Co-Pro-M48 Microprocessor Co-Processor

Features

High Performance. Low Power AVR 8-Bit Microcontroller Co-Processor
Speed - 20MHz

I/O and Package

- **18 Programmable I/O Lines**
- **6 – Special function high latching input pins**
- **6 – Special function low latching input pins**
- **Internal pull-ups option**
- **28-pin PDIP**

Interface

- **Synchronous serial communication**
- **Shiftin and Shiftout commands from any Basic Stamp**

Peripheral Features

- **Real Time 24 Hour Seconds Minutes and Hours counter.**
- **6 - Channel 8 Bit ADC – VCC Reference and 1.1V internal Reference**

Memories

- **256 byte SRAM buffer Unlimited Write/Erase Cycles**
- **256 byte EEPROM – 100,000 Write/Erase Cycles**
- **6 – Banks of 256 Byte FLASH memory storage 10,000 Write/Erase Cycles**

Operating Voltage:

- **2.7 – 5.5V**

Temperature Range:

- **-40C to 85C**

Low Power Sleep Mode

Co-Pro-M48 Command Reference

_init	Initialize Co-Processor
_errors	Get error flags
_low	Set pin (0…17) low
_high	Set pin (0…17) high
_input	Set pin (0…17) to input
_inputp	Set pin (0..17) to input with pullup
_push	Push a byte on the internal 256 byte SRAM buffer
_pop	Pop a byte from the internal 256 byte SRAM buffer
_zstack	Set the SRAM stack pointer to Zero (0)
_clrbl	Clear the B-Latching pins (pins 6…11)
_clrcl	Clear the C-Latching pins (pins 12…17)
_rpinsa	Read pins (0…5)
_rpinsb	Read pins (6…11)
_rpinsc	Read pins (12…18)
_pinsbl	Read the B-Latching pins
_pinscl	Read the C-Latching pins
_seconds	Read seconds (0…59)
_minutes	Read minutes (0…59)
_hours	Read hours (0..23)
_time	Read seconds, minutes, hours
_stime	Set Seconds, Minutes, Hours
_adc0	Read adc0 – 1.1V Reference
_adc1	Read adc1 – 1.1V Reference
_adc2	Read adc2 – 1.1V Reference
_adc3	Read adc3 – 1.1V Reference
_adc4	Read adc4 – 1.1V Reference
_adc5	Read adc5 – 1.1V Reference
_adcvcc0	Read adc0 – VCC Reference
_adcvcc1	Read adc1 – VCC Reference
_adcvcc2	Read adc2 – VCC Reference
_adcvcc3	Read adc3 – VCC Reference
_adcvcc4	Read adc4 – VCC Reference
_adcvcc5	Read adc5 – VCC Reference
_writeflash	Write 256 bytes from the SRAM buffer to FLASH
_readflash	Read 256 bytes from FLASH memory to the SRAM buffer
_readsram	Read a byte at the current SRAM, increment pointer
_setstack	Set the SRAM pointer to a specific location
_ewrite	Write a byte at a specified address to EEPROM
_eread	Read a byte from a specified address from EEPROM

Initialize Coprocessor

Byte Code: $FFAA

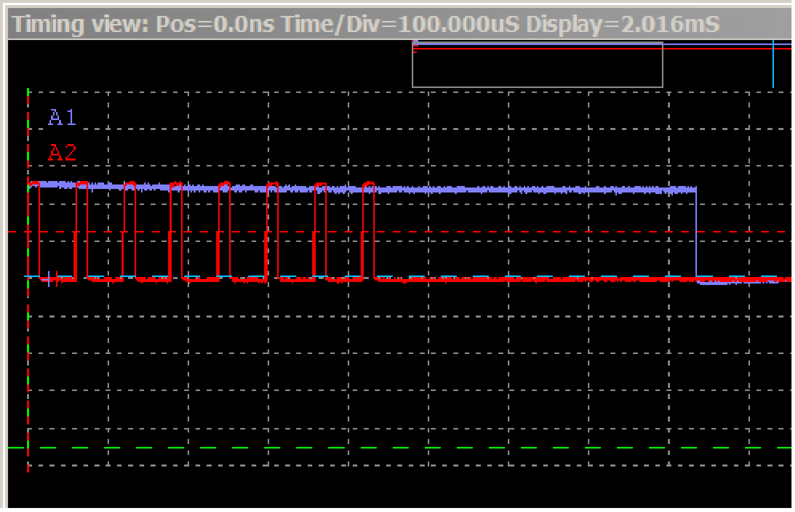
Function

Sets the internal timer width to read 8 bits.

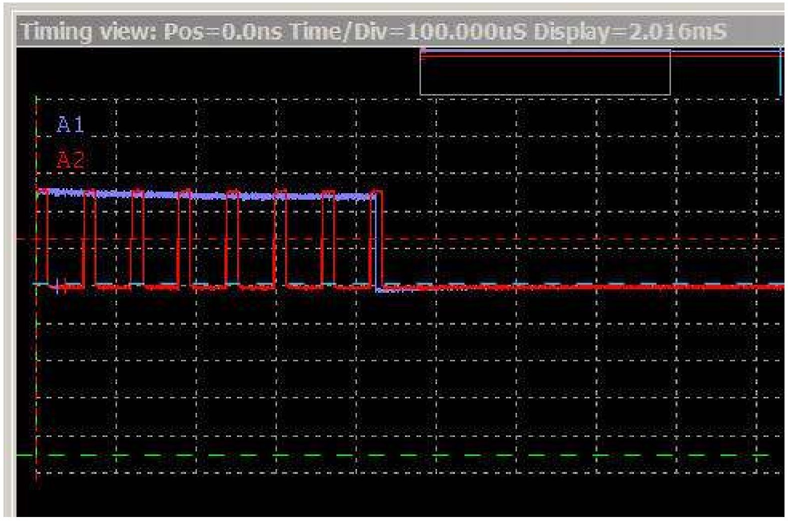
Explanation

The Coprocessor receives data using a clock input and a data input. Eight bits of data are required for each command. The Basic Stamp 2, sends eight bits of data using the shiftout command up to 16 kBits/Sec. Other stamps can send data faster. This command synchronizes the timeout error control.

Below is the default timeout when the Coprocessor is started (blue line) of 833uS. If less than 7 bits are received the co-processor will timeout. If 8 bits are received the timeout clock is stopped and the data is processed.


Timing view: Pos=0.0ns Time/Div=100.000uS Display=2.016mS

Below is the timeout after initialization (Blue line). It has been shortened to the data rate of 8 bits. This allows any errors in sending data to be processed faster than an arbitrary timeout. Error flags for timeout conditions can be read with the ERROR command.

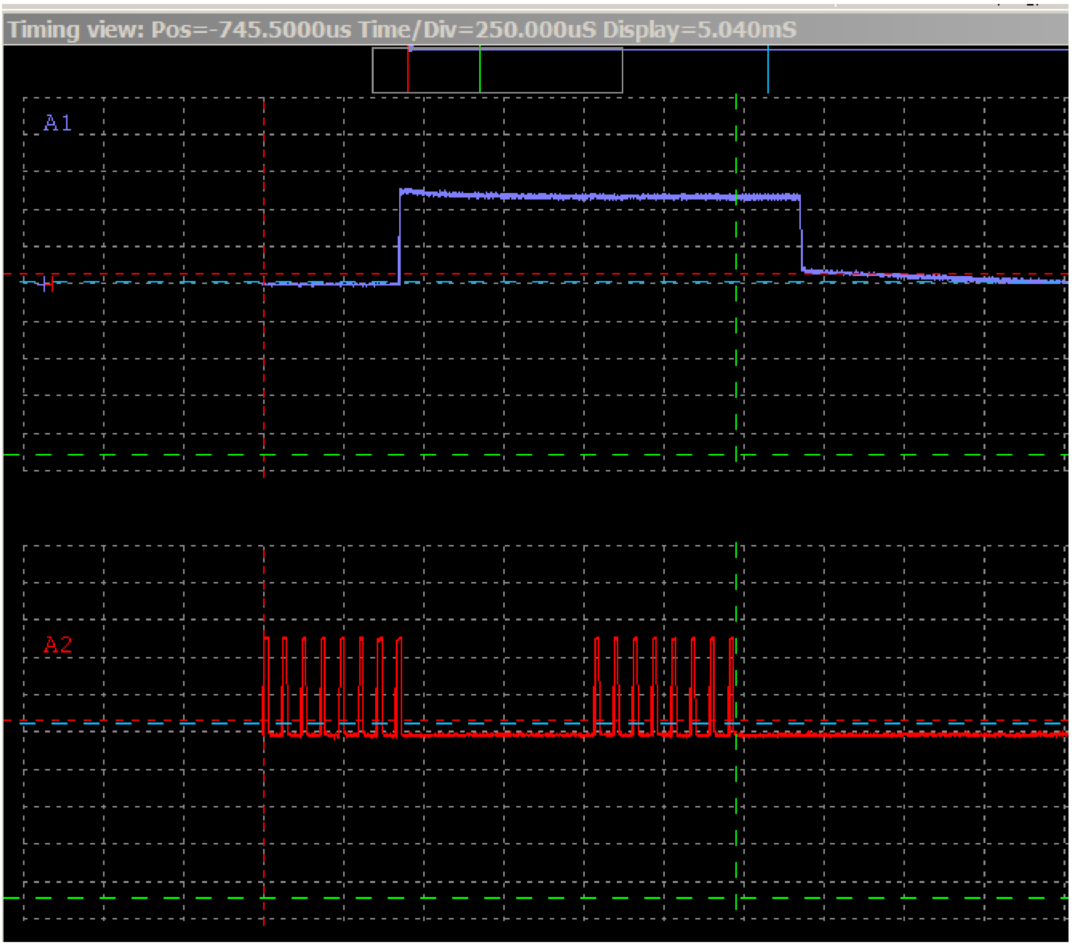


Basic Stamp Example:

```
Shiftout Dpin,Cpin,1, [$FFAA\16]
```

Timing window for receiving data from co-processor

When receiving data from the co-processor the Basic Stamp uses the shiftin command. For every byte requested a 1.2 mS timer is set to time-out the request if 8 bits are not clocked out within the time window. Below, the blue line shows the time-out window. If 8 bits are not clocked out of the co-processor within this window the data exchange is terminated for that byte and the co-processor returns to input mode ready to accept new commands. Multi byte reads will also terminate in a time-out condition. The Time-out window starts with the last bit read of the preceding command that requested the data.


Timing view: Pos=-745.5000us Time/Div=250.000uS Display=5.040mS
A1
A2

Set Pin to Output Low

Byte Code: **$B0** *Pin*

Variables: *Pin*

Pin is a one BYTE value 0 through 17. It can be a BYTE variable or a constant.

Function

Sets any of the Coprocessors 18 pins to Output LOW

Explanation:

The Coprocessor has 18 pins that can be set four different modes.

Input without Pullup
Input with Pullup
Output Low
Output High

Basic Stamp Examples:

```
Shiftout Dpin, Cpin, 1, [$B0, 0]       ‘Sets the Coprocessors pin 0 to Output Low

‘Using the 16 bit shiftout command with the optional 16 bit command.

Shiftout Dpin, Cpin, 1, [$B000\16,$B001\16]  ‘Set pins 0 and 1 to output low

‘Using shiftout with an 8 bit BYTE variable.

ledpin         VAR   BYTE

For ledpin=0 to 17
 Shiftout Dpin, Cpin, 1, [$B0, ledpin] ‘The loop will set all 18 pins  to Output low
Next
```

Set Pin to Output High

Byte Code: **$B1** *Pin*

Variables: *Pin*

Pin is a one BYTE value 0 through 17. It can be a BYTE variable or a constant.

Function

Sets any of the Coprocessors 18 pins to Output HIGH

Explanation:

The Coprocessor has 18 pins that can be set four different modes.

Input without Pullup
Input with Pullup
Output Low
Output High

Basic Stamp Examples:

```
Shiftout Dpin, Cpin, 1, [$B1, 0]      ‘Sets the Coprocessors pin 0 to Output HIGH

‘Using the 16 bit shiftout command with the optional 16 bit command.

Shiftout Dpin, Cpin, 1, [$B100\16,$B101\16]  ‘Set pins 0 and 1 to Output HIGH

‘Using shiftout with an 8 bit BYTE variable.

ledpin        VAR   BYTE

For ledpin=0 to 17
 Shiftout Dpin, Cpin, 1, [$B1, ledpin] ‘The loop will set all 18 pins HIGH
Next
```

Set Pin to INPUT no pullup

Byte Code: **$B2** *Pin*

Variables: *Pin*

Pin is a one BYTE value 0 through 17. It can be a BYTE variable or a constant.

Function

Sets any of the Coprocessors 18 pins to Output LOW

Explanation:

The Coprocessor has 18 pins that can be set four different modes.

Input without Pull-up
Input with Pull-up
Output Low
Output High

Basic Stamp Examples:

```
Shiftout Dpin, Cpin, 1, [$B2, 0]      ‘Sets the Coprocessors pin 0 to Input

‘Using the 16 bit shiftout command with the optional 16 bit command.

Shiftout Dpin, Cpin, 1, [$B200\16,$B201\16]  ‘Set pins 0 and 1 to Input

‘Using shiftout with an 8 bit BYTE variable.

ledpin        VAR   BYTE

For ledpin=0 to 17
 Shiftout Dpin, Cpin, 1, [$B2, ledpin] ‘The loop will set all 18 pins to Input
Next
```

Set Pin to INPUT with internal pullup

Byte Code: **$B6** *Pin*

Variables: *Pin*

Pin is a one BYTE value 0 through 17. It can be a BYTE variable or a constant.

Function

Sets any of the Coprocessors 18 pins to Input with internal pullup

Explanation:

The Coprocessor has 18 pins that can be set four different modes.

Input without Pull-up
Input with Pull-up
Output Low
Output High

The internal pull-ups allow circuits for example a switch to pull the input to ground without using an external pull-up resistor. The voltage on the input pin will read HIGH and have a small output current in this state.

Basic Stamp Examples:

```
Shiftout Dpin, Cpin, 1, [$B6, 0]      ‘Sets the Coprocessors pin 0 to Input with Pull-up

‘Using the 16 bit shiftout command with the optional 16 bit command.

Shiftout Dpin, Cpin, 1, [$B600\16,$B601\16]  ‘Set pins 0 and 1 to Input with pull-up.

‘Using shiftout with an 8 bit BYTE variable.

ledpin        VAR   BYTE

For ledpin=0 to 17
 Shiftout Dpin, Cpin, 1, [$B2, ledpin] ‘The loop will set all 18 pins to Input with pull-ups
Next
```

PUSH BYTE TO SRAM

Byte Code: **$D0** *Pin*

Variables: *Pin*

Pin is a one BYTE value $00 through $FF. It can be a BYTE variable or a constant.

Function

Writes one byte to the 255 byte circular buffer

Explanation:

Assembly language programmers have a limited number of registers (Varialbe space) to work with. When additional space is needed during programming the programmer can store values on what is called a stack. It is similar to a stack of dinner plates. Stack up 10 plates then take the plates back off the pile in reverse order, the 10th plate would be the first to come off then the 9th and so on. The PUSH command $D0 works the same way. Up to 255 bytes can be pushed on the stack and then taken back off the stack with the POP command $D1.

This stack is a little different in that it works like a circle. If 256 bytes are pushed on the stack without popping the stack the 256th byte will overwrite the first location 0 additional pushes without pops will continue to overwrite previous data.

One simple use of this circle buffer is to store the last 255 bytes of of a sensor for example a temperature reading could be stored every 6 minutes. The data buffer would then contain the last 24 hours of temperature readings.

Typical usage would be to store basic stamp variables and then use the variables for something else and then restore the original values using the POP command.

Basic Stamp Example:

Shiftout Dpin, Cpin, 1, [$D0, Myvar] ‘Pushes the value of Myvar in the stack.

Also see: POP BYTE FROM SRAM $D1

POP BYTE FROM SRAM

Byte Code: **$D1**

Variables: *None*

Function

Sets up coprocessor output to pop one byte off the stack.

Input via shiftin *Byte*

The SHIFTIN command is used immediately after to read the data.

Explanation:

The POP command $D1 is used to read data from the SRAM stack in the coprocessor. Two commands are used, $D1 is shifted out followed by the shiftin command to receive the data.

Basic Stamp Example:

```
Myvar VAR Byte
Shiftout Dpin, Cpin, 1, [$D1]   `Setup for output of 1 byte from stack
Shiftin Dpin, Cpin, 0, [Myvar]    `One byte is transferred from the coprocessor to the variable Myvar
```

Set the stack pointer to 0

Byte Code: **$D2**

Variables: *None*

Function

Sets the memory pointer to the 255 byte circular stack to zero.

Explanation:

The 255 byte stack can be written to EPROM or FLASH memory for long term storage using the WRITEFLASH $D2 command. When the PUSH command is used to store data that is to be stored in Eprom or Flash memory later retrieval of the data requires you know what byte the data starts from. This command sets the pointer to zero so the first PUSH command writes the data in Sram to location 0.

Basic Stamp Example:

```
Shiftout Dpin, Cpin, 1, [$D2]   `Reset stack pointer to 0
```