General Purpose I/O Pins

The *gpio* pins are named PA0-PA7, PB0-PB7, PD0-PD5, and PE0-PE7 in IsoMax.[1] Three of these pins control the on-board LEDs, and have synonyms REDLED, GRNLED, and YELLED. The *gpio* pins can be used for programmed single-bit I/O. Ports A and B can also be used for byte-wide I/O.

Bitwise Programmed I/O

The following operations perform simple programmed I/O on single port bits:

ON
OFF
TOGGLE

These words make the designated pin an output. ON outputs a logic high level. OFF outputs a logic low level. TOGGLE inverts the level that was last output on this pin. Examples:

```
PA0 ON
PB2 OFF
PB2 TOGGLE
```

SET

This word makes the designated pin an output, and then outputs a high or low level depending on its argument. A true value (nonzero) will output a logic high. A false value (zero) outputs a logic low. Examples:

```
1 PE1 SET          ...outputs a logic high level
8000 PD0 SET       ...also outputs a logic high level
0 PD1 SET          ...outputs a logic low level
```

ON?
OFF?

These words make the designated pin an input, and then check its current logic level.
ON? returns true (-1) if the pin is a logic high, false (0) if the pin is a logic low.
OFF? returns true (-1) if the pin is a logic low, false (0) otherwise.
OFF? is the logical inverse of ON? Examples:

```
PA3 ON?            ...returns true if PA3 pin is high
PB1 OFF?           ...returns true if PB1 pin is low
```

?ON
?OFF

These words are similar to ON? and OFF? except that they do not make the pin an input. If the pin was already an input, they return true or false depending on the pin's logic level. If the pin is an output, these words return true or false depending on the value last output. For example, you can use

```
PB2 TOGGLE  PB2 ?ON
```

to invert the state of the PB2 output pin, and then report the new state of the pin.

Bytewise Programmed I/O

Pins PA0-PA7 and PB0-PB7 can also be used as byte input or byte output ports. When they are used in this manner, they are named PORTA and PORTB, and the following operations are available:

PUTBYTE

This word makes all eight pins of the designated port outputs, and then outputs its argument value to that port. Only the low 8 bits of the argument are used. For example,

```
HEX 1234 PORTA PUTBYTE
```

outputs the value 34 hex (the low eight bits) to the eight pins PA0-PA7.

---

[1] Please note that some of these pins control on-board functions, and are not available on any connector.

GETBYTE This word makes all eight pins of the designated port inputs, and then returns the 8-bit value read from that port. The high 8 bits of the result are returned as zero. Example:

```
PORTB GETBYTE
```

SPI I/O Port

The IsoPod has a single Serial Peripheral Interface port, named SPI0 in IsoMax.[2] This port uses pins PE4-PE7, as follows:

PE4 = SCLK (Serial Clock)
PE5 = MOSI (Master Out Slave In)
PE6 = MISO (Master In Slave Out)
PE7 = SS\ (Slave Select)

Important Note: using pins PE4-PE7 for programmed I/O, as described previously, will *override* the SPI port. This might be useful, for instance, to use PE7 as a programmed output when the IsoPod is acting as an SPI Master (and therefore does not require the Slave Select input). However, if you inadvertently use one of the SPI pins for bit I/O, you will not be able to use the SPI functions until you reset the IsoPod.

The following *spi* operations are available:

BITS Specifies the number of bits to be transmitted/received on the designated SPI port. Values from 2 to 16 decimal may be specified.

MSB-FIRST
LSB-FIRST Specifies the order in which bits are transmitted or received. Most SPI peripheral devices expect data to be sent MSB first.

ACTIVE-HIGH
ACTIVE-LOW These words control the value of the CPOL bit in the designated SPI port.

ACTIVE-HIGH sets CPOL=0, which outputs or detects a low-to-high transition as the "leading edge" of a clock pulse (hence the pulse is "active high").

ACTIVE-LOW sets CPOL=1, which outputs or detects a high-to-low transition as the "leading edge" of a clock pulse (hence the pulse is "active high").

Refer to the Motorola documentation for more information about clock phase and polarity.

LEADING-EDGE
TRAILING-EDGE These words control the value of the CPHA bit in the designated SPI port.

LEADING-EDGE sets CPHA=0, which causes receive data to be captured by master and slave on the first (leading) edge of the clock pulse. When CPHA=0, the SCLK signal remains inactive for the first half of the first SCLK cycle.

TRAILING-EDGE sets CPHA=1, which causes receive data to be captured by master and slave on the second (trailing) edge of the clock pulse. When CPHA=1, the first SCLK cycle begins with an edge on the SCLK line from its inactive to its active level.

Refer to the Motorola documentation for more information about clock phase and polarity.

[2] Subsequent numbers, e.g. SPI1, SPI2, etc., are reserved for future versions of this product.

MBAUD — This specifies the data rate of the designated SPI port. It accepts an integer argument which may have one of four values: 1, 2, 5, or 20. These values will set the SPI clock rate to 1.25, 2.5, 5, or 20 MHz, respectively. Any other values will be ignored. For example,

```
5 SPI0 MBAUD        ...sets SPI port to operate at 5 MHz
```

MASTER — This activates the SPI port as an SPI Master. MISO is an input, and MOSI and SCLK are outputs. SS\ is ignored. All configuration values should be specified before using MASTER. No data will be transmitted or received until MASTER or SLAVE is specified.

SLAVE — *not yet defined*

TX-SPI — Transmits one word over the SPI port. It expects a 16-bit argument.

If the port is configured as MASTER, this word will be transmitted immediately (or as soon as the previous word is transmitted, if it has not yet finished).

RX-SPI — Receives one word from the SPI port. It returns a 16-bit value.

If the port is configured as MASTER, then a word will be received on the SPI port *every time* a word is transmitted (with TX-SPI), and *only* when a word is transmitted. This data will be received regardless of whether a slave is connected or selected. RX-SPI will wait for this data word to be received, which will also be when the TX-SPI data has been sent. This is a convenient way to determine when the transmission is finished, so the software can disable the slave select line (if necessary).

ADC Input Pins

The *adcs* input pins are named ADC0-ADC7 in IsoMax. These pins can only be used for analog-to-digital conversion.

Analog-to-Digital Conversion

ANALOGIN — Starts an analog conversion on the designated input. This returns a result in the range 0-32760 decimal (0-7FF8 hex). For example,

```
ADC0 ANALOGIN .        ...measures the voltage on pin ADC0 and prints the result.
```

Note that the low 3 bits of the result are always zero. The ADC produces a 12-bit result, which is scaled to a 15-bit unsigned value.

Timer I/O Pins

The *timer* I/O pins are named TA0-TA3, TB0-TB3, TC0-TC1, and TD0-TD2 in IsoMax.[3] Please note that TAx and TBx are dual-function pins, and have different names on the I/O connector:

TA0 = PHASEA0
TA1 = PHASEB0
TA2 = INDEX0
TA3 = HOME0
TB0 = PHASEA1
TB1 = PHASEB1
TB2 = INDEX1
TB3 = HOME1

These pins can be used for programmed I/O, PWM output, and PWM input.

Programmed I/O

The following operations perform simple programmed I/O:

ON
OFF
TOGGLE

These words make the designated pin an output. ON outputs a logic high level. OFF outputs a logic low level. TOGGLE inverts the level that was last output on this pin. Examples:

```
TA0 ON
TB2 OFF
TB2 TOGGLE
```

SET

This word makes the designated pin an output, and then outputs a high or low level depending on its argument. A true value (nonzero) will output a logic high. A false value (zero) outputs a logic low. Examples:

```
1 TC1 SET          ...outputs a logic high level
8000 TD0 SET       ...also outputs a logic high level
0 TD1 SET          ...outputs a logic low level
```

ON?
OFF?

These words make the designated pin an input, and then check its current logic level.
ON? returns true (-1) if the pin is a logic high, false (0) if the pin is a logic low.
OFF? returns true (-1) if the pin is a logic low, false (0) otherwise.
OFF? is the logical inverse of ON? Examples:

```
TA3 ON?            ...returns true if TA3 (HOME0) pin is high
TB1 OFF?           ...returns true if TB1 (PHASEB1) pin is low
```

?ON
?OFF

These words are similar to ON? and OFF? except that they do not make the pin an input. If the pin was already an input, they return true or false depending on the pin's logic level. If the pin is an output, these words return true or false depending on the value last output. For example, you can use

```
TB2 TOGGLE  TB2 ?ON
```

to invert the state of the TB2 output pin, and then report the new state of the pin.

[3] Timer TD3 is used internally by IsoMax and is not available to the user.

PWM Output

The following operations output a pulse-width-modulated (PWM) square wave on the designated pin:

`PWM-PERIOD` This specifies the period of the PWM square wave. It accepts an integer argument in the range 256-65535 decimal (100-FFFF hex). This integer is multiplied by 0.4 usec to determine the period of the square wave. For example,

```
DECIMAL 50000 TA0 PWM-PERIOD
```

will produce a period of 50,000 * 0.4 = 20,000 usec (20 msec), corresponding to a 50 Hz square wave. The lowest frequency that can be achieved is 38 Hz, corresponding to a period of 65,535. `PWM-PERIOD` must be specified before `PWM-OUT` is used.

`PWM-OUT` This specifies the duty cycle of the PWM square wave. It accepts an integer argument in the range 0-65535 decimal (0-FFFF hex). 0 outputs a 0% duty cycle (always off) and FFFF outputs a 100% duty cycle (always on). For example,

```
HEX 4000 TA0 PWM-OUT
```

will output a square wave with a 25% duty cycle (25% on, 75% off). The duty cycle parameter does not depend on the PWM period; that is, hex 4000 will produce a 25% duty cycle regardless of what period is specified.

`PWM-OUT` makes the designated pin an output and begins outputting the PWM signal. No square wave will be output until `PWM-OUT` is used.

`ACTIVE-HIGH` `ACTIVE-LOW` These words control the polarity of the PWM output. When `ACTIVE-HIGH` is specified, the "on" state for the PWM output is a logic high level. When `ACTIVE-LOW` is specified, the "on" state is a logic low level. For example,

```
TA0 ACTIVE-LOW  HEX 4000 TA0 PWM-OUT
```

will output a square wave that is 25% low and 75% high. `ACTIVE-HIGH` is the default after a reset.

Example of use

An RC servo requires a 50 Hz (20 msec) PWM waveform with an on time varying from 1 msec to 2 msec, on output pin PHASEA0 (TA0). We wish to control this with an integer parameter from 0 to 100.

The desired range of on-time corresponds to a duty cycle varying from 5% to 10%. 5% of 65535 decimal is 3277, and 10% is 6554. We need to add a fraction of 3277 to the base value of 3277. This is easily accomplished with the */ scaling operator:

```
DECIMAL
: SET-RC-SERVO ( n -- )
   3277 100 */       ( scale 0-100 to 0-3277 )
   3277 +            ( add base value of 3277 )
   TA0 PWM-OUT       ( output that duty cycle, 5-10% of 65535 )
;

50000 TA0 PWM-PERIOD   ( set period to 20 msec )
TA0 ACTIVE-HIGH

50 SET-RC-SERVO     ( output 50% of range, 1.5 msec pulse width )
```

PWM Input

The following operations measure (input) a pulse-width-modulated (PWM) square wave on the designated pin:

`SET-PWM-IN` This makes the designated pin an input, and then starts measurement of pulse width. After `SET-PWM-IN` the pin will wait for an "active" transition, and then wait for an "inactive" transition (see `ACTIVE-HIGH` and `ACTIVE-LOW`). The time that the input is active will be measured. `SET-PWM-IN` measures a *single* pulse.

After issuing SET-PWM-IN, you should poll the timer with CHK-PWM-IN until it returns a nonzero value. This nonzero value is the pulse width. No pulse width will be reported until both a leading edge and a trailiing edge have been seen.

`CHK-PWM-IN` While the designated input pin is waiting for a pulse (or while it is measuring a pulse), this will return zero. After a pulse has been measured, this will return the width of the pulse, as an integer in the range 0-65535 decimal (0-FFFF hex). This integer is multiplied by 0.4 usec to determine the duration of the pulse.

Note: only the first nonzero value returned by CHK-PWM-IN will be valid. Subsequent calls to `CHK-PWM-IN` may return the same value, but may not, depending on whether additional pulses are seen on the input. Subsequent values will not be valid. To measure another pulse, you *must* reissue the command `SET-PWM-IN` to reset the counter.

The largest width that can be measured is 65,535 * 0.4 usec, or 26.214 msec. Pulse widths wider than this will give unpredictable results.

`ACTIVE-HIGH`
`ACTIVE-LOW` These words control the polarity of the PWM *input*. When `ACTIVE-HIGH` is specified, `SET-PWM-IN` will measure the time that the input is a logic high (i.e., the width of the high level). Measurement begins at the low-to-high transition, and ends at the high-to-low transition.

When `ACTIVE-LOW` is specified, it will measure the time that the input is a logic low.

Example of use

An RC servo control signal with an on time varying from 1 msec to 2 msec is connected to pin TC1. We wish to measure the on-time in microseconds (1000 to 2000).

`CHK-PWM-IN` will return an on-time in units of 0.4 usec. We need to scale this by 4/10 to get a result in microseconds. This is easily accomplished with the */ scaling operator:

```
DECIMAL
: GET-RC-SERVO ( -- n )
   TC1 ACTIVE-HIGH
   TC1 SET-PWM-IN
   BEGIN TC1 CHK-PWM-IN ?DUP UNTIL   ( wait for nonzero value )
   4 10 */                           ( scale by 4/10 )
;

GET-RC-SERVO .     ( measure pulse width and print result )
```

PWM Output Pins

The *pwmout* pins are named PWMA0-PWMA5 and PWMB0-PWMB5 in IsoMax. These pins can be used for programmed output or PWM output. These pins cannot be programmed as inputs.

Programmed Output

The following operations perform simple programmed output:

ON
OFF
TOGGLE

These words make the designated pin an output. ON outputs a logic high level. OFF outputs a logic low level. TOGGLE inverts the level that was last output on this pin. Examples:

```
PWMA0 ON
PWMB2 OFF
PWMB2 TOGGLE
```

SET

This word makes the designated pin an output, and then outputs a high or low level depending on its argument. A true value (nonzero) will output a logic high. A false value (zero) outputs a logic low. Examples:

```
1 PWMB1 SET         ...outputs a logic high level
8000 PWMA0 SET      ...also outputs a logic high level
0 PWMA1 SET         ...outputs a logic low level
```

?ON
?OFF

These words return the current state of the designated PWM output pin. They do *not* make the pin an input; they simply reflect the level currently being output on the pin. For example, you can use

```
PWMB2 TOGGLE  PWMB2 ?ON
```

to invert the state of the PWMB2 output pin, and then report the new state of the pin.

PWM Output

The following operations output a pulse-width-modulated (PWM) square wave on the designated pin:

PWM-PERIOD

This specifies the period of the PWM square wave. It accepts an integer argument in the range 256-32767 decimal (100-7FFF hex). This integer is multiplied by 0.4 usec to determine the period of the square wave. For example,

```
DECIMAL 25000 TA0 PWM-PERIOD
```

will produce a period of 25,000 * 0.4 = 10,000 usec (10 msec), corresponding to a 100 Hz square wave.

Note that while the scaling of these values is identical to that for the Timer I/O pins -- 0.4 usec per count -- the allowable range is smaller, with a maximum of 32,767. This means that on the *pwmout* pins, the lowest frequency that can be achieved is 76.3 Hz, corresponding to a period of 32,767.

Note especially that PWM period is controlled in groups of six pins. All of the PWMAx pins must have the same period. So, if you specify PWM-PERIOD for *any* PWMAx pin, it will affect *all* of the PWMAx pins. Likewise for the six PWMBx pins. PWM-PERIOD must be specified for *at least* one of the six pins in a group, before PWM-OUT is used for any of those six pins. PWM-PERIOD need only be specified once for all six pins in the group.

You may use some pins in the group for programmed output while others are outputting PWM.

PWM-OUT — This specifies the duty cycle of the PWM square wave. It accepts an integer argument in the range 0-65535 decimal (0-FFFF hex). 0 outputs a 0% duty cycle (always off) and FFFF outputs a 100% duty cycle (always on). For example,

```
HEX 4000 PWMA0 PWM-OUT
```

will output a square wave with a 25% duty cycle (25% on, 75% off). The duty cycle parameter does not depend on the PWM period; that is, hex 4000 will produce a 25% duty cycle regardless of what period is specified.

PWM-OUT begins outputting the PWM signal. No square wave will be output until PWM-OUT is used.

ACTIVE-HIGH
ACTIVE-LOW — *not yet defined*

PWM Input Pins

Pins FAULTA0-FAULTA3, FAULTB0-FAULTB3, ISA0-ISA2, and ISB0-ISB2, associated with the PWM generator, are treated by IsoMax as general-purpose input pins. These *pwmin* pins cannot be programmed as outputs.

Programmed Input

The following operations perform simple programmed input:

ON?
OFF? — These words make the check the current logic level of the designated pin.
ON? returns true (-1) if the pin is a logic high, false (0) if the pin is a logic low.
OFF? returns true (-1) if the pin is a logic low, false (0) otherwise.
OFF? is the logical inverse of ON? Examples:

```
FAULTA3 ON?        ...returns true if FAULTA3 pin is high
ISB1 OFF?          ...returns true if ISB1 pin is low
```

?ON
?OFF — Since the *pwmin* pins are permanently configured as inputs, these words are identical to ON? and OFF? for these pins.