MiniProg Users Guide and Example Projects

Cypress MicroSystems, Inc.
2700 162nd Street SW, Building D
Lynnwood, WA 98037
Phone: 800.669.0557
Fax: 425.787.4641

TABLE OF CONTENTS

INTRODUCTION TO MINIPROG

The Cypress MicroSystems MiniProg gives you the ability to program PSoC parts quickly and easily.

It is small and compact, and connects to your PC using the provided USB 2.0 cable.

During prototyping, the MiniProg can be used as an in-system serial programmer (ISSP) to program PSoC devices on your PCB (see application notes AN2014 and AN2026 available online at www.cypress.com for more details).

For production purposes, we recommend using the CY3207ISSP programmer or a third-party production programmer.

Once the MiniProg is connected, you can use PSoC Programmer software to program. (This free software can either be launched from within PSoC Designer or run as a standalone program.)

WHAT COMES WITH MY MINIPROG?

Please confirm that your kit includes the following items:

- MiniEval Evaluation Board
- MiniProg Programmer
- CY8C29466-24PXI 28-Pin DIP Sample
- PSoC Designer CD
- USB Cable
- User Guide

SPECIFICATIONS FOR MINIPROG

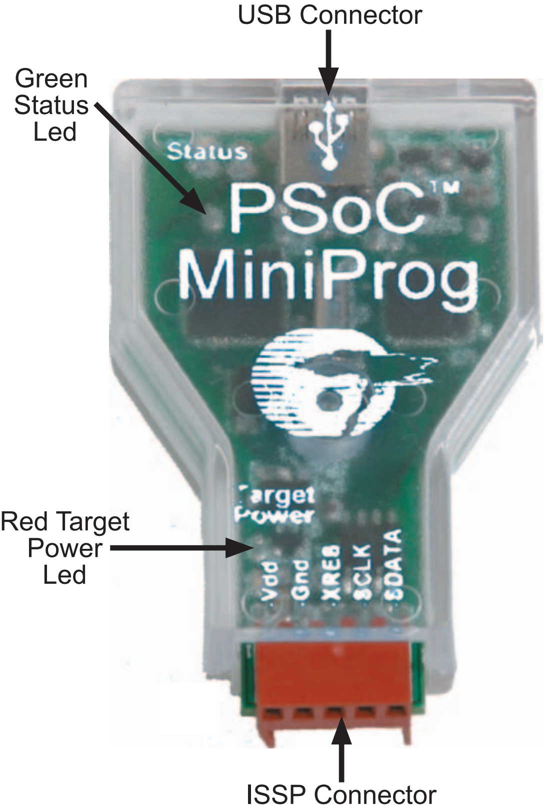
The operating temperature of the MiniProg is from 0° C to 50° C.

Always plug the USB cable into the MiniProg before attaching it to the five-pin header on the board.

When using an ISSP adapter cable with MiniProg, keep the length under six inches to avoid signal integrity issues.

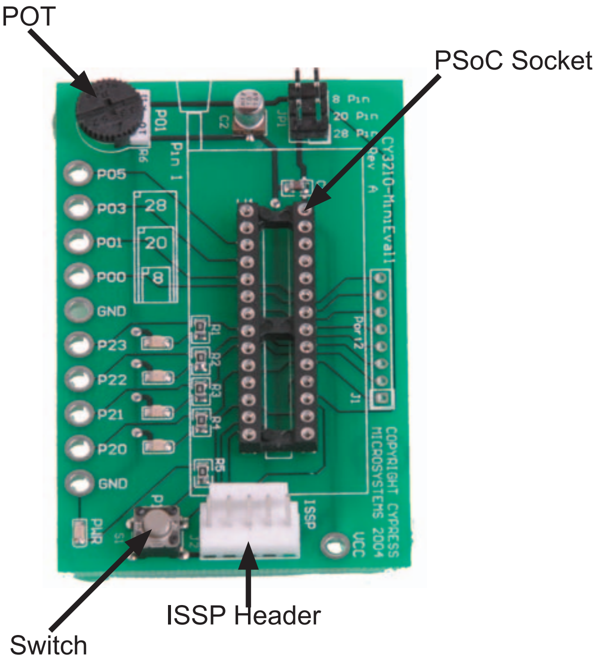
When using MiniProg, the LEDs blink at a variable rate to track connection status. The green LED near the USB connector turns on after MiniProg is plugged into the computer and configured by the OS. If MiniProg cannot find the correct driver in the system, this LED will not turn on. After the device has been configured, the LED stays on at about a 4-Hz blink rate. This changes during programming, where the blink duty cycle increases.

The red LED at the bottom turns on when the MiniProg powers the part. The LED is off when power is provided by the target board.


USB Connector
Green Status Led
Status
PSoC™
MiniProg
Target Power
Red Target Power Led
Vdd
Gnd
XRES
SCLK
SDATA
ISSP Connector

MINIEVAL DESCRIPTION

Shown below is the MiniEval1 board, which can be used with the MiniProg programmer to evaluate a PSoC device using some simple example projects. No wire connections are needed from the PSoC Socket.



Four Example Projects are described in the following sections. Each section is organized as follows:

Project Name: PSoC Designer project name.

Purpose: Overview of the project.

Implementation: A more detailed overview.

Example Code (main.asm): Code to run the project.

The example projects are available in PSoC Designer. To use, open PSoC Designer and browse to select the correct file. The example projects are found in …`\Program Files\Cypress MicroSystems\ PSoC Designer\Examples`. Choose the chip type you desire and open the project's .soc file.

When using the MiniEval programmer, do not use the "Connect" and "Download" buttons in PSoC Designer. These are for use with an In-Circuit Emulator (ICE).
Instead, click on the "Program" button to use PSoC Programmer with your PSoC.

EXAMPLE PROJECT #1 BLINK AN LED

Project Name: **ASM_Example_Blink_LED**

Purpose: To demonstrate blinking an LED at a varying duty cycle using a hardware PWM.

Implementation: The clock dividers VC1, VC2, and VC3 are used to divide the 24 MHz system clock by 16, 16 and 256, respectively. The resulting 366 Hz clock is used as the input to an 8-bit PWM. This in turn produces an LED blink period of 1.4 Hz.

Example Code (main.asm):

```
// include m8c specific declarations
include “m8c.inc”
// include User Module API specific
declarations
include “psocapi.inc”
export  _main:

_main:
    // Enable PWM
    lcall   PWM8_1_Start
    lcall   PWM8_1_EnableInt

    // Enable Global interrupts
    M8C_EnableGInt
loop:
    jmp     loop
```

Project Name: **ASM_Example_DAC_ADC**

Purpose: To demonstrate a PSoC project that outputs a SINE wave using a 6-bit DAC. The SINE wave period is based on the current ADC value of the potentiometer.

Implementation: This project uses a 64-entry SINE look-up table to generate values used to update a 6-bit DAC. An 8-bit counter is utilized to generate an interrupt at the DAC update rate (1/64 SINE wave period). By adjusting the counter period, the DAC frequency and the resulting SINE frequency may be modified. The counter period is reloaded with the current ADC conversion value. The ADC input voltage may be between 0 and Vdd volts depending on the potentiometer. At higher frequencies, SINE wave jitter may be observed due to the large timing impact of a one-count change in the ADC conversion.

Example Code (main.asm):

```
// include m8c specific declarations
include “m8c.inc”
// include User Module API specific
declarations
include “psocapi.inc”

export  _main
export  bADCvalue
```

```
export  bTablePos
export  SINtable

// inform assembler that variables follow
area    bss(RAM)

// Store ADC value for debug watch variable
bADCvalue:   blk 1

// Stores last table position index
bTablePos:   blk 1

// inform assembler that program code follows
area    text(ROM,REL)
_main:
    // starts DAC value update counter
    lcall Counter8_1_Start
    lcall Counter8_1_EnableInt
// Turn on PGA power
    mov   A, PGA_1_MEDPOWER
    lcall PGA_1_Start

    // Turn on DAC power
    mov   A, DAC6_1_HIGHPOWER
    lcall DAC6_1_Start

    // Turn on ADC power
    mov   A, DELSIG8_1_HIGHPOWER
```

```
    lcall DELSIG8_1_Start
    lcall DELSIG8_1_StartAD

    // Enable Global interrupts
    M8C_EnableGInt

loop:
    // if ADC conversion complete then.....
    lcall DELSIG8_1_fIsDataAvailable
    jz    loop
    // get ADC result and convert to offset
binary
    lcall DELSIG8_1_cGetDataClearFlag
    add   A, 0x80
    // store value for debug watch variable
    mov   [bADCvalue], A

    // counter period less then 0x03 is
invalid
    cmp   A, 0x03
    // excessive interrupt servicing
    jnc     LoadCounter
    mov   A, 0x03
```

```
LoadCounter:
    // update DAC update rate
    lcall Counter8_1_WritePeriod
    jmp   loop

area lit


// 64 entry SINE look-up table
SINEtable:
db  31, 33, 36, 39, 41, 44, 46, 49, 51, 53,
55, 56, 58, 59, 59
db  60, 60, 60, 59, 59, 58, 56, 55, 53, 51,
49, 47, 44, 42, 39
db  36, 33, 31, 28, 25, 22, 19, 16, 13, 11,
9, 7, 5, 3, 2, 1, 0
db  0, 0, 0, 1, 2, 3, 4, 6, 7, 10, 12, 14,
17, 20, 23, 26, 29

area text
```

EXAMPLE PROJECT #3 DYNAMICALLY RE-CONFIGURING A PWM

Project Name: **ASM_Example_Dynamic_PWM_PRS**

Purpose: To demonstrate PSoC's dynamic re-configuration capability by switching a digital block between a PWM8 and a PRS8 (Pseudo Random Sequence). This example project also demonstrates the advantages of using a PRS to generate a pulse width. A benefit of the PRS is that it does not generate the strong frequency harmonics of an equivalent PWM.

Implementation: The clock dividers VC1, VC2, and VC3 are used to divide the 24 MHz system clock by 16, 16 and 128, respectively. The resulting 732 Hz clock becomes the input to an 8-bit Counter User Module in the base configuration (this is the first configuration in PSoC Designer).

If the button on the MiniProg is released, configuration PWM_config is loaded and a period of two is loaded into the counter. If the button is pressed and held, configuration PRS_config is loaded and a period of 128 is loaded into the counter.

The PWM configuration contains a standard 8-bit PWM with a duty cycle of 50%. Both the pulse width and terminal count outputs are displayed on LEDs.

The PRS configuration contains a PRS with pulse density (analogous to pulse width) and shifted bit stream output on LEDs.

Example code (main.Asm):

```
// include m8c specific declarations
include “m8c.inc”
// include User Module API specific
declarations
include “psocapi.inc”
export  _main:

_main:
    // configure port pins
    and   reg[PRT1DR], ~0x10
    mov   reg[PRT2DR], 0x00

    // start clock generator
    lcall Counter8_1_Start

    // load PRS configuration
    lcall LoadConfig_PRS_Config
    jmp   PWM
PRS:
    // stop and unload PWM configuration
    lcall PWM8_1_Stop
    lcall UnloadConfig_PWM_Config
    // then load PRS config
    lcall LoadConfig_PRS_Config

    // update clock divider, don’t wait for
```

```
period
     reload
    lcall Counter8_1_Stop

    mov   A, 0x7F
    lcall Counter8_1_WritePeriod
    lcall Counter8_1_Start
    // configure and start PRS
    mov   A, 0x01
    lcall PRS8_1_WriteSeed
    mov   A, 0xB8
    lcall PRS8_1_WritePolynomial
```

```
    lcall PRS8_1_Start
    // load compare value, must be loaded
after PRS is
     started
    mov   reg[PRS8_1_SEED_REG], 0x7F

PRSloop:
    // wait for button release
    tst   reg[PRT1DR], 0x10
    jnz   PRSloop
    // simple debounce
    tst   reg[PRT1DR], 0x10
    jnz   PRSloop
    jmp   PWM

PWM:
    // stop and unload PRS configuration
    lcall PRS8_1_Stop
    lcall UnloadConfig_PRS_Config
    // then load PWM config
    lcall LoadConfig_PWM_Config
    // update clock divider, don't wait for
period
     reload
    lcall Counter8_1_Stop
```

```
    mov   A, 0x01
    lcall Counter8_1_WritePeriod
    lcall Counter8_1_Start

    // configure and start PWM
    mov   A, 0xFF
    lcall PWM8_1_WritePeriod
    mov   A, 0x7F
    lcall PWM8_1_WritePulseWidth
    // enable PWM
    lcall PWM8_1_Start

PWMloop:
    // wait for button release
    tst   reg[PRT1DR], 0x10
    jz    PWMloop
    // simple debounce
    tst   reg[PRT1DR], 0x10
    jz    PWMloop
    jmp   PRS
```

EXAMPLE PROJECT #4 COMBINING PWMS USING OUTPUT LOGIC

Project Name: **ASM_Example_LED_Logic**

Purpose: To demonstrate a PSoC project designed to blink an LED using the output of two PWMs. The outputs are combined using an AND gate in an output bus logic block. This logical combination results in a beat frequency of 1.4 Hz.

Implementation: The clock dividers VC1 and VC2 are used to divide the 24 MHz system clock by 16 and 16, respectively. The resulting 93.37 kHz clock becomes the input to the two 8-bit PWM User Modules with respective periods of 256 and 255. This produces the LED beat frequency of 1.4 Hz.

Example code (main.Asm):

```
// include m8c specific declarations
include "m8c.inc"

// include User Module API specific
declarations
include "psocapi.inc"

export  _main:

_main:
    // Enable PWM1
    lcall PWM8_1_Start
    // Enable PWM2
    lcall PWM8_2_Start

loop:
    jmp   loop
```

CYPRESS CUSTOMER SUPPORT

We are committed to meeting your every need.

For more information about PSoC, check us out on the web at www.cypress.com/psoc. There you will find data sheets, hundreds of application notes, contact information for local PSoC certified consultants, and recorded tele-training modules for newcomers to the PSoC world.

We offer live tele-training sessions regularly. Check online at www.cypress.com/support/training.ctm for the next scheduled time.

For application support please contact us online or call between 8 am – 6 pm PST at 1.800.669.0557 ext. 4814. We offer a four-hour response time at our call center during normal business hours.

http://www.cypress.com/ http://www.cypress.com/support/mysupport.cfm