AND TOLERANCING PER ASME Y14.5M, 1994.
2. CONTROLLING DIMENSION: MILLIMETERS
3. DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION. DAMBAR PROTRUSION SHALL BE 0.08 MAX. AT MMC. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OF THE FOOT.
4. DIMENSIONS D1 AND E1 DO NOT INCLUDE MOLD FLASH, PROTRUSIONS, OR GATE BURRS. MOLD FLASH, PROTRUSIONS, OR GATE BURRS SHALL NOT EXCEED 0.25 PER SIDE. DIMENSIONS D1 AND E1 ARE MAXIMUM PLASTIC BODY SIZE INCLUDING MOLD MISMATCH.
5. THE TOP PACKAGE BODY SIZE IS SMALLER THAN THE BOTTOM PACKAGE SIZE AND TOP PACKAGE WILL NOT OVERHANG THE BOTTOM.
6. DATUM PLANE H IS LOCATED AT THE BOTTOM MOLD PARTING LINE COINCIDENT WITH WHERE THE LEAD EXITS THE BODY.
7. DIMENSIONS D1 AND E1 TO BE DETERMINED AT DATUM PLANE H.
8. DATUMS A-B AND D ARE DETERMINED AT DATUM PLANE H.
9. A1 IS DEFINED AS THE VERTICAL DISTANCE FROM THE SEATING PLANE TO THE LOWEST POINT ON THE PACKAGE BODY.
10. DIMENSIONS D AND E TO BE DETERMINED AT DATUM PLANE C.
11. EXPOSED PAD SIZE IS AFFECTED BY MOLD FLASH AND MOLD FLASH IS CONTROLLED BY FVIGFINAL VISUAL INSPECTION0 SPEC.

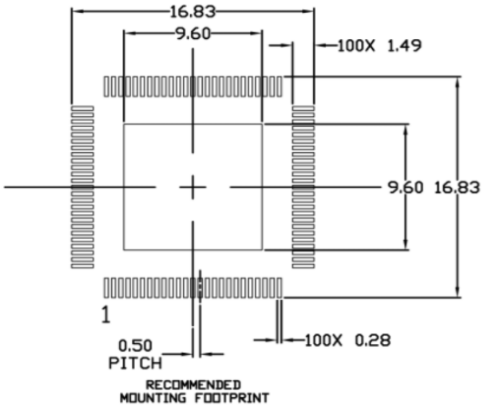
DIM	MILLIMETERS		
	MIN.	NOM.	MAX.
A	----	----	1.20
A1	0.05	----	0.15
A2	0.95	1.00	1.05
b	0.17	0.22	0.27
c		0.20 REF	
D	15.80	16.00	16.20
D1	13.80	14.00	14.20
D2		9.50 REF	
E	15.80	16.00	16.20
E1	13.80	14.00	14.20
E2		9.50 REF	
e		0.50 BSC	
L	0.45	0.60	0.75
L1		1.00 REF	
M	0°	----	7°



DETAIL A

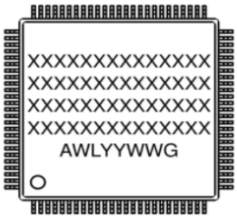


BOTTOM VIEW



RECOMMENDED
MOUNTING FOOTPRINT

GENERIC MARKING DIAGRAM*



XXX = Specific Device Code
A = Assembly Location
WL = Wafer Lot
YY = Year
WW = Work Week
G = Pb−Free Package

*This information is generic. Please refer to device data sheet for actual part marking. Pb−Free indicator, "G" or microdot " ▪", may or may not be present. Some products may not follow the Generic Marking.

DOCUMENT NUMBER:	98AON94348G	Electronic versions are uncontrolled except when accessed directly from the Document Repository. Printed versions are uncontrolled except when stamped "CONTROLLED COPY" in red.
DESCRIPTION:	TQFP100 14X14, 0.5P	PAGE 1 OF 1

www.onsemi.com

CHANGE LOG

Date	Notes
2021-05-05	First public release.
2021-05-27	Added Cog RAM, Locks, CORDIC Solver, and Smart I/O Pins sections. Updated all Hardware Connections diagrams. Corrected Hub RAM size type and clarified address type in Propeller 2 (P2X8C4M64P) RAM Memory Configuration table.
2021-07-09	Changed "Additional Documentation and Resources" to "Preface" and described this document's intention. Added Cog Attention section. In the Smart I/O Pins section, revised descriptions, included direction and state, pin registers and special instructions, added an I/O Pin Timing section, replaced the Pin Mode table with an enhanced version, and enhanced all I/O pin circuits to improve clarity. Clarified CORDIC Solver pipeline stream and throughput capability. Added Host Communication, P2 Monitor, TAQOZ, and Rebooting sections.

PARALLAX INCORPORATED

Parallax Inc.
599 Menlo Drive, Suite 100
Rocklin, CA 95765
USA

Office: +1 916-624-8333
Toll Free US: 888-512-1024

sales@parallax.com
support@parallax.com

www.parallax.com/p2
forums.parallax.com