PROPELLER 2 MEMORY

In the Propeller 2, there are two primary types of memory:

HUB MEMORY

    128K bytes of main memory shared by all cogs

        - cogs launch from this memory
        - cogs can access this memory as bytes, words, longs, and quads (4 longs)
        - $00000..$00E7F is ROM - contains Booter, SHA-256/HMAC, and Monitor
        - $00E80..$1FFFF is RAM - for application usage

COG MEMORY (8 sets)

    512 longs of register RAM for code and data usage

        - simultaneous instruction, source, and destination reading, plus writing

    256 longs of push/pop RAM for data and video usage

        - pushes are 1-clock
        - pops are 2-clock
        - video circuit can read data simultaneously and asynchronously

HUB MEMORY INSTRUCTIONS

These instructions read and write hub memory.

All instructions use D as the data conduit, except WRQUAD/RDQUAD/RDQUADC, which use the four QUAD registers. The QUADs can be mapped into cog register space using the SETQUAD instruction or kept hidden, in which case they are still useful as data conduit and as a read cache. If mapped, the QUADs overlay four contiguous cog registers which can begin at any double-even address (%xxxxxxx00). These overlaid registers can be read and written as any other registers, as well as executed.

The cached reads RDBYTEC/RDWORDC/RDLONGC/RDQUADC will do a RDQUAD if the current read address is outside of the 4-long window of the prior RDQUAD. Otherwise, they will immediately return cached data. The CACHEX instruction invalidates the cache, forcing a fresh RDQUAD next time a cached read executes.

Hub memory instructions must wait for their cog's hub cycle, which comes once every 8 clocks. The timing relationship between a cog's instruction stream and its hub cycle is generally indeterminant, causing these instructions to take varying numbers of clocks. Timing can be made determinant, though, by intentionally spacing these instructions apart so that after the first in a series executes, the subsequent hub memory instructions fall on hub cycles, making them take the minimal numbers of clocks. The trick is to write useful code to go between them.

WRBYTE/WRWORD/WRLONG/WRQUAD/RDQUAD complete on the hub cycle, making them take 1..8 clocks.

RDBYTE/RDWORD/RDLONG complete on the 2nd clock after the hub cycle, making them take 3..11 clocks.

RDBYTEC/RDWORDC/RDLONGC take only 1 clock if data is cached, otherwise 3..11 clocks.

RDQUADC takes only 1 clock if data is cached, otherwise 1..8 clocks.

After a RDQUAD, the QUAD registers are accessible via D and S on the 3rd clock and executable on the 5th clock.

```
instructions                                                                                                  clocks
-------------------------------------------------------------------------------------------------------------------
000000 000 0 CCCC DDDDDDDDD SSSSSSSSS     WRBYTE  D,S       'write lower byte in D at S                 1..8
000000 000 1 CCCC DDDDDDDDD SUPNNNNNN     WRBYTE  D,PTR     'write lower byte in D at PTR               1..8
000000 Z01 0 CCCC DDDDDDDDD SSSSSSSSS     RDBYTE  D,S       'read byte at S into D                     3..11
000000 Z01 1 CCCC DDDDDDDDD SUPNNNNNN     RDBYTE  D,PTR     'read byte at PTR into D                   3..11
000000 Z11 0 CCCC DDDDDDDDD SSSSSSSSS     RDBYTEC D,S       'read cached byte at S into D          1, 3..11
000000 Z11 1 CCCC DDDDDDDDD SUPNNNNNN     RDBYTEC D,PTR     'read cached byte at PTR into D        1, 3..11

000001 000 0 CCCC DDDDDDDDD SSSSSSSSS     WRWORD  D,S       'write lower word in D at S                 1..8
000001 000 1 CCCC DDDDDDDDD SUPNNNNNN     WRWORD  D,PTR     'write lower word in D at PTR               1..8
000001 Z01 0 CCCC DDDDDDDDD SSSSSSSSS     RDWORD  D,S       'read word at S into D                     3..11
000001 Z01 1 CCCC DDDDDDDDD SUPNNNNNN     RDWORD  D,PTR     'read word at PTR into D                   3..11
000001 Z11 0 CCCC DDDDDDDDD SSSSSSSSS     RDWORDC D,S       'read cached word at S into D          1, 3..11
000001 Z11 1 CCCC DDDDDDDDD SUPNNNNNN     RDWORDC D,PTR     'read cached word at PTR into D        1, 3..11

000010 000 0 CCCC DDDDDDDDD SSSSSSSSS     WRLONG  D,S       'write D at S                               1..8
000010 000 1 CCCC DDDDDDDDD SUPNNNNNN     WRLONG  D,PTR     'write D at PTR                             1..8
000010 Z01 0 CCCC DDDDDDDDD SSSSSSSSS     RDLONG  D,S       'read long at S into D                     3..11
000010 Z01 1 CCCC DDDDDDDDD SUPNNNNNN     RDLONG  D,PTR     'read long at PTR into D                   3..11
000010 Z11 0 CCCC DDDDDDDDD SSSSSSSSS     RDLONGC D,S       'read cached long at S into D          1, 3..11
000010 Z11 1 CCCC DDDDDDDDD SUPNNNNNN     RDLONGC D,PTR     'read cached long at PTR into D        1, 3..11

000011 000 0 CCCC DDDDDDDDD 010110000     WRQUAD  D         'write QUADs at D                           1..8
000011 001 1 CCCC SUPNNNNNN 010110000     WRQUAD  PTR       'write QUADs at PTR                         1..8
000011 000 0 CCCC DDDDDDDDD 010110001     RDQUAD  D         'read quad at D into QUADs                  1..8
000011 001 1 CCCC SUPNNNNNN 010110001     RDQUAD  PTR       'read quad at PTR into QUADs                1..8
000011 010 0 CCCC DDDDDDDDD 010110001     RDQUADC D         'read cached quad at D into QUADs       1, 1..8
000011 011 1 CCCC SUPNNNNNN 010110001     RDQUADC PTR       'read cached quad at PTR into QUADs     1, 1..8
-------------------------------------------------------------------------------------------------------------------
```

PTR expressions:

```
    INDEX = -32..+31 for simple offsets, 0..31 for ++'s, or 0..32 for --'s
    SCALE = 1 for byte, 2 for word, 4 for long, or 16 for quad

    S = 0 for PTRA, 1 for PTRB
    U = 0 to keep PTRx same, 1 to update PTRx
    P = 0 to use PTRx + INDEX*SCALE, 1 to use PTRx (post-modify)
    NNNNNN = INDEX
    nnnnnn = -INDEX
```

```
    SUPNNNNNN     PTR expression
    ------------------------------------------------------------------------------
    000000000     PTRA              'use PTRA
    100000000     PTRB              'use PTRB
    011000001     PTRA++            'use PTRA,                PTRA += SCALE
    111000001     PTRB++            'use PTRB,                PTRB += SCALE
    011111111     PTRA--            'use PTRA,                PTRA -= SCALE
    111111111     PTRB--            'use PTRB,                PTRB -= SCALE
    010000001     ++PTRA            'use PTRA + SCALE,        PTRA += SCALE
    110000001     ++PTRB            'use PTRB + SCALE,        PTRB += SCALE
    010111111     --PTRA            'use PTRA - SCALE,        PTRA -= SCALE
    110111111     --PTRB            'use PTRB - SCALE,        PTRB -= SCALE

    000NNNNNN     PTRA[INDEX]       'use PTRA + INDEX*SCALE
    100NNNNNN     PTRB[INDEX]       'use PTRB + INDEX*SCALE
    011NNNNNN     PTRA++[INDEX]     'use PTRA,                PTRA += INDEX*SCALE
    111NNNNNN     PTRB++[INDEX]     'use PTRB,                PTRB += INDEX*SCALE
    011nnnnnn     PTRA--[INDEX]     'use PTRA,                PTRA -= INDEX*SCALE
    111nnnnnn     PTRB--[INDEX]     'use PTRB,                PTRB -= INDEX*SCALE
    010NNNNNN     ++PTRA[INDEX]     'use PTRA + INDEX*SCALE,  PTRA += INDEX*SCALE
    110NNNNNN     ++PTRB[INDEX]     'use PTRB + INDEX*SCALE,  PTRB += INDEX*SCALE
    010nnnnnn     --PTRA[INDEX]     'use PTRA - INDEX*SCALE,  PTRA -= INDEX*SCALE
    110nnnnnn     --PTRB[INDEX]     'use PTRB - INDEX*SCALE,  PTRB -= INDEX*SCALE
```

Examples:

```
000000 Z01 1 CCCC DDDDDDDDD 000000000     RDBYTE  D,PTRA         'read byte at PTRA into D
000001 000 1 CCCC DDDDDDDDD 111000001     WRWORD  D,PTRB++       'write lower word in D at PTRB,      PTRB += 2
000010 Z01 1 CCCC DDDDDDDDD 011111111     RDLONG  D,PTRA--       'read long at PTRA into D,           PTRA -= 4
000011 001 1 CCCC 110000001 010110001     RDQUAD  ++PTRB         'read quad at PTRB+16 into QUADs,    PTRB += 16
000000 000 1 CCCC DDDDDDDDD 010111111     WRBYTE  D,--PTRA       'write lower byte in D at PTRA-1,    PTRA -= 1

000001 000 1 CCCC DDDDDDDDD 100000111     WRWORD  D,PTRB[7]      'write lower word in D to PTRB+7*2
000010 Z11 1 CCCC DDDDDDDDD 011001111     RDLONGC D,PTRA++[15]   'read cached long at PTRA into D,    PTRA += 15*4
000011 001 1 CCCC 111111101 010110000     WRQUAD  PTRB--[3]      'write QUADs at PTRB,                PTRB -= 3*16
000000 000 1 CCCC DDDDDDDDD 010000110     WRBYTE  D,++PTRA[6]    'write lower byte in D to PTRA+6*1,  PTRA += 6*1
000001 Z01 1 CCCC DDDDDDDDD 110110110     RDWORD  D,--PTRB[10]   'read word at PTRB-10*2 into D,      PTRB -= 10*2
```

Bytes, words, longs, and quads are addressed as follows:

```
    for WRBYTE/RDBYTE/RDBYTEC, address = %XXXXXXXXXXXXXXXXX (bits 16..0 are used)
    for WRWORD/RDWORD/RDWORDC, address = %XXXXXXXXXXXXXXXX- (bits 16..1 are used)
    for WRLONG/RDLONG/RDLONGC, address = %XXXXXXXXXXXXXXX-- (bits 16..2 are used)
    for WRQUAD/RDQUAD/RDQUADC, address = %XXXXXXXXXXXXX---- (bits 16..4 are used)
```

address	**byte**	**word**	**long**	**quad**
00000-	50	*7250	*706F7250	*0C7CCC030C7C200020302E32706F7250
00001-	72	7250	706F7250	0C7CCC030C7C200020302E32706F7250
00002-	6F	*706F	706F7250	0C7CCC030C7C200020302E32706F7250
00003-	70	706F	706F7250	0C7CCC030C7C200020302E32706F7250
00004-	32	*2E32	*20302E32	0C7CCC030C7C200020302E32706F7250
00005-	2E	2E32	20302E32	0C7CCC030C7C200020302E32706F7250
00006-	30	*2030	20302E32	0C7CCC030C7C200020302E32706F7250
00007-	20	2030	20302E32	0C7CCC030C7C200020302E32706F7250
00008-	00	*2000	*0C7C2000	0C7CCC030C7C200020302E32706F7250
00009-	20	2000	0C7C2000	0C7CCC030C7C200020302E32706F7250
0000A-	7C	*0C7C	0C7C2000	0C7CCC030C7C200020302E32706F7250
0000B-	0C	0C7C	0C7C2000	0C7CCC030C7C200020302E32706F7250
0000C-	03	*CC03	*0C7CCC03	0C7CCC030C7C200020302E32706F7250
0000D-	CC	CC03	0C7CCC03	0C7CCC030C7C200020302E32706F7250
0000E-	7C	*0C7C	0C7CCC03	0C7CCC030C7C200020302E32706F7250
0000F-	0C	0C7C	0C7CCC03	0C7CCC030C7C200020302E32706F7250
00010-	45	*FE45	*0DC1FE45	*0D7CC6010C7CC6010CFCB6E30DC1FE45
00011-	FE	FE45	0DC1FE45	0D7CC6010C7CC6010CFCB6E30DC1FE45
00012-	C1	*0DC1	0DC1FE45	0D7CC6010C7CC6010CFCB6E30DC1FE45
00013-	0D	0DC1	0DC1FE45	0D7CC6010C7CC6010CFCB6E30DC1FE45
00014-	E3	*B6E3	*0CFCB6E3	0D7CC6010C7CC6010CFCB6E30DC1FE45
00015-	B6	B6E3	0CFCB6E3	0D7CC6010C7CC6010CFCB6E30DC1FE45
00016-	FC	*0CFC	0CFCB6E3	0D7CC6010C7CC6010CFCB6E30DC1FE45
00017-	0C	0CFC	0CFCB6E3	0D7CC6010C7CC6010CFCB6E30DC1FE45
00018-	01	*C601	*0C7CC601	0D7CC6010C7CC6010CFCB6E30DC1FE45
00019-	C6	C601	0C7CC601	0D7CC6010C7CC6010CFCB6E30DC1FE45
0001A-	7C	*0C7C	0C7CC601	0D7CC6010C7CC6010CFCB6E30DC1FE45
0001B-	0C	0C7C	0C7CC601	0D7CC6010C7CC6010CFCB6E30DC1FE45
0001C-	01	*C601	*0D7CC601	0D7CC6010C7CC6010CFCB6E30DC1FE45
0001D-	C6	C601	0D7CC601	0D7CC6010C7CC6010CFCB6E30DC1FE45
0001E-	7C	*0D7C	0D7CC601	0D7CC6010C7CC6010CFCB6E30DC1FE45
0001F-	0D	0D7C	0D7CC601	0D7CC6010C7CC6010CFCB6E30DC1FE45

* new word/long/quad

PTRA/PTRB INSTRUCTIONS

Each cog has two 17-bit pointers, PTRA and PTRB, which can be read, written, modified, and used to access hub memory.

On cog startup, the PTRA and PTRB registers are initialized as follows:

    PTRA = %X_XXXXXXXX_XXXXXXXX, data from launching cog, usually a pointer
    PTRB = %X_XXXXXXXX_XXXXXX00, long address in hub where cog code was loaded from

instructions			clocks
000011 ZCR 1 CCCC DDDDDDDDD 000010010	GETPTRA D	'get PTRA into D	1
000011 ZCR 1 CCCC DDDDDDDDD 000010011	GETPTRB D	'get PTRB into D	1
000011 000 1 CCCC DDDDDDDDD 010110010	SETPTRA D	'set PTRA to D	1
000011 001 1 CCCC nnnnnnnnn 010110010	SETPTRA #n	'set PTRA to 0..511	1
000011 000 1 CCCC DDDDDDDDD 010110011	SETPTRB D	'set PTRB to D	1
000011 001 1 CCCC nnnnnnnnn 010110011	SETPTRB #n	'set PTRB to 0..511	1
000011 000 1 CCCC DDDDDDDDD 010110100	ADDPTRA D	'add D into PTRA	1
000011 001 1 CCCC nnnnnnnnn 010110100	ADDPTRA #n	'add 0..511 into PTRA	1
000011 000 1 CCCC DDDDDDDDD 010110101	ADDPTRB D	'add D into PTRB	1
000011 001 1 CCCC nnnnnnnnn 010110101	ADDPTRB #n	'add 0..511 into PTRB	1
000011 000 1 CCCC DDDDDDDDD 010110110	SUBPTRA D	'subtract D from PTRA	1
000011 001 1 CCCC nnnnnnnnn 010110110	SUBPTRA #n	'subtract 0..511 from PTRA	1
000011 000 1 CCCC DDDDDDDDD 010110111	SUBPTRB D	'subtract D from PTRB	1
000011 001 1 CCCC nnnnnnnnn 010110111	SUBPTRB #n	'subtract 0..511 from PTRB	1

QUAD-RELATED INSTRUCTIONS

Each cog has four QUAD registers which form a 128-bit conduit between the hub memory and the cog. This conduit can transfer four longs every 8 clocks via the WRQUAD/RDQUAD instructions. It can also be used as a 4-long/8-word/16-byte read cache, utilized by RDBYTEC/RDWORDC/RDLONGC/RDQUADC.

Initially hidden, these QUAD registers are mappable into cog register space by using the SETQUAD instruction to set a double-even address (%xxxxxxx00) where the base register is to appear, with the other three registers following. To hide the QUAD registers, use SETQUAD to set an address which is not double-even.

instructions			clocks
000011 000 1 CCCC 000000000 000001000	CACHEX	'invalidate cash	1
000011 Z01 1 CCCC DDDDDDDDD 000010001	GETTOPS D	'get top bytes of QUADs into D	1
000011 000 1 CCCC DDDDDDDDD 011100010	SETQUAD D	'set QUAD start address to D	1
000011 001 1 CCCC nnnnnnnnn 011100010	SETQUAD #n	'set QUAD start address to 0..511	1