# AN2104

# PSoC® 1 - Dynamic Reconfiguration with PSoC Designer™

**Author: Ben Kropf**

**Associated Project: Yes**

**Associated Part Family: PSoC 1**

**Software Version: PSoC® Designer™ 5.4 SP1**

**Related Application Notes: AN49079**

AN2104 gives a detailed explanation of dynamic reconfiguration, and presents best practices for successfully implementing dynamic reconfiguration with PSoC Designer. Dynamic reconfiguration is a unique feature of PSoC Designer that allows a designer to easily create a project that uses more resources of the chip than are statically available. This is done by time multiplexing the resources inside of a PSoC Chip.

## Contents

1 Introduction....1
2 Creating Configurations in PSoC Designer....2
2.1 Guidelines for Configuration Organization....3
2.2 Making Changes to Configurations....7
2.3 Firmware Coding With Dynamic Reconfiguration....7
2.4 Unloading and Loading Configurations....9
2.5 Interrupts and Dynamic Reconfiguration....9
2.6 Reconfiguration Execution Time....9
2.7 Do-It-Yourself Dynamic Reconfiguration....9
3 Example Project....10
4 Summary....10
A Device Editor Configuration Changes....11
B File Changes and Dynamic Reconfiguration....12
C Interrupts and Dynamic Reconfiguration....14
D Example Project Instructions....16
Document History....20
Worldwide Sales and Design Support....21

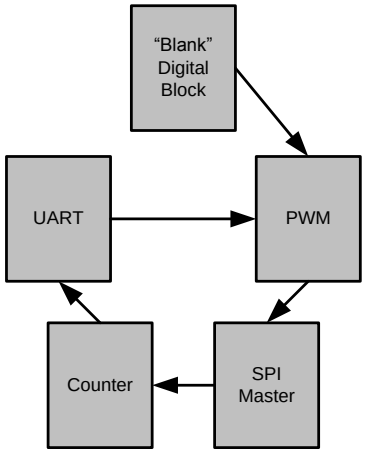
## 1 Introduction

Every programmable semiconductor device, including PSoC 1, has limited resources. However, the technique of dynamic reconfiguration developed by Cypress allows PSoC 1 devices to reuse analog and digital resources to achieve greater levels of functionality.

The power of dynamic reconfiguration can be seen in the example of an electric shaver. A typical electric shaver may need battery charging and motor driving functionality. It may be possible to use a smaller PSoC with limited resources to cut costs for this application. A more limited device would only have the resources to do either battery charging or motor driving, but not both. Dynamic reconfiguration allows the PSoC device to perform both operations. Dynamic reconfiguration does this by time multiplexing the resources inside the PSoC device. Because motor driving and battery charging do not occur at the same time, it is possible for them to share internal PSoC resources.

The resources these two operations share are the highly configurable analog and digital resources, referred to as analog or digital "blocks". Figure 1 shows how one digital block is reconfigured to perform a variety of digital functions. Analog blocks are reconfigurable in the same way. Only one function may be active at any point in time. It is by reusing the analog and digital blocks that dynamic reconfiguration is accomplished. For information on implementing dynamic reconfiguration with CapSense®, see the application note AN49079 – CapSense® Plus Dynamically Reconfiguring CapSense.

Figure 1. PSoC Block Reconfiguration

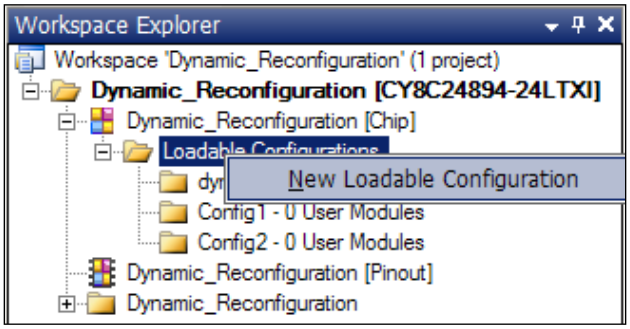



The PSoC Designer software development tool allows you to easily implement dynamic reconfiguration in firmware. Multiple hardware configurations for the device are defined with an intuitive GUI. You can switch between and interact with these hardware configurations at a firmware API level.

## 2 Creating Configurations in PSoC Designer

This section describes how to set up a project that uses Dynamic Reconfiguration in PSoC Designer. To create a new hardware configuration, navigate to **Interconnect** > **Add Loadable Configuration to "Your Project Name"** or in the **Workspace Explorer**, right-click the **Loadable Configurations** folder and select **New Loadable Configuration**. See Figure 2.

Figure 2. Adding New Configuration




After a loadable configuration is added, a new folder appears under **Loadable Configurations**. As Figure 2 shows, two new configurations have been added, Config1 and Config2. To see the Analog and Digital Block usage for each configuration, double-click on the configuration folder.

A configuration is renamed by right-clicking on the configurations folder. Configurations are deleted by right-clicking the configurations folder or by navigating to Interconnect > Delete 'Config Name' Loadable Config…

Changes to individual configurations are now made by double-clicking the desired configuration folder. The interconnect view of the Device Editor changes to display the hardware setup for the selected configuration. For more information regarding the use of the PSoC Designer software, see the *IDE User Guide* for PSoC Designer.

The original configuration in a project is treated differently from any added configurations. Therefore, the original configuration is now referred to as the "base" configuration in this document. Any secondary added configurations are referred to as "overlays."

## 2.1 Guidelines for Configuration Organization

This section describes how to determine the configuration organization for a project. Do the following in the order they are listed:

1. Determine the required functionality for the project.
2. Determine the user modules required for the required functionality.
3. Determine the analog and digital block resources that each user module uses.

Table 1. Functionality, User Module, and Block Requirements

| Functionality | User Module | Blocks Used |
|---|---|---|
| CapSense buttons | CSD | 3 digital, 1 analog |
| Thermistor measurement | ADCINC and PGA | 1 digital, 2 analog |
| $I^2C$ slave communication | EZI2Cs | 0 |
| Factory test or calibration (Half-Duplex UART) | RX8 and TX8 | 1 digital for each |
| ~1 ms system timer | Timer8 | 1 digital |

Table 1 is an example list of user modules that are needed for an application. Because of the need for the CSD user module (UM), assume that a CY8C24894-24LTXI device is used. This device has a CSD UM, four digital blocks, and six analog blocks. From Table 1, it is seen that seven digital blocks and three analog blocks are needed. Dynamic reconfiguration enables functional requirements such as these with seemingly fewer resources than required.

4. Determine which user modules must be active during various states of system operation.

   As seen in Table 2, the half-duplex UART only needs to be active when the device is in the factory. The system timer and the $I^2C$ slave must be active at all times. The CapSense functionality must be active for 50 percent of the time during normal operation. The thermistor measurement must be active for 5 percent of normal operation.

Table 2. Functionality Time Requirements

| Functionality | User Module | Active Period |
|---|---|---|
| CapSense buttons | CSD | Scan capacitive elements for 10 ms every 20 ms |
| Thermistor measurement | ADCINC and PGA | Measure the thermistor for 1 ms every 20 ms |
| $I^2C$ slave communication | EZI2Cs | Always active (must respond to master at any time) |
| Factory test or calibration | RX8 and TX8 | Only active during factory calibration |
| 1-ms system timer | Timer8 | Always active |

5. Place any user modules that must be active at all times in the base configuration and never unload the base configuration. However, there are limitations for the placement in the base configuration. For more details, see Appendix E.

   Figure 3 shows a diagram of the user module placement in the base configuration. (Figure 3 through Figure 8 show stylized drawings of the Device Editor view in PSoC Designer).

6. Place any user modules that have no block conflicts in the base configuration.

   In Figure 3, the programmable gain amplifier (PGA) UM is placed in the base configuration. This is because there are no other user modules that also need to use that analog block at some time during system operation. The PGA is now always loaded so that code need not be executed to unload or reload it. The Timer8 UM is placed in the extreme-right digital block. This is because the CSD UM only allows its digital blocks to be placed in the extreme-left digital blocks. Therefore, the Timer8 block must not interfere with the CSD blocks.

   It is also best at this point to configure all global resources, interconnections, and pin settings. This is because all overlays added after this will have the same settings. If overlays need to have different settings from the base, they can be changed after the overlay is added.

Figure 3. Base Configuration

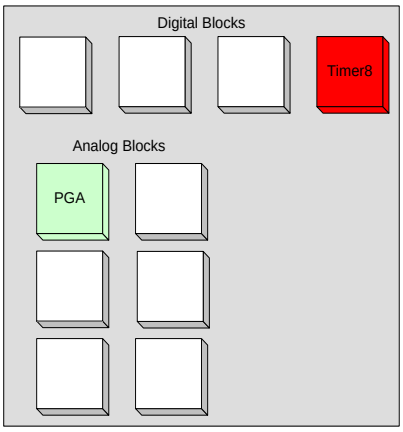



Figure 4. First Overlay Configuration

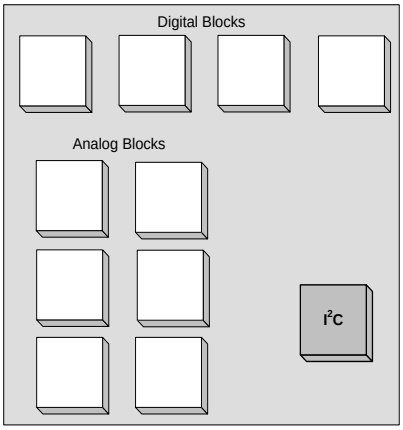



Figure 5. Second Overlay Configuration

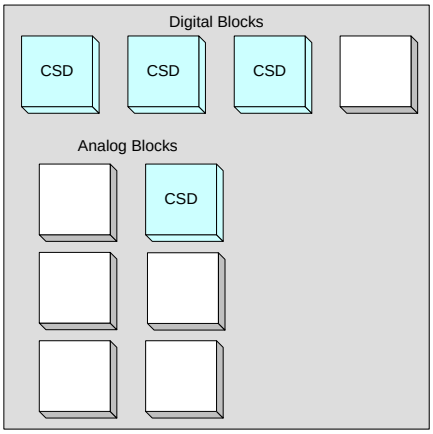


Figure 6. Third Overlay Configuration

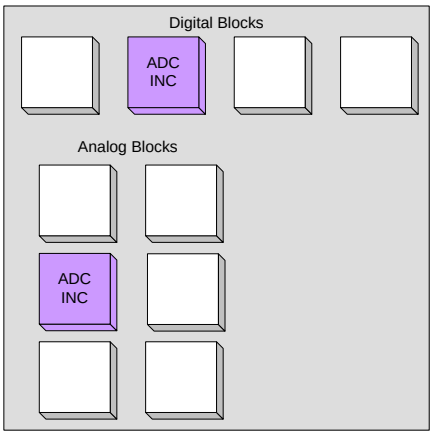



Figure 7. Fourth Overlay Configuration

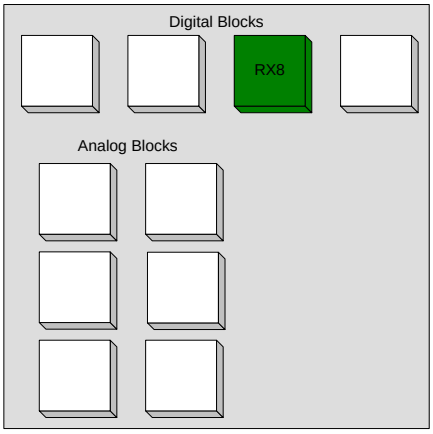



Figure 8. Fifth Overlay Configuration

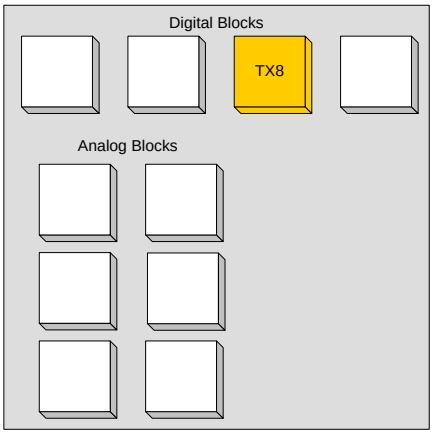


Table 3. Configuration Purposes

| Config | Purpose |
|---|---|
| Base | Always loaded |
| Overlay 1 | Always loaded for $I^2C$ |
| Overlay 2 | Loaded during normal operation when capacitive element scanning is needed |
| Overlay 3 | Loaded during normal operation when thermistor measurements are needed |
| Overlay 4 | Loaded during factory mode when waiting for data |
| Overlay 5 | Loaded during factory mode when sending data |

7. Add overlay configuration for $I^2C$ and ADC user module

   For the $I^2C$ and ADC user module, create an overlay configuration. The $I^2C$ and the ADC user module that uses decimator should not be placed in the base configuration because the $I^2C$ and decimator are considered global resources. The PSoC Designer resets the $I^2C$ and decimator control registers when any other overlay configuration is loaded over the base configuration. Thus, the overlay configuration with the $I^2C$ and the ADC user module should always be loaded along with the base configuration.

   Note that the $I^2C$ pin configuration (Name, Select, Drive, Interrupt, and Initial Value) should be copied to all other configurations including the base configuration.

   Figure 4 shows an example of the first overlay configuration for the $I^2C$ user module. There are other user modules that also should not be placed in the base configuration. See Appendix E.

8. Add additional overlay configurations and place the remaining user modules in them.

   Ensure that any user module placed in these overlay configurations do not conflict with the used blocks in the base configuration or overlays that must be active at the same time. This is because the base configuration should always remain loaded and the overlays should be loaded or unloaded on top of the base configuration. Any overlays that share a block must not be loaded at the same time. The blocks used in Figure 5 through Figure 8 do not conflict with the blocks used in the base configuration of Figure 3 or the first overlay configuration of Figure 4. This means that any single overlay may be loaded on top of the base. Also, any overlays that do not conflict with each other may be loaded at the same time. PSoC Designer does not automatically detect or prevent the occurrence of loading two configurations that conflict with each other at the same time. You must ensure that conflicting configurations are never loaded at the same time in the application code.

   Table 3 lists the configurations and the purpose for which they exist. Each configuration is loaded at a particular time to serve a particular purpose. When the functionality of the configuration is complete, it is unloaded and the next configuration is loaded.

9. Optimize miscellaneous settings

   When all the configurations are added and the user modules are placed, it is best to make further optimizations. Use the exact same global resource, pin, and signal interconnection settings for each configuration as much as possible. This way, these settings need not be reconfigured when overlays are loaded or unloaded. For instance, if one pin is configured as an output for a UART UM in an overlay, it is best to configure this same pin as an output in all the configurations. The pin then need not be reconfigured when switching between overlays. Doing these optimizations decreases the project code size and the execution time of the dynamic reconfiguration functions. It also ensures that there are no unexpected pin state changes when changing configurations.

   The project accompanying this application note is optimized in this way. One example of this is that the P1[1] GPIO is connected to the GOO[1] signal bus in each configuration. The *UART_TX* configuration is the only one that actually uses this signal path. However, by changing each of the configurations to have this interconnection setting, less code needs to be executed to connect or disconnect this signal path when switching between configurations.

   You must carefully consider the pin states when switching between configurations. For instance, the P1[1] output pin mentioned earlier is HIGH whenever the *UART_TX* configuration is not loaded. This is because it is still configured as an output, but is not connected to the output of the TX8 UM. Global output pins that are not driven by hardware are pulled HIGH. If the state of the pin needs to be LOW, the digital row LUT can be set to FALSE to manually set the output to LOW.

The example presented in this section shows how only four digital blocks are used to implement the functionality that requires seven digital blocks. Initially, it appears that it is not possible to implement the functional requirements in this device. However, dynamic reconfiguration enables all required functionality. The project accompanying this application note (AN2104) contains the configurations and user modules discussed previously in this section.

**Note:** This example project is provided only for understanding how multiple configurations are created. Not for functional testing.

## 2.2 Making Changes to Configurations

Understanding how changes made from the Device Editor to one configuration affect other configurations is important. Generally, changes made to the base configuration cause the same change in all overlays. Changes made to overlays do not affect any other configuration. There are exceptions to these general rules. See Table 7 in Device Editor Configuration Changes for full information on configuration interactions in the Device Editor.

It is difficult to keep track of all the differences in settings across configurations or avoid making unintended changes in an overlay caused by making changes to the base configuration. For these reasons, it is recommended to only make changes to the base configuration that should always be active for all configurations.

## 2.3 Firmware Coding with Dynamic Reconfiguration

The Device Editor of PSoC Designer is a tool for adding and setting up multiple PSoC hardware configurations. The Application Editor of PSoC Designer is where you write the code for the application. Some of this code must interact with the configurations. When a second hardware configuration is added to a project, there is new code that gets automatically generated. File Changes and Dynamic Reconfiguration contains more detailed information about changes to the project that occur when using dynamic reconfiguration.

Table 4. Dynamic Reconfiguration API Functions

| Function Name | Description |
|---|---|
| `LoadConfig_[config_name]` | Configures the device to implement the named configuration. |
| `ReloadConfig_[base_name]*` | Exists only for the base configuration; reloads all hardware register settings to make the base configuration active. |
| `UnloadConfig_[config_name]` | Configures the device to undo the settings of a loaded configuration. |
| `UnloadConfig_Total*` | Configures the device to unload the settings of all configurations. |
| `Is[config_name]Loaded` | Returns a non-zero value if the named configuration is loaded; returns zero otherwise. |

**Note:** * The use of these functions is not recommended because the base configuration must never be unloaded. They remain in the API for backward compatibility.

Table 4 lists and describes automatically generated functions that allow manipulation of the configurations from an abstract level. The functions that are primarily used are the `LoadConfig` and `UnloadConfig` functions. These are used to load and unload overlay configurations.

Figure 9. Dynamic Reconfiguration Software Flow

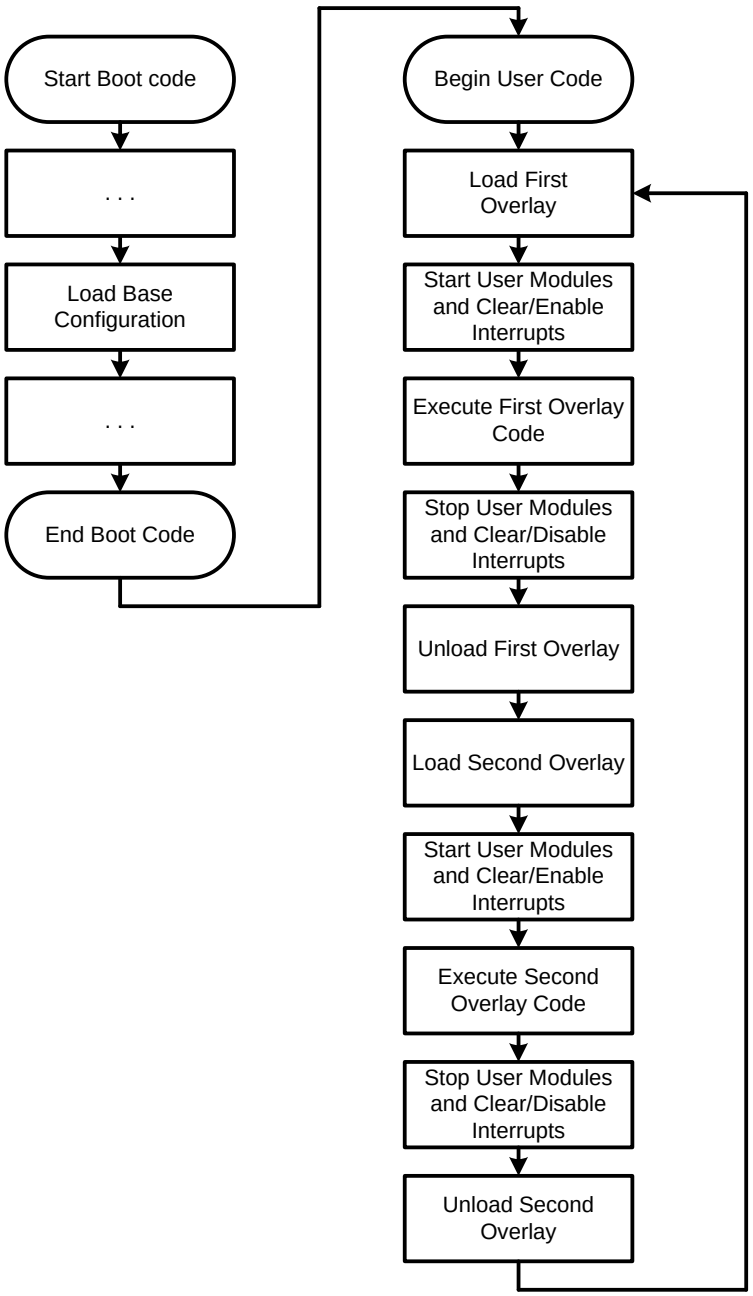



Figure 9 is a flowchart of recommended program software flow when using dynamic reconfiguration. The section on the left of the figure is code that executes in the *boot.asm* file. This is where the base configuration gets loaded. Therefore, the base is the only active configuration when code is entered in the *main.c* or *main.asm* file. The section on the right is a typical main loop.

Although the example in Figure 9 only shows two interchanging overlays, it is possible that there may be more. It is also possible for more than one overlay to be loaded at the same time. However, you must ensure that multiple overlays that are active at the same time do not use blocks that conflict with each other or the base configuration. PSoC Designer does not automatically detect this or prevent it from happening.

## 2.4 Unloading and Loading Configurations

It is a good practice to disable and clear all interrupts related to user modules in a configuration that is about to be unloaded. It is also recommended to disable all the user modules in this configuration before unloading it. This is done by calling the user modules Stop API. These steps ensure that all hardware is disabled and that all interrupts for the hardware cannot occur.

Right after loading a new configuration, user modules in that configuration need to be started; this can be done by calling the start API of the user module. If the loaded user modules use interrupts, these interrupts should be cleared and then enabled for each user module after the configuration is loaded.

## 2.5 Interrupts and Dynamic Reconfiguration

When using dynamic reconfiguration in PSoC Designer, interrupt handling is automatically improved to execute the proper interrupt service routines (ISRs). The PSoC device's hardware blocks and their associated interrupt vectors are shared when using dynamic reconfiguration. For this reason, some mechanism must choose the right ISR to execute depending upon what a block's function is. For example, one digital block may share the functionality between a Timer and an SPIS UM. Each of these UMs has a unique interrupt service routine; however, they share the same interrupt vector. The improved ISR handling checks to see which overlay is active and then jumps to the appropriate ISR. Interrupts and Dynamic Reconfiguration has more detailed information about handling interrupts when using dynamic reconfiguration.

## 2.6 Reconfiguration Execution Time

Using the API functions to load and unload configuration takes CPU execution time. This time is dependent both on the number of blocks being reconfigured and the frequency of the CPU clock. Equation 1 shows that the execution time ($t$) is directly proportional to the number of reconfigured registers ($N_R$) and inversely proportional to the frequency of the CPU clock ($F_{CPU}$).

$$t \propto \frac{N_R}{F_{CPU}} \qquad \text{Equation 1}$$

Table 5 lists typical load and unload times for a somewhat major reconfiguration of six blocks and other miscellaneous hardware settings.

Table 6 lists typical load and unload times for a less major reconfiguration of only two blocks and some miscellaneous hardware settings. The times in these tables give a good idea of how much time dynamic reconfiguration takes. Unload times are generally shorter because fewer registers need to be written to unload a configuration.

Table 5. Reconfiguration Times for Six Blocks

| CPU Clock | Load Time (ms) | Unload Time (ms) |
|---|---|---|
| 24 MHz | 0.216 | 0.172 |
| 3 MHz | 1.72 | 1.44 |
| 0.75 MHz | 7.00 | 5.36 |

Table 6. Reconfiguration Times for Two Blocks

| CPU Clock | Load Time (ms) | Unload Time (ms) |
|---|---|---|
| 24 MHz | 0.096 | 0.072 |
| 3 MHz | 0.760 | 0.600 |
| 0.75 MHz | 3.00 | 2.32 |

## 2.7 Do-It-Yourself Dynamic Reconfiguration

Do-It-Yourself Dynamic Reconfiguration is not recommended for most designs, but may be appropriate when code space or execution times are important. Ease of use, code maintainability, and readability are reduced when doing Do-It-Yourself Dynamic Reconfiguration. Also knowledge of PSoC hardware configuration registers is required.

Sometimes, it is more effective to write custom dynamic reconfiguration code. In this case, overlay configurations should not be added in the Device Editor. The code generated by PSoC Designer may sometimes not be fast enough or may generate too much code. Therefore, it may be preferable to write custom functions that modify the registers correctly to dynamically reconfigure the hardware.

For example, if one digital block must be shared between an ADC user module and a TX8 serial transmitter user module, it may be better to write two small functions that reconfigure only the digital block between the functionality needed for the ADC and the TX8 functionality. The analog blocks of the ADC need not be unloaded or loaded when this reconfiguration is done. If two custom functions are written, they are very small in code size (about 30 bytes each) and execute much faster than the times listed in Table 5 and Table 6. This is a much more efficient alternative to adding overlay configurations to a project.

If custom reconfiguration functions are written, ensure that any shared interrupt vectors are handled correctly. This is done by having a separate ISR for each configuration. When an interrupt occurs for the block, a flag should be checked to see what the block's function is. After this check is made, the appropriate ISR is executed.

The best method for accomplishing Do-It-Yourself Dynamic Reconfiguration is to create separate projects with the user modules that are going to be reconfigured placed in the desired blocks. After that, it is best to run a diff on the files *PSoCConfigTBL.asm*, and *PsoCConfig.asm*. These files contain all of the device configuration information. Based on those differences, simple routines can be created to change the register settings for Do-It-YourSelf Dynamic Reconfiguration. When you do this, you do not have the convenience of the Cypress-supplied API for the UM that you are reconfiguring to. It is best to review the API found in the Cypress UM, and copy over whatever functions are relevant to your design.

# 3 Example Project

Included with this application note is an example project (AN2104_Working_Example). This project has two overlays and a base configuration. One of the overlays has an RX8 UM placed in it. When this overlay is loaded, the RX8 UM listens for the ADC command byte.

When the ADC command byte is received, the RX overlay is unloaded, and the ADC Overlay is loaded. In this overlay, the ADC reads a potentiometer, and sends the results out via a TX8 UM that is placed in the same block as the RX8 UM. After the result is sent out, the ADC overlay is unloaded and the RX overlay is loaded again; and the process repeats.

This example is created in such a way that more configurations can be added, and more code can be written for those configurations.

Instructions on how to use the project are found in Example Project Instructions.

# 4 Summary

This application note describes how to use dynamic reconfiguration in PSoC devices. It also describes best practices when using dynamic reconfiguration. When developing firmware projects for PSoC devices, it is best to examine what and when digital and analog resources are used. Often, dynamic reconfiguration can be used to fit more digital and analog functionality into any PSoC device.

# A Device Editor Configuration Changes

Table 7. Setting Interactions when Changing Base Configuration Settings

| Changed Setting | Affects |
|---|---|
| Pin Name | Change affects all the overlays in which the pin has the same name as in the base. |
| Pin Drive | Change affects all the overlays in which the pin has the same drive as in the base. |
| Pin Interrupt | Change affects all the overlays in which the pin has the same interrupt setting as in the base. |
| Pin Select | Change affects all pin settings in all configurations. |
| Global Resource Parameter | Change affects all the configurations whether overlay has the same or a different setting. |
| User Module Addition | Does not affect any other configurations. |
| User Module Deletion | Does not affect any other configurations. |
| User Module Settings | Does not affect any other configurations. |
| Digital Row Input Settings | Does not affect any other configurations. |
| Digital Row Output Settings | Does not affect any other configurations. |
| Analog Input Multiplexers | Change affects all the overlays in which the multiplexer has the same setting as in the base. |
| Analog Clock Multiplexers | Change affects all the overlays in which the multiplexer has the same setting as in the base. |
| Analog Comparator Bus Settings | Change affects all overlays in which the comparator bus has the same setting as in the base. |

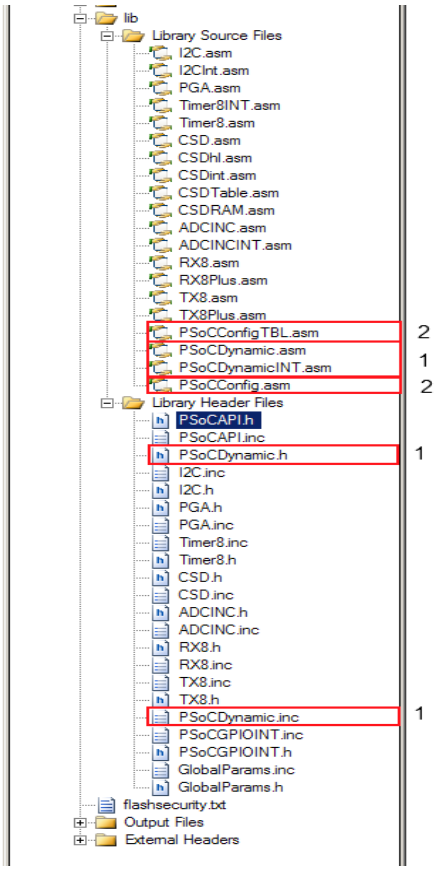
**Note:** Changes to settings in overlay configurations do not affect any other project configurations.

# B File Changes and Dynamic Reconfiguration

This appendix describes the new files that are generated in a project when using dynamic reconfiguration. It also describes changes that take place in files that already exist.

There are files that are generated for a project when dynamic reconfiguration is activated by adding an overlay to a project. Figure 10 shows the PSoC Designer view of the files that are generated. Sections marked '1' show two assembly files: one header file and one include file generated for this project to accommodate the use of dynamic reconfiguration. The two files highlighted in the section marked '2' change significantly when using dynamic reconfiguration. These highlighted files are discussed.

Figure 10. Project Directory With New Files




The *psocdynamic.asm* file adds one function for each configuration in the project. Each function is named `Is[ConfigName]Loaded`. An array of bytes in RAM is declared when dynamic reconfiguration is used in a project. The size of the array is the number of configurations in a project divided by eight, rounded up. There is one byte for every eight configurations. This array of bytes is named `ACTIVE_CONFIG_STATUS`. It is declared near the bottom of the *psocconfig.asm* file. Each bit of each byte corresponds to one of the configurations. If the corresponding bit is HIGH, the configuration is loaded. If it is LOW, the configuration is not loaded. You must not modify these. The `Is[ConfigName]Loaded` function is called to check if a configuration is active. This gives the firmware the capability to know which configurations are loaded at any given time. There is no limit to the number of overlay configurations that may be added to a project. The only practical limit is the amount of flash memory and RAM used by the functionality of each configuration.

The *psocdynamicint.asm* file determines how interrupts are handled when using dynamic reconfiguration. This is explained in Interrupts and Dynamic Reconfiguration.

The *psocdynamic.h* file exports all dynamic reconfiguration functions for use in C code. Each configuration gets three functions that are exported in the file as follows:

Code 1. Exported C Function Declarations

```
extern void LoadConfig_an2104( void);
extern void UnloadConfig_an2104( void);
extern char Isan2104Loaded( void);
```

In each of these declarations, "an2104" is the name of the configuration. These functions were discussed previously in this application note. There is also one function added that affects all the configurations. It is called UnloadConfig_Total and is used to unload all the user module settings of all the configurations. It also clears all ACTIVE_CONFIG_STATUS bytes, indicating that no configurations are loaded. It does not affect pin and interconnect settings, but only user module settings. It is not recommended to use this function because it is not recommended to unload the base configuration. The function still exists for the purpose of backward compatibility.

The base configuration has one additional function that overlays do not have. This function is named ReloadConfig_[BaseName] and it only loads user-module register settings for the base configuration. It is not recommended to use this function because the base configuration should never be reloaded. The function still exists for the purpose of backward compatibility.

The *psocdynamic.inc* file defines values that are used by the dynamic reconfiguration functions (LoadConfig and UnloadConfig) can be found in *PsoC™Config.asm*. In this file, all of the nondigital and analog block settings are configured, including pin settings, global select settings, row LUT settings, and Decimator settings. Register settings for the digital and analog blocks can be found in *PSoCConfigTBL.asm.*

# C Interrupts and Dynamic Reconfiguration

Interrupts must be handled differently when using dynamic reconfiguration. It is possible that digital and analog blocks are shared between different user modules. Therefore, it is necessary to correctly choose the ISR to execute based on the user module loaded at the time. There is one interrupt vector for each digital block and one for each analog column. Therefore, the interrupt vector must be routed to the ISR for the user module that is loaded when the interrupt occurs. It is best to place user modules in such a way that the number of shared interrupt vectors is minimized.

Code 2 shows this implementation. It is taken from the interrupt vector table located in *boot.asm*.

Code 2. New Interrupt Vector Label

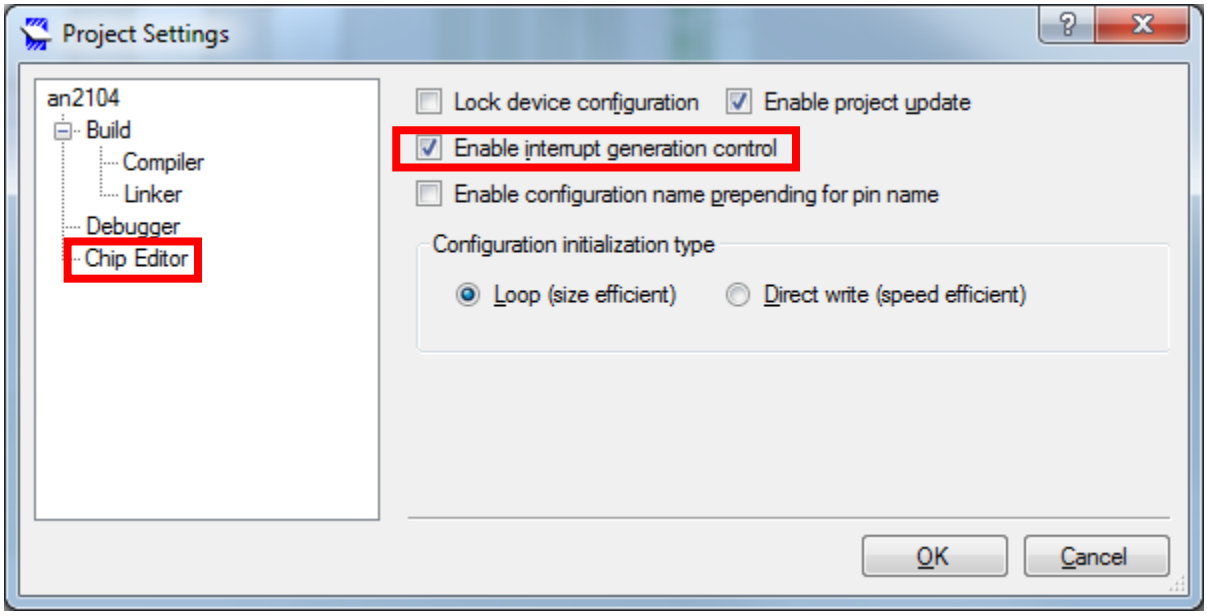
```
org   2Ch;PSoC Block DCB03 Interrupt Vector
ljmp   Dispatch_INTERRUPT_11
reti
```

Normally, this interrupt vector would have jumped to a routine with a name specific to one user module. The vector now jumps to a routine called `Dispatch_INTERRUPT_11` instead of one user module's ISR. This interrupt handler is found in the *psocdynamicint.asm* file that is added to the project. This interrupt handler determines the configuration loaded that actually uses that block. This function consecutively checks each configuration that shares the block to see which is active. When it finds the active configuration, it jumps to the appropriate ISR for the loaded user module. If many configurations share the same block, this function takes longer to execute. This also causes different interrupts to have different latencies.

This extra interrupt handling is undesirable because it adds more execution time to each interrupt. The execution time for this extra handler increases with each user module that uses the block, because more configurations must be checked. It is impossible to service different interrupts from the same resource without having them run a little slower, due to the extra handling that must be done. However, there is some flexibility in PSoC Designer that allows optimization of this added interrupt handling.

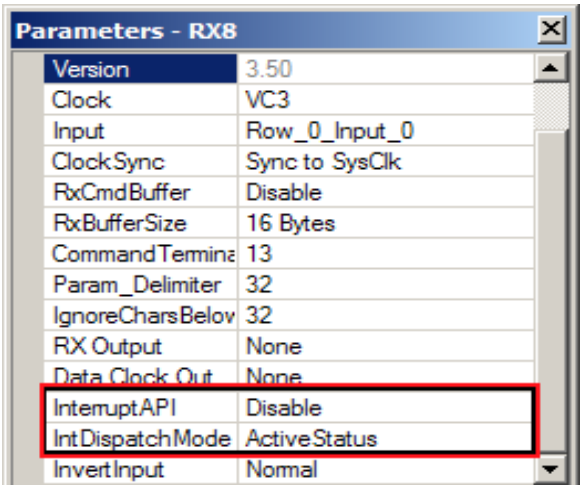
If Interrupt Generation Control is enabled, you get flexible options to individually control the ISR code. The option of controlling interrupt ISR generation is enabled in the **Device Editor** tab of the **Project** > **Settings** > **Chip Editor** Menu. The box marked **Enable interrupt generation control** may be checked as seen in Figure 11. By default, this box is unchecked.

Figure 11. Enabling Interrupt Generation Control Setting



When this option is enabled, user modules in all the configurations have two new user module parameters added. This is seen in Figure 12. The first of these parameters is **InterruptAPI**. The interrupt API parameter can be set to either **Enable** (default) or **Disable**. If this option is set to **Enable**, the ISR call for this user module is included in the interrupt dispatch code. A separate interrupt assembly language file is included in the project that contains the ISR code. If this parameter is changed to **Disable**, the ISR call is no longer present in the interrupt vector table, and the automatically generated file is *not* included in the project. If using dynamic reconfiguration and there are overlapping interrupt vectors, one of the user module's interrupts may be disabled if it is not used. If this is done, the interrupt dispatch code (seen in Code 2) directly calls the ISR for the user module that still has an active interrupt. This results in less interrupt latency.

Figure 12. Interrupt Control Parameters




The **InterruptAPI** parameter does not appear (even if this option is enabled) in user modules that have interrupts essential to their functionality. For instance, many ADC user modules must use interrupts to do their conversions. Therefore, this parameter is not available to disable such an ADC's interrupts.

In many cases, both interrupts from two overlapping user modules in two different configurations must be used. This is where the second new parameter may be useful. This is the **IntDispatchMode** parameter. The two options for the parameter are **ActiveStatus** (default) and **OffsetPreCalc**. If a user module has this parameter set to **ActiveStatus**, it functions normally as described in the beginning of this section. By setting this parameter to **OffsetPreCalc** for all of the overlapping user modules, a little RAM is used to speed up the process of dispatching the CPU to the correct ISR. When this option is set, a smaller vector table is generated for the shared interrupt vector. Every time a new configuration is loaded, a function (located in *psocconfig.asm*) executes inside the configuration load function. This function calculates the offset of the correct ISR to jump to in the new, smaller interrupt vector table. This allows the interrupt latency to be shorter and causes each of the ISRs that share the vector to have the same latency.

# D Example Project Instructions

## D.1 Project Objective

This example project demonstrates how to use dynamic reconfiguration.

## D.2 Overview

This project has two overlays and a base configuration. One of the overlays has an RX8 UM placed in it. When this overlay is loaded, the RX8 UM listens for the ADC command byte.

When the ADC command byte is received, the RX overlay is unloaded, and the ADC overlay is loaded. In this overlay, the ADC reads a potentiometer and sends the results out via a TX8 UM that is placed in the same block as the RX8 UM. After the result is sent out, the ADC overlay is unloaded and the RX overlay is loaded again; and the process repeats.

## D.3 User Module List

- RX8
- TX8
- ADCINCVR
- PGA

## D.4 User Module Parameters

### D.4.1 RX8

| Parameter | Value |
|---|---|
| Clock | VC3* |
| Input | Row_0_Input_0 |
| Clock Sync | Sync to SysClk |
| RxCmdBuffer | Disable |

*****The baud rate of the RX8 is the input clock/8. For this example, the input clock is 461 kHz, making the baud rate approximately 57,600.

### D.4.2 TX8

| Parameter | Value |
|---|---|
| Clock | VC3* |
| Input | Row_0_Output_1 |
| Clock Sync | Sync to SysClk |

*The baud rate of the RX8 is the input clock/8. For this example, the input clock is 461 kHz, making the baud rate approximately 57,600.

### D.4.3 ADCINCVR

| Parameter | Value |
|---|---|
| Input | ACB00 |
| Clock | VC1 |
| ADCResolution | 12-bit |
| CalcTime | 20 |
| Data Format | Unsigned |

### D.4.4 PGA

| Parameter | Value |
|---|---|
| Gain | 1 |
| Input | AnalogColumn_InputMUX_0* |
| Reference | AGND |
| AnalogBus | Disable |

*The AnalogColumn_InputMUX_0 should be set to P0.1

## D.5 Relevant Global Resource Settings

- **CPU_Clock** SysClk/1. Set the CPU to operate at the fastest speed for the best performance
- **VC1** 12. This sets the data clock for the ADC to be 2 MHz.
- **VC3 Source** SysClkc/1. VC3 source is the 24-MHz IMO.
- **VC3 Divider 52** This divider sets the input to the RX8 and the TX8 UMs to allow them to operate at 57,600 baud.
- **Analog Power** SC On/Ref HIGH. Set to HIGH for the best ADC performance.
- **RefMux** ($V_{DD}$/2) +/- ($V_{DD}$/2). Set to allow ADC to read entire range of potentiometer input.

## D.6 Operation of Firmware

When the part starts up, the RX configuration is loaded. The firmware then waits for a single byte to be received. When the byte is received, the firmware checks to see if there is an error. If there is an error, it waits for a new byte. If there is no error, the firmware checks to see what the received value is.

If the value is equal to the ADC command, then the firmware unloads the RX configuration, and loads the ADC configuration. After the ADC configuration is loaded, the ADC is started, and a sample is taken. Once the sample is completed, the value is transmitted out via the TX8 UM. After this is completed, the ADC configuration is unloaded, and the process restarts.

There are several things to note: the ADC command is ASCII 1, which is decimal 49. Also, when the ADC configuration is unloaded, the TX8 output has the potential to glitch LOW. To avoid this, the row output LUT is set HIGH before the configuration is unloaded. When the ADC configuration is loaded, the LUT must be set back to output TX8.

## D.7 Hardware Connections

This project is designed to operate on the PSoCEVAL1 Board.

P1[4] > RX (located to the left of the Port 0 header)

P1[5] > TX (located to the left of the Port 0 header)

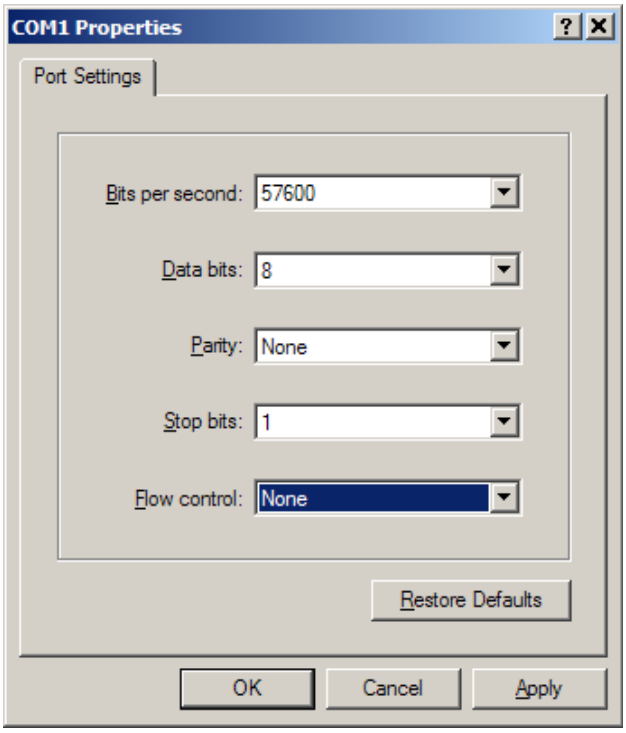
P0[1] > VR

Connect a DB9 cable from the computer to DB9 connector on PSoCEVAL1 board.

## D.8 Operation Instructions

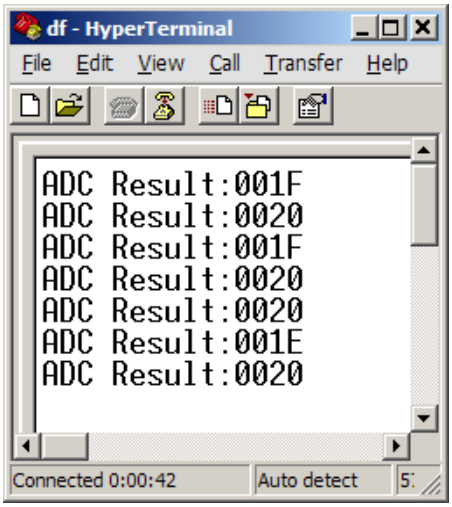
1. Launch HyperTerm or equivalent terminal application and configure it according to Figure 13. Make sure to connect to the appropriate port.

Figure 13. Terminal Settings




2. Once an appropriate connection is made, type a “1” in the terminal window. When you do this, the ADC results should appear as seen in Figure 14:

Figure 14. Terminal Output



# E User Module Placement Limitation in the Base Configuration

There are a few limitations on the placement of certain user modules in the base configuration. Some of the user modules use global resources (such as dedicated $I^2C$ block and decimator) for its function. If these user modules are placed in the base configuration, and an overlay is loaded during runtime, PSoC Designer resets the global resource configuration registers. This causes the user modules to stop working. To overcome this limitation, such user modules should always be placed in an overlay and this overlay should never be unloaded to retain the configuration of the user modules.

This limitation applies to the following user modules:

- $I^2C$-related user modules (I2CHW and EzI2Cs) in all PSoC 1 devices
- ADC user modules (ADCINC, ADCINCVR, DELSIG, and EzADC) that use decimator in all PSoC 1 devices
- CapSense user module (CSD) that uses decimator in CY8C24x94, CY8C28xxx, CY8CLED04, CY8CLED0xD, and CY8CLED0xG.
- All user modules that use dedicated hardware blocks (such as Timer, CapSense, and I2C/SPI) in CY8C20x34/24, CY8C20xx6A, and CY8C20xx7S.

## Document History

Document Title: AN2104 - PSoC® 1 - Dynamic Reconfiguration with PSoC Designer™

Document Number: 001-36000

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | 1514403 | Frank Berkner | 09/27/2007 | New application note. |
| *A | 2512933 | BTK | 06/05/2008 | Extensive rewrite of original application note. Modified document name. Updated content to make the document current. Added additional information on dynamic reconfiguration best practices. Added appendices regarding interrupts, project changes, and Device Editor changes. |
| *B | 3190456 | TDU | 03/08/2011 | Updated Title and Abstract<br>Updated Content to PD5.1<br>Edited Do-It-Yourself Section<br>Added Working Example Project |
| *C | 3288754 | TDU | 06/21/2011 | Added information on using I2C with dynamic reconfiguration.<br>Updated as per template. |
| *D | 3439949 | TDU | 11/16/2011 | Updated Unloading and Loading Configurations section.<br>Updated template. |
| *E | 4442266 | GRAA | 07/14/2014 | Sunset review. |
| *F | 5309188 | RJVB | 06/17/2016 | Added information on using the I2C and ADC (decimator based) in the project with dynamic reconfiguration.<br>Updated the projects to PSoC Designer 5.4 SP1<br>Added Appendix E User module placement limitation in the base configuration<br>Updated template |
| *G | 5713570 | AESATMP9 | 04/26/2017 | Updated logo and copyright. |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| ARM® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Internet of Things | cypress.com/iot |
| Memory | cypress.com/memory |
| Microcontrollers | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| Power Management ICs | cypress.com/pmic |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless Connectivity | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

## Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709