CYPRESS

Connecting From Last Mile To First Mile.™

# Module 4: Dynamic Re-configuration

PERSONAL ACCESS ENTERPRISE METRO CORE

# Objectives:

Section 1: Understanding User Modules

- Data Sheet
- Registers
- User module configuration code

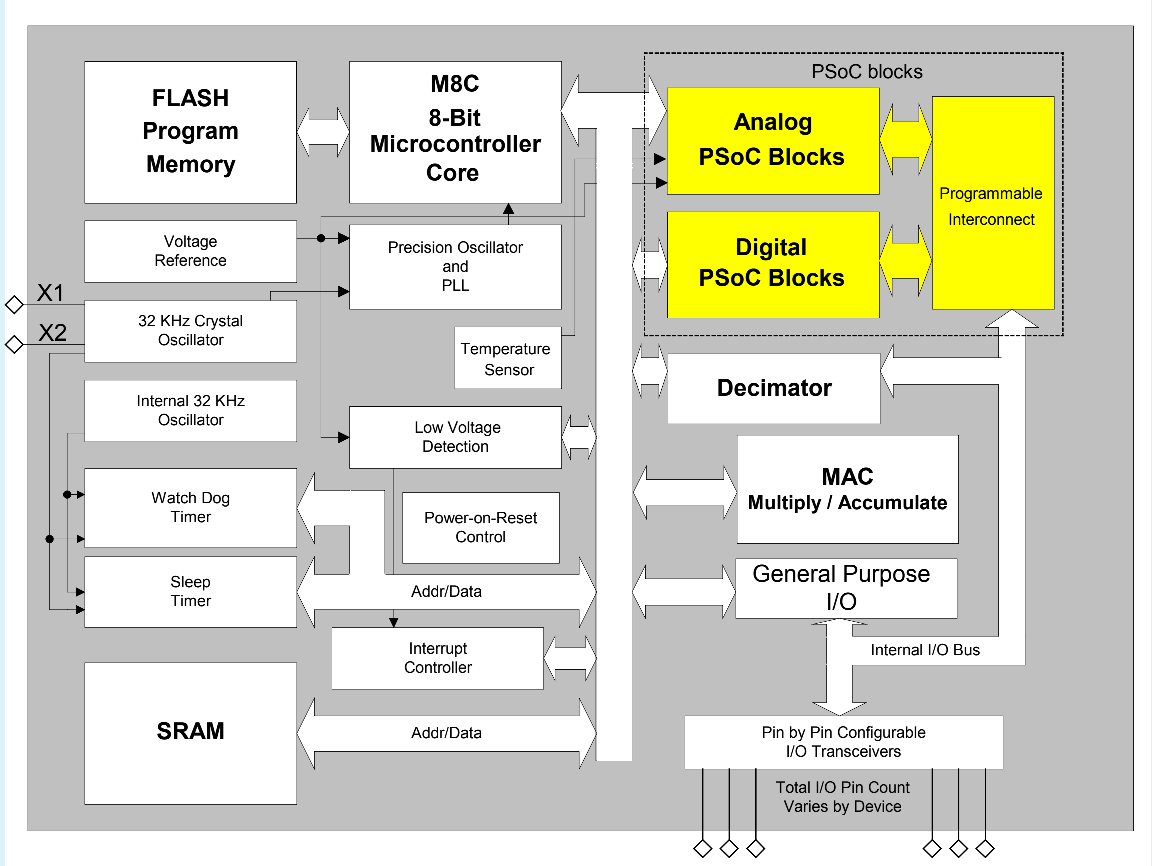
Part 2: Dynamic Re-configuration

- Creating Project
- Device Editor
- Application Editor
- Debugger

Part 3: Dynamic Re-configuration Hands on

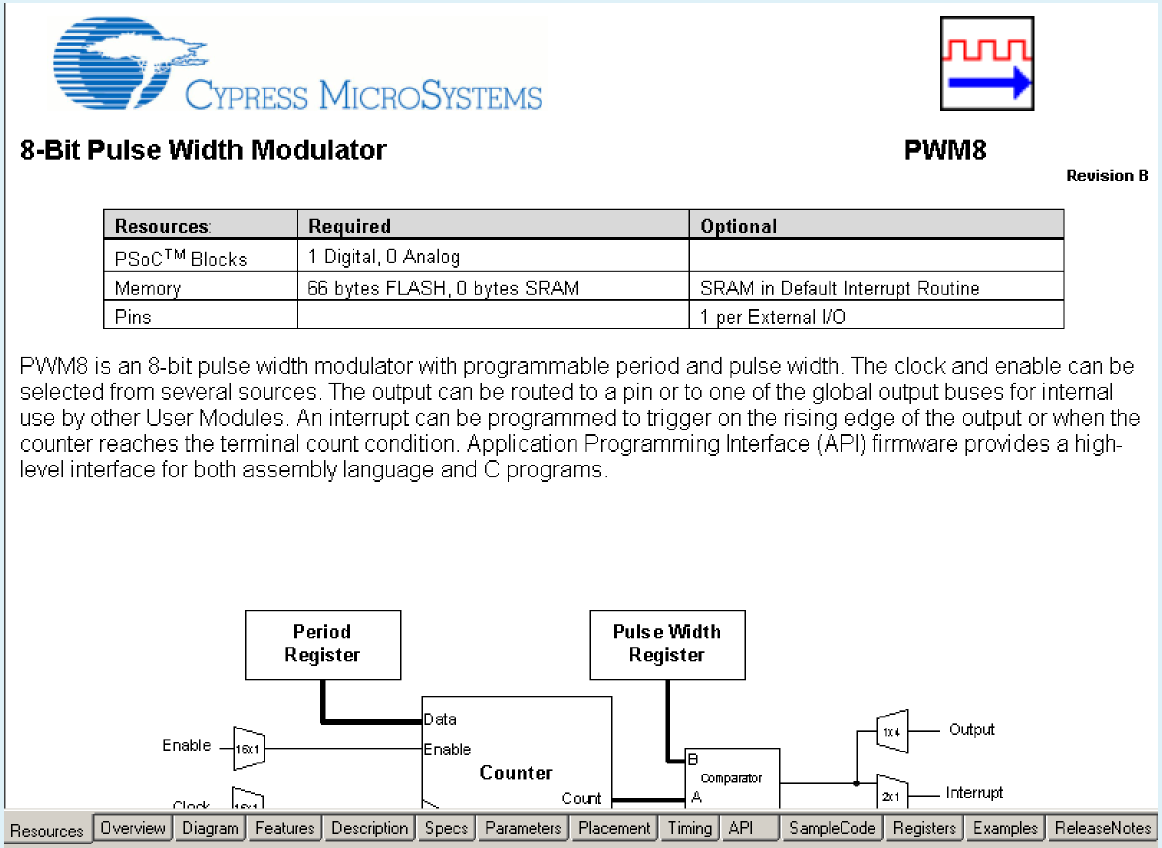
- Half-duplex UART

# User Modules are built from PSoC Blocks



# User Module data sheet sections

- Resources
- Overview
- Diagram
- Features
- Description
- Specs
- Parameters
- Placement
- Timing
- API
- SampleCode
- Registers
- Examples
- ReleaseNotes



CYPRESS MICROSYSTEMS

**8-Bit Pulse Width Modulator** **PWM8**

**Revision B**

| Resources: | Required | Optional |
|---|---|---|
| PSoC™ Blocks | 1 Digital, 0 Analog | |
| Memory | 66 bytes FLASH, 0 bytes SRAM | SRAM in Default Interrupt Routine |
| Pins | | 1 per External I/O |

PWM8 is an 8-bit pulse width modulator with programmable period and pulse width. The clock and enable can be selected from several sources. The output can be routed to a pin or to one of the global output buses for internal use by other User Modules. An interrupt can be programmed to trigger on the rising edge of the output or when the counter reaches the terminal count condition. Application Programming Interface (API) firmware provides a high-level interface for both assembly language and C programs.

# User Module View Options

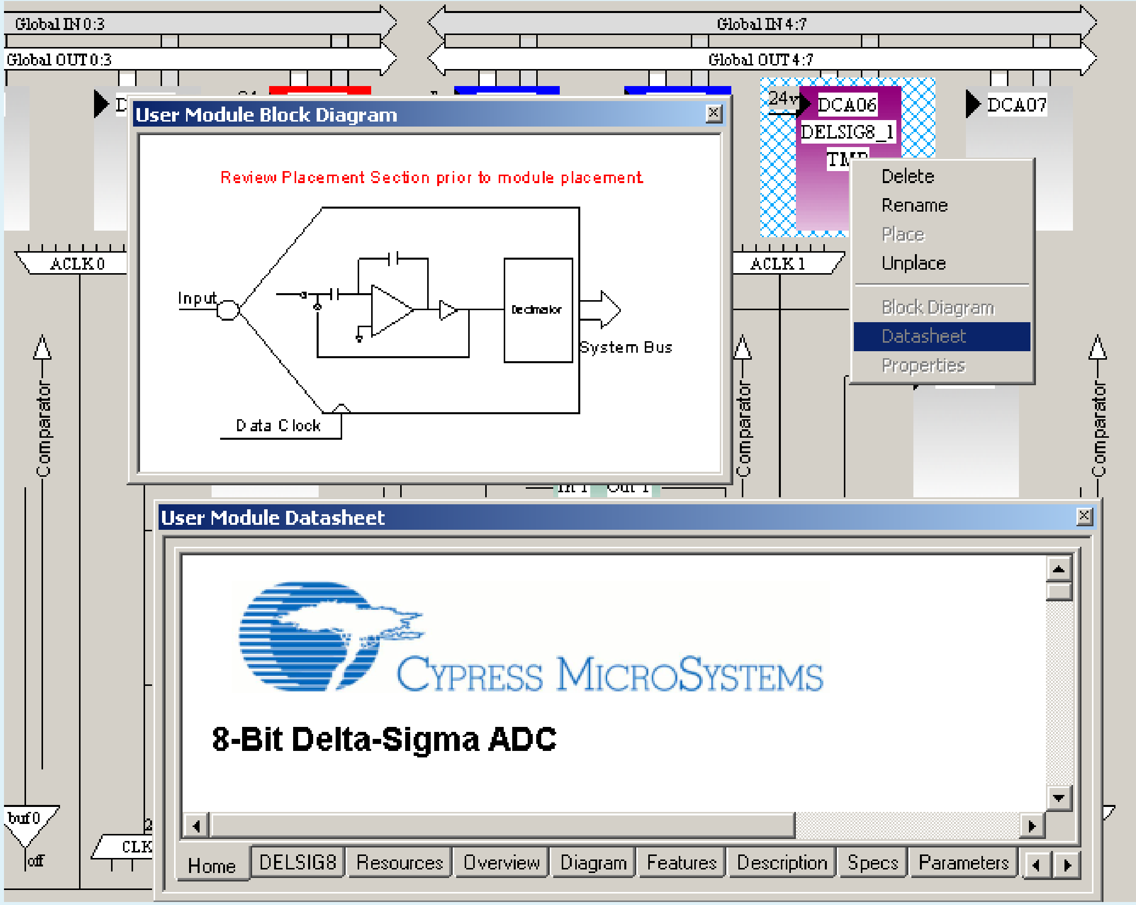
## 3.10 update

**UM Data Sheet**

- Right-click on UM

**UM Block Diagram**

- Right-click on UM



User Module Block Diagram

Review Placement Section prior to module placement.

User Module Datasheet

CYPRESS MICROSYSTEMS

8-Bit Delta-Sigma ADC

# Objectives:

Section 1: Understanding User Modules

- Data sheet
- Registers
- User module configuration code

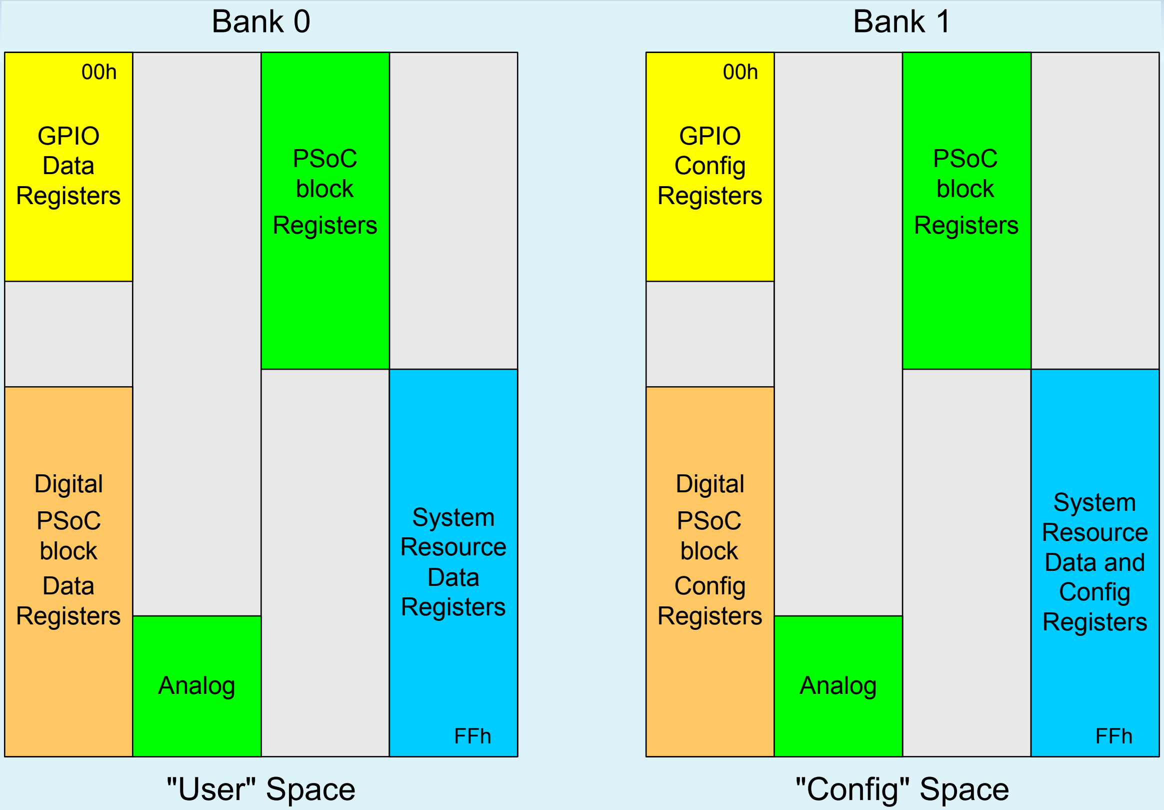
Part 2: Dynamic reconfiguration

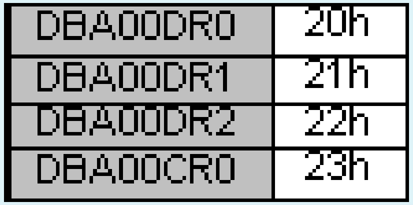
- Creating Project
- Device Editor
- Application Editor
- Debugger

Part 3: Dynamic reconfiguration Hands on

- Half-duplex UART

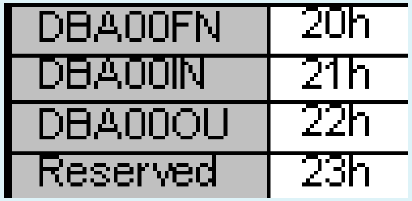
# I/O and register space





| | |
|---|---|
| DBA00DR0 | 20h |
| DBA00DR1 | 21h |
| DBA00DR2 | 22h |
| DBA00CR0 | 23h |

**DBA00 registers**

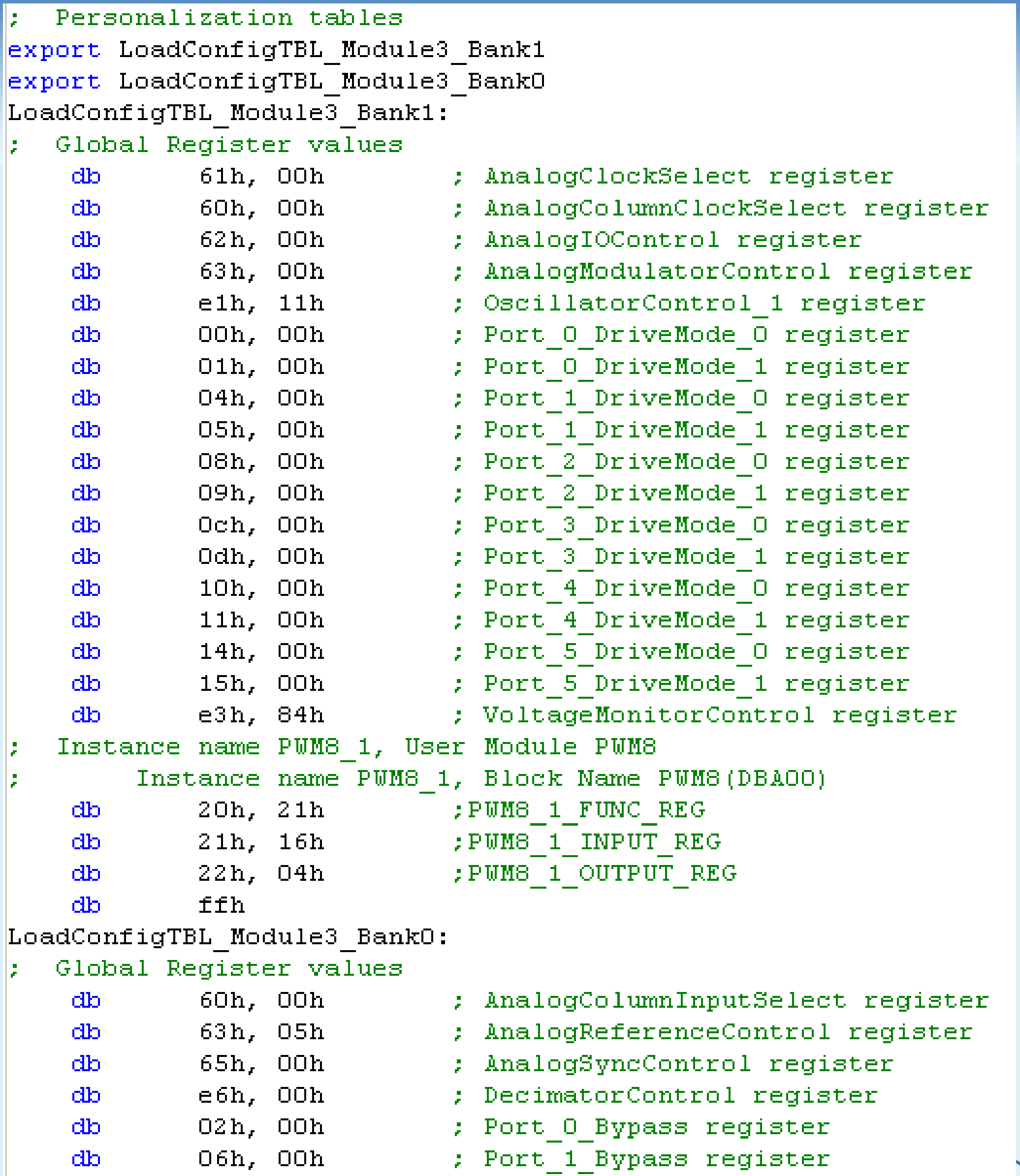


| | |
|---|---|
| DBA00FN | 20h |
| DBA00IN | 21h |
| DBA00OU | 22h |
| Reserved | 23h |

# Registers

## PSoCConfigTBL.asm

- Loads configuration values into registers
- boot.asm loads table values on startup
- Bank 1 values are loaded followed by bank 0
- Each define byte (db) contains a register address and configuration value



```
;  Personalization tables
export LoadConfigTBL_Module3_Bank1
export LoadConfigTBL_Module3_Bank0
LoadConfigTBL_Module3_Bank1:
;  Global Register values
    db      61h, 00h        ;  AnalogClockSelect register
    db      60h, 00h        ;  AnalogColumnClockSelect register
    db      62h, 00h        ;  AnalogIOControl register
    db      63h, 00h        ;  AnalogModulatorControl register
    db      e1h, 11h        ;  OscillatorControl_1 register
    db      00h, 00h        ;  Port_0_DriveMode_0 register
    db      01h, 00h        ;  Port_0_DriveMode_1 register
    db      04h, 00h        ;  Port_1_DriveMode_0 register
    db      05h, 00h        ;  Port_1_DriveMode_1 register
    db      08h, 00h        ;  Port_2_DriveMode_0 register
    db      09h, 00h        ;  Port_2_DriveMode_1 register
    db      0ch, 00h        ;  Port_3_DriveMode_0 register
    db      0dh, 00h        ;  Port_3_DriveMode_1 register
    db      10h, 00h        ;  Port_4_DriveMode_0 register
    db      11h, 00h        ;  Port_4_DriveMode_1 register
    db      14h, 00h        ;  Port_5_DriveMode_0 register
    db      15h, 00h        ;  Port_5_DriveMode_1 register
    db      e3h, 84h        ;  VoltageMonitorControl register
;  Instance name PWM8_1, User Module PWM8
;       Instance name PWM8_1, Block Name PWM8(DBA00)
    db      20h, 21h        ;PWM8_1_FUNC_REG
    db      21h, 16h        ;PWM8_1_INPUT_REG
    db      22h, 04h        ;PWM8_1_OUTPUT_REG
    db      ffh
LoadConfigTBL_Module3_Bank0:
;  Global Register values
    db      60h, 00h        ;  AnalogColumnInputSelect register
    db      63h, 05h        ;  AnalogReferenceControl register
    db      65h, 00h        ;  AnalogSyncControl register
    db      e6h, 00h        ;  DecimatorControl register
    db      02h, 00h        ;  Port_0_Bypass register
    db      06h, 00h        ;  Port_1_Bypass register
```

# Objectives:

Section 1: Understanding User Modules

- Data sheet
- Registers
- User module configuration code

Part 2: Dynamic reconfiguration

- Creating Project
- Device Editor
- Application Editor
- Debugger

Part 3: Dynamic reconfiguration Hands on

- Half-duplex UART

# Interrupt controller

Interrupt Sources and Priorities

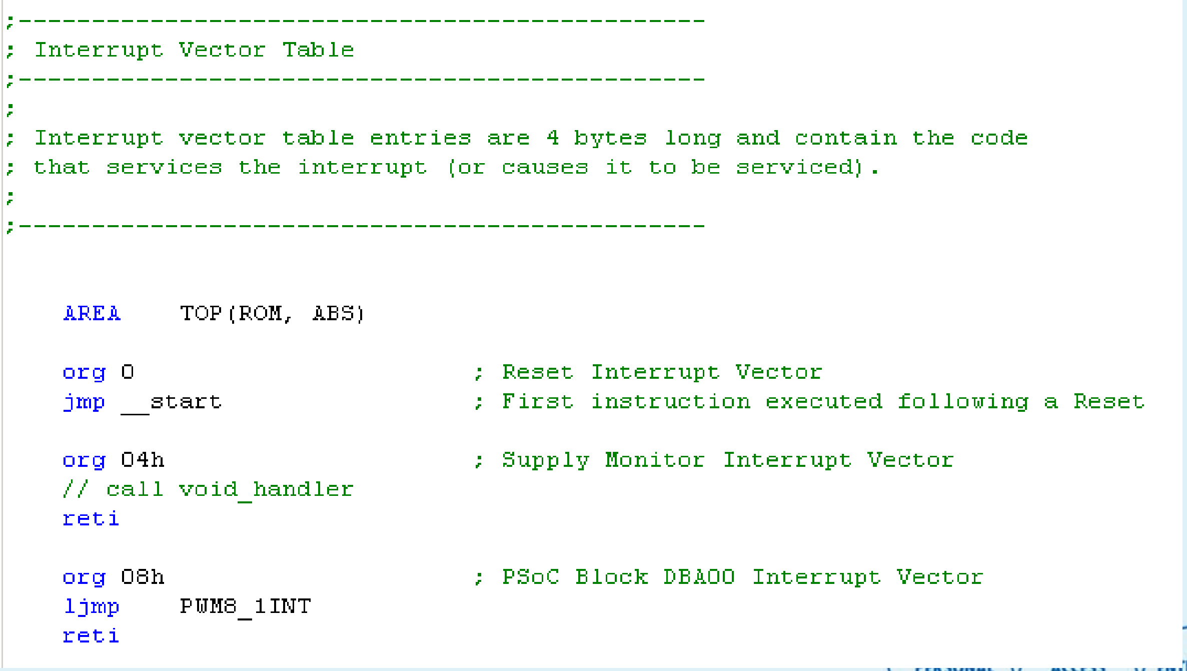
- Power On Reset (POR) and Watch Dog (WDT)
- Power (brown-out) monitor,
- Digital PSoC blocks (8 interrupts total, one per block)
- Analog Comparator Bus (4 interrupts total, one per analog column)
- General Purpose I/O (1 interrupt total, shared by all pins)
- Sleep Timer

Independent mask and enable for each source

# boot.asm interrupt vector table

Each entry is a 4 byte LJMP instruction + RETI

- Because the vectors are in Flash, creative solutions are needed for dynamic reconfiguration of PSoC Blocks when different ISRs are required
- Changes must be made in boot.tpl



```
;-----------------------------------------------
; Interrupt Vector Table
;-----------------------------------------------
;
; Interrupt vector table entries are 4 bytes long and contain the code
; that services the interrupt (or causes it to be serviced).
;
;-----------------------------------------------


    AREA    TOP(ROM, ABS)

    org 0                          ; Reset Interrupt Vector
    jmp __start                    ; First instruction executed following a Reset

    org 04h                        ; Supply Monitor Interrupt Vector
    // call void_handler
    reti

    org 08h                        ; PSoC Block DBA00 Interrupt Vector
    ljmp    PWM8_1INT
    reti
```

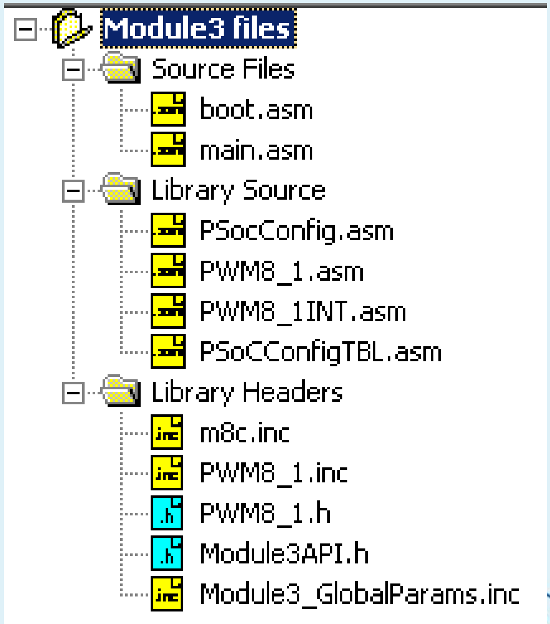
# User Module generated code

User Modules

- User Module interface code PWM8_1.asm
- Interrupt routine PWM8_1INT.asm
- Assembly include file PWM8_1.inc
- C header file PWM8_1.h

Global settings

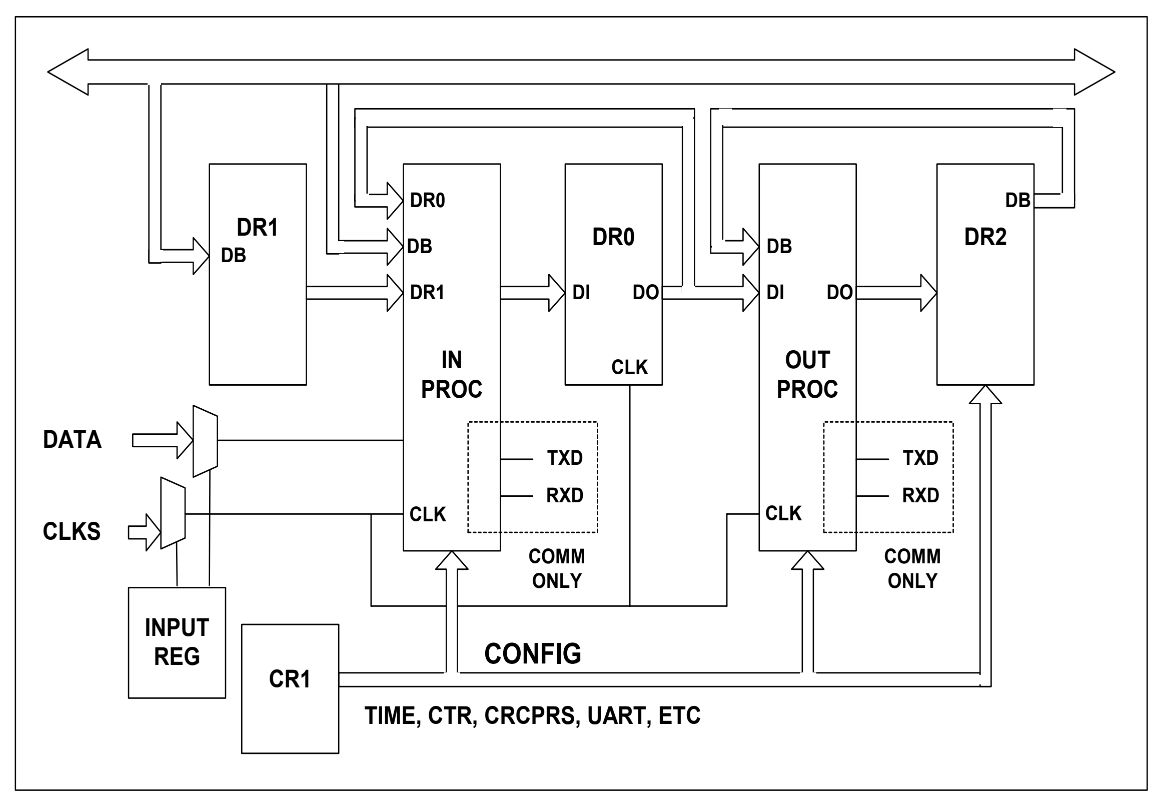
- PSoCConfigTBL.asm
- Module3_GlobalParams.inc



- Module3 files
  - Source Files
    - boot.asm
    - main.asm
  - Library Source
    - PSocConfig.asm
    - PWM8_1.asm
    - PWM8_1INT.asm
    - PSoCConfigTBL.asm
  - Library Headers
    - m8c.inc
    - PWM8_1.inc
    - PWM8_1.h
    - Module3API.h
    - Module3_GlobalParams.inc

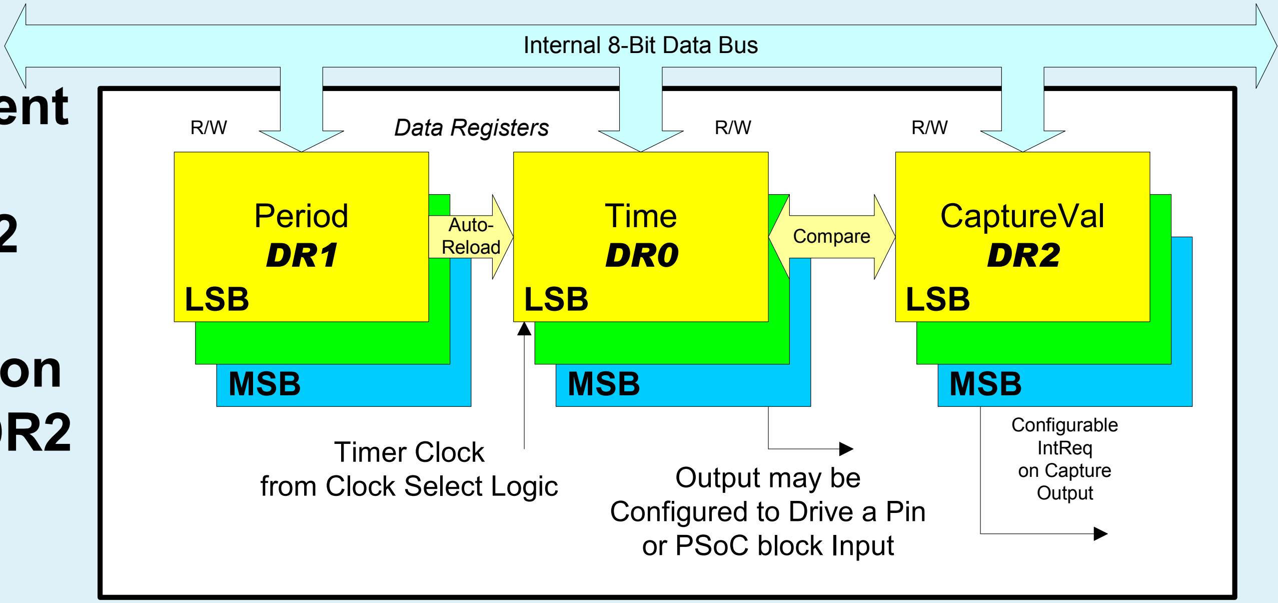
# Digital User Modules

- Eight 8-bit Digital PSoC Blocks Available
- Four Digital Basic Blocks
- Four Digital Comm Blocks



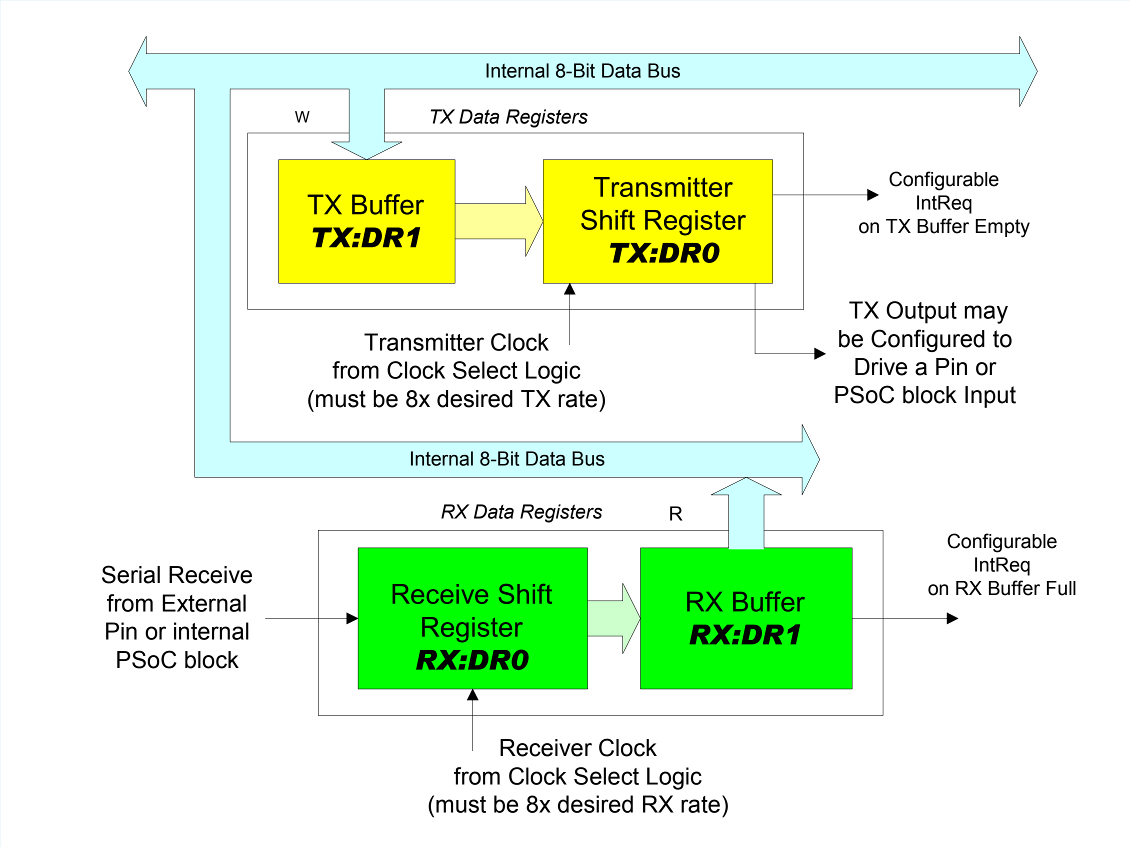
# Timer User Module:

- Measures the time between events
- Data Register 0 (DR0) is a down counter
- HW or SW Capture event latches the current value of DR0 in DR2
- Interrupt on Terminal Count (DR0==0) or on Compare of DR0::DR2



# UART User Module

- Requires Comm type PSoC Block
- 2 PSoC Block for full duplex, 1 for half duplex by dynamic reconfiguration
- Configurable START, STOP bits, PARITY
- Over samples input data by 8x



Internal 8-Bit Data Bus

W

*TX Data Registers*

TX Buffer
***TX:DR1***

Transmitter Shift Register
***TX:DR0***

Configurable IntReq on TX Buffer Empty

Transmitter Clock from Clock Select Logic (must be 8x desired TX rate)

TX Output may be Configured to Drive a Pin or PSoC block Input

Internal 8-Bit Data Bus

*RX Data Registers*

R

Serial Receive from External Pin or internal PSoC block

Receive Shift Register
***RX:DR0***

RX Buffer
***RX:DR1***

Configurable IntReq on RX Buffer Full

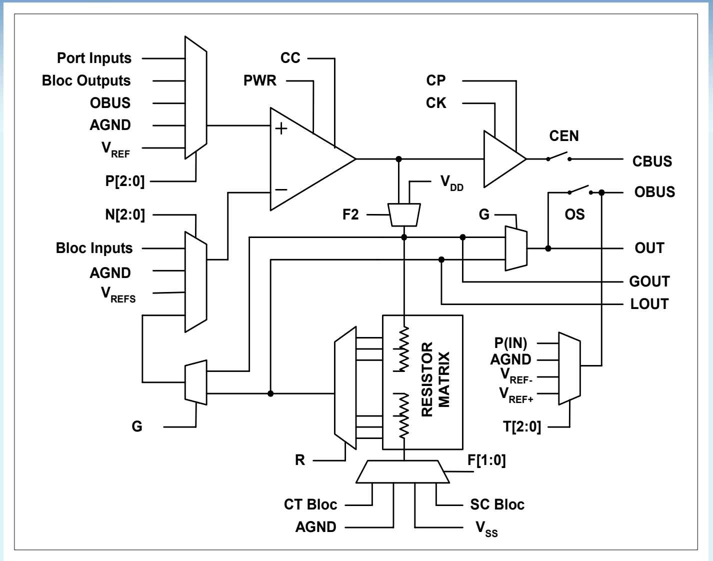
Receiver Clock from Clock Select Logic (must be 8x desired RX rate)

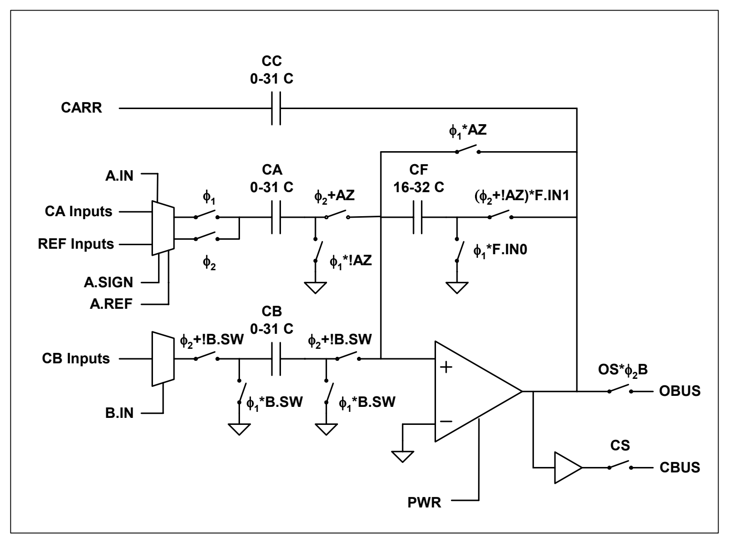
# Analog User Modules

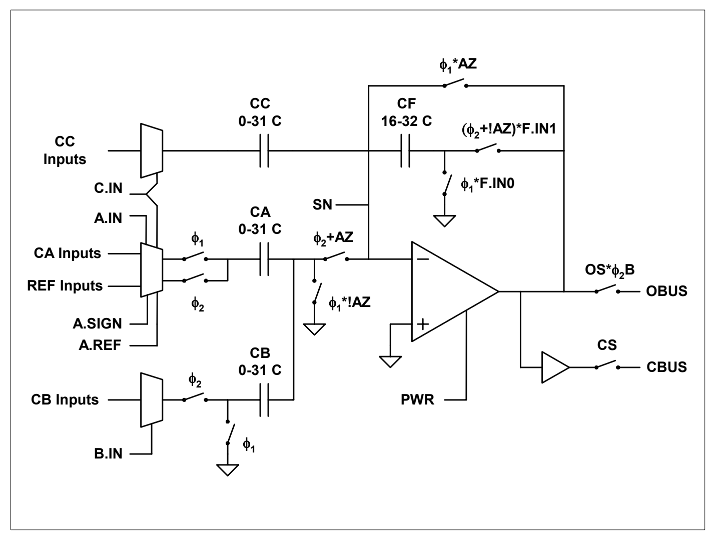
## Continuous Time

## Switched Capacitor Type A

## Switched Capacitor Type B



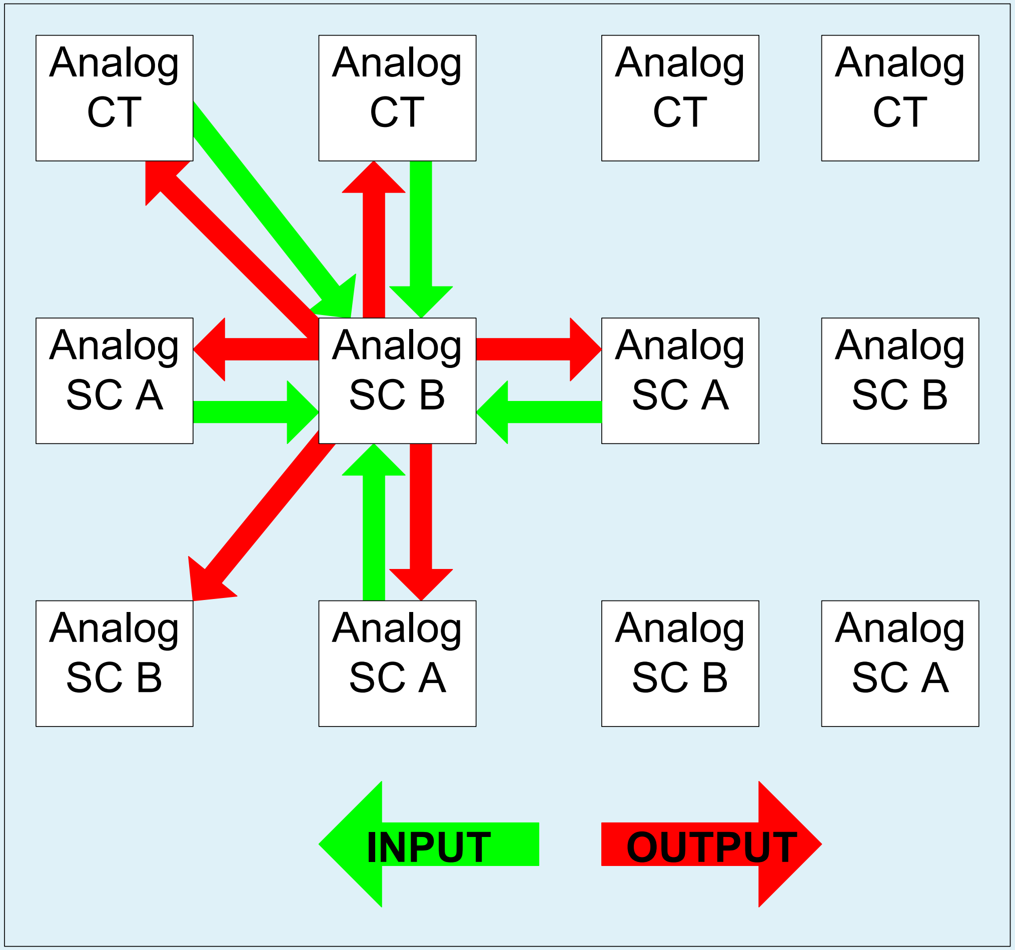




# Local analog interconnect

**SCA to SCB connections support multiple PSoC block User Modules**

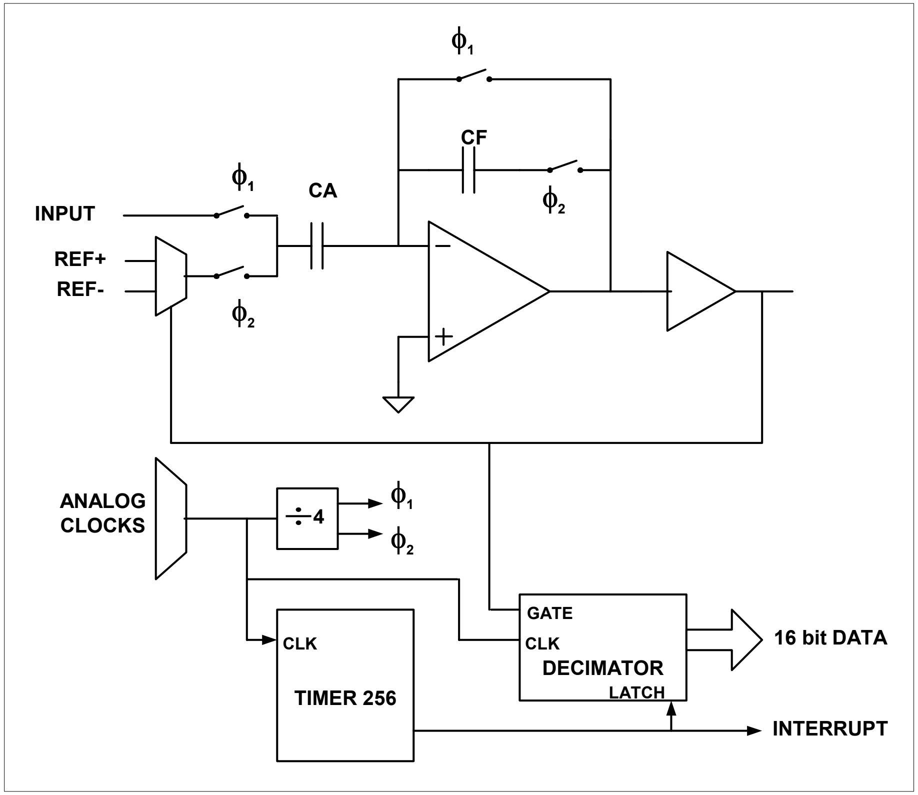
- **Biquad filter**
- **Multiple precision DAC**



# Analog User Module

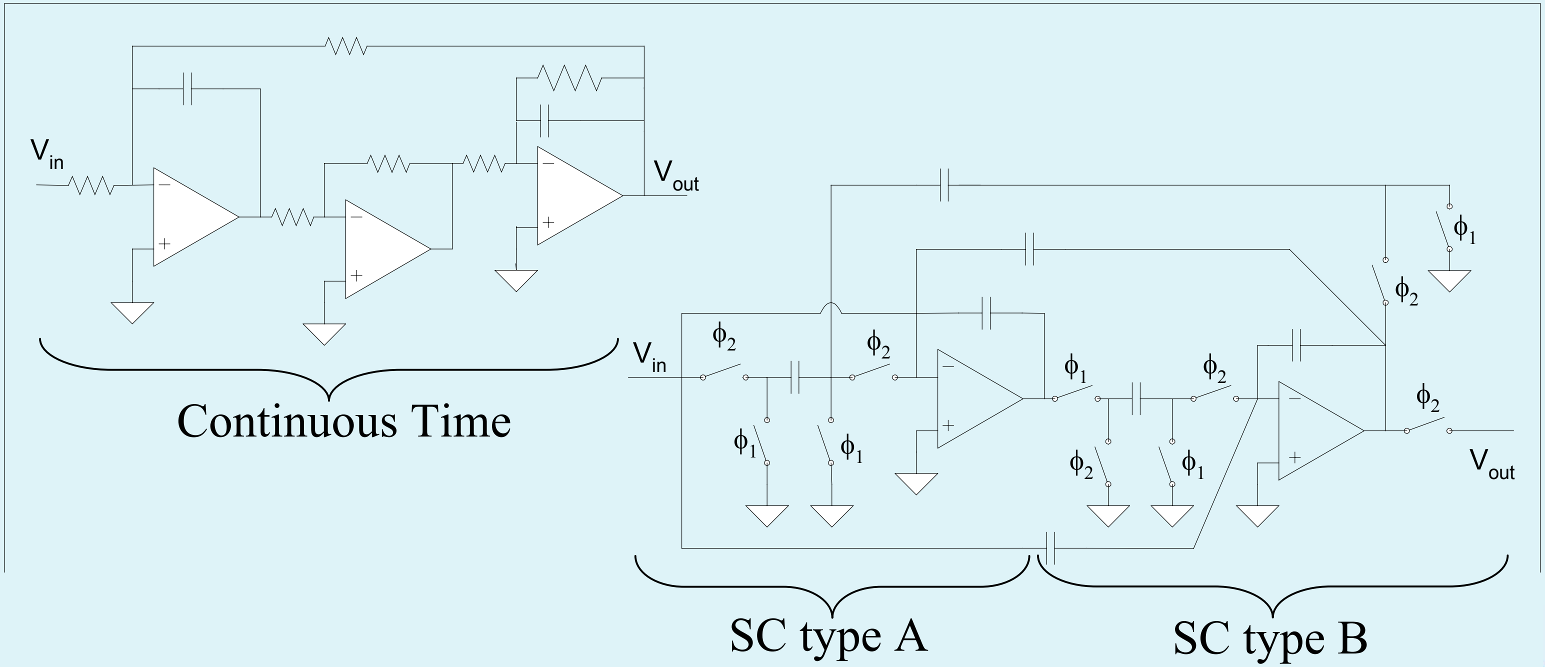
## Δ–Σ ADC

- One SC PSoC block
- One Digital PSoC block
- Hardware Decimator
  - Reduce S/W load



# Filter User Modules

- Integrators are used to form Biquad filters
- Compare the Op-amp efficiency of the switched cap Biquad to its continuous time equivalent



# Objectives:

Section 1: Understanding User Modules

- Data sheet
- Registers
- User module configuration code

Part 2: Dynamic reconfiguration

- Creating Project
- Device Editor
- Application Editor
- Debugger

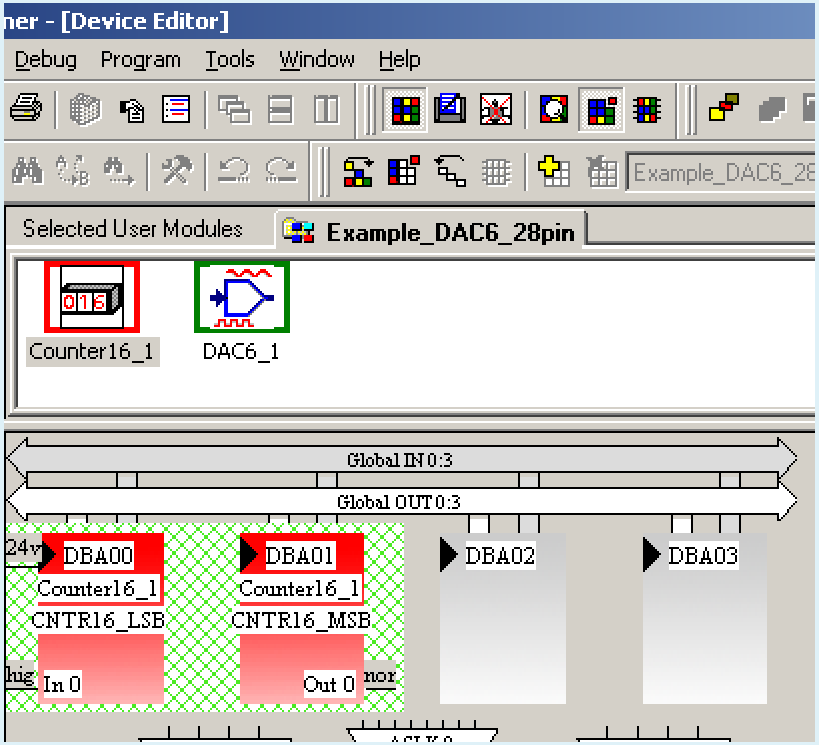
Part 3: Dynamic reconfiguration Hands on

- Half-duplex UART

# Projects without Dynamic Reconfiguration

Only change is the addition of the base configuration tab Example_DAC6_28pin

- Project is created identically as in v2.xx
- No additional resources are used



# Dynamic Reconfiguration Benefits

## Ability to reuse resources

- Example - The equivalent of placing 23 UMs or more in the 20 available PSoC Blocks
- Allows a lower cost part to have the peripheral resources of a much larger part.
- Provides new product features for free
- Allows last minute changes

## Example:

- A coke machine uses most PSoC blocks to receive payment and disperse beverages all day. By dynamically reconfiguring at 2am into a modem it can call headquarters to order a restock. This reconfiguration allows a new level of functionality for the same cost as the basic functionality.

# Creating projects with Dynamic Reconfiguration

1. Start project like a fixed configuration project but only selected and configured the resources required at all times.
2. Partition the remaining resources required into individual configurations based on when they are needed.
3. Create the new configurations and configure the resources
4. Generate application
5. Create project code utilizing configuration APIs for loading and unloading each configuration as needed

# Objectives:

Section 1: Understanding User Modules

- Data sheet
- Registers
- User module configuration code

Part 2: Dynamic reconfiguration

- Creating Project
- Device Editor
- Application Editor
- Debugger

Part 3: Dynamic reconfiguration Hands on

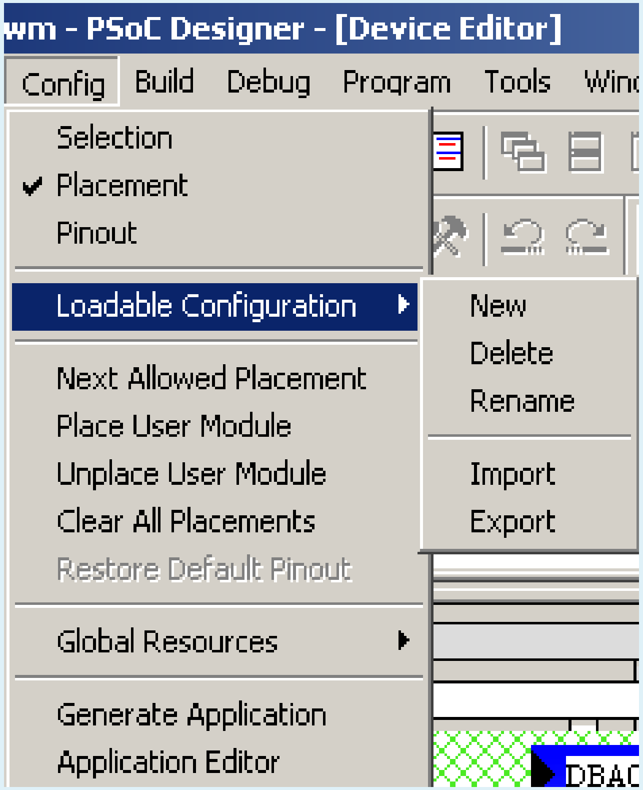
- Half-duplex UART

# Configuration Options

- **Base configuration is created with project**

- **Loadable configurations**
  - Create a new configuration
  - Delete an existing configuration
  - Rename an existing configuration
  - Import a configuration from file
  - Export an existing configuration to file for use in other projects (base configuration can also be exported from any project)

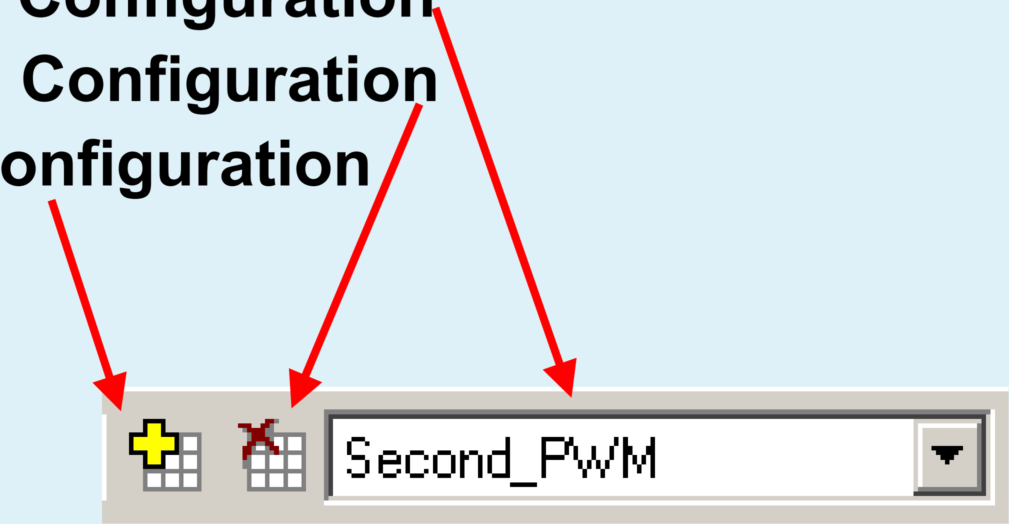
- **Loadable configurations are created just like the base**



# Managing Configurations

## Configuration toolbar

- Select Configuration
- Delete Configuration
- Add Configuration



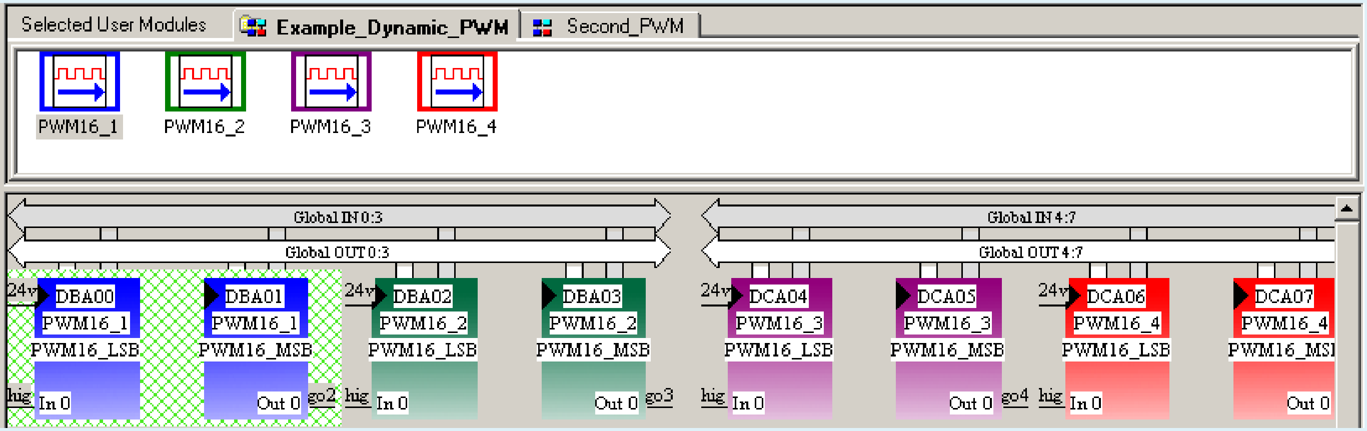
Second_PWM

## Configuration renaming

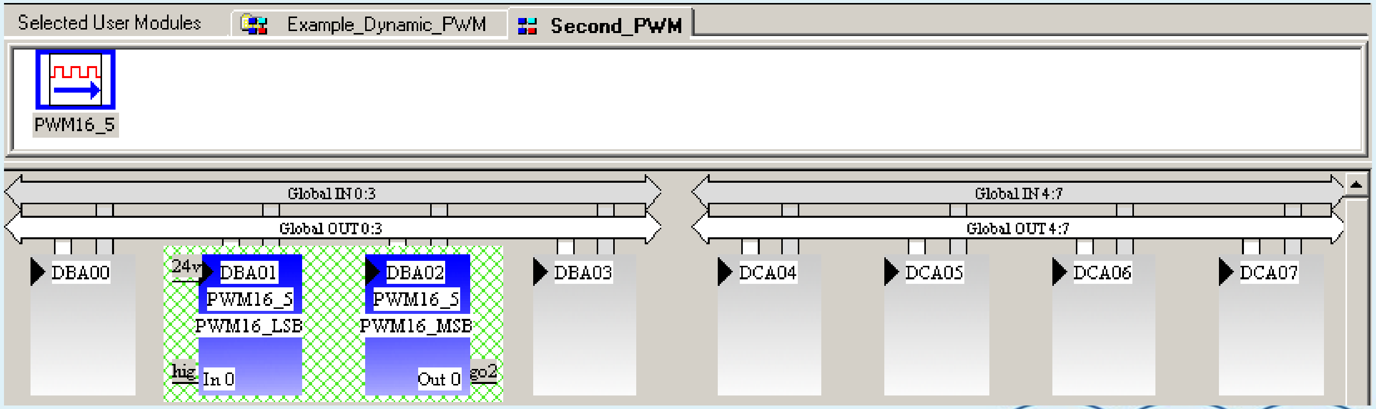
- Right click to rename

# UM placement configuration view

## Base Configuration



## Loadable configuration – UMs must have unique name



# Objectives:

Section 1: Understanding User Modules

- Data sheet
- Registers
- User module configuration code

Part 2: Dynamic reconfiguration

- Creating Project
- Device Editor
- Application Editor
- Debugger

Part 3: Dynamic reconfiguration Hands on

- Half-duplex UART

# Dynamic reconfiguration APIs

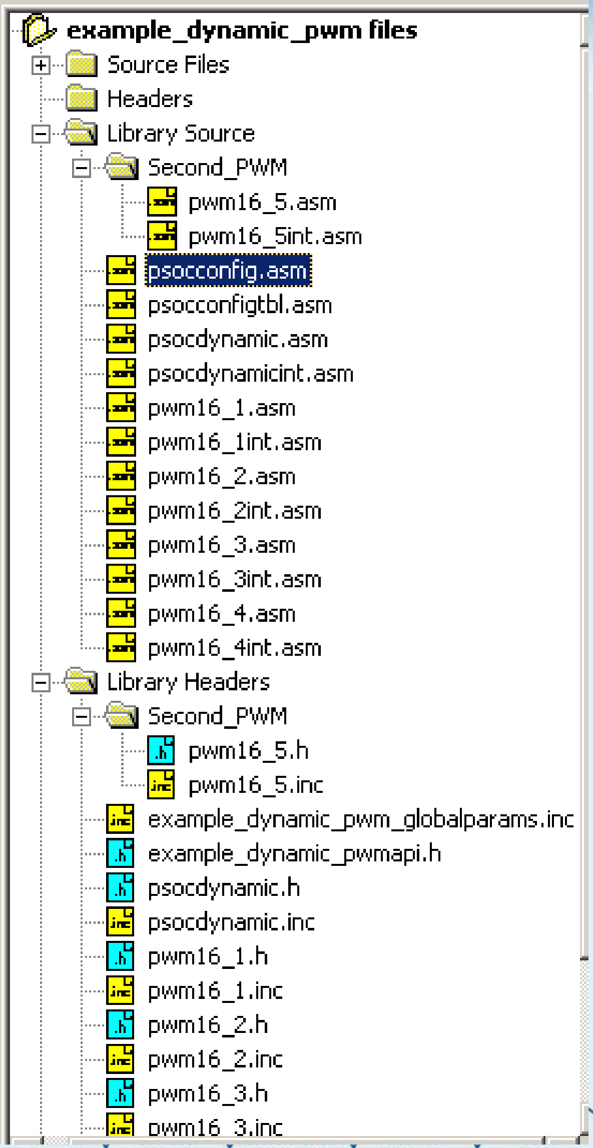
## psocconfigtbl.asm

- Contains all register settings to create each configuration

## psocconfig.asm

## psocdynamic.asm

## psocdynamicint.asm

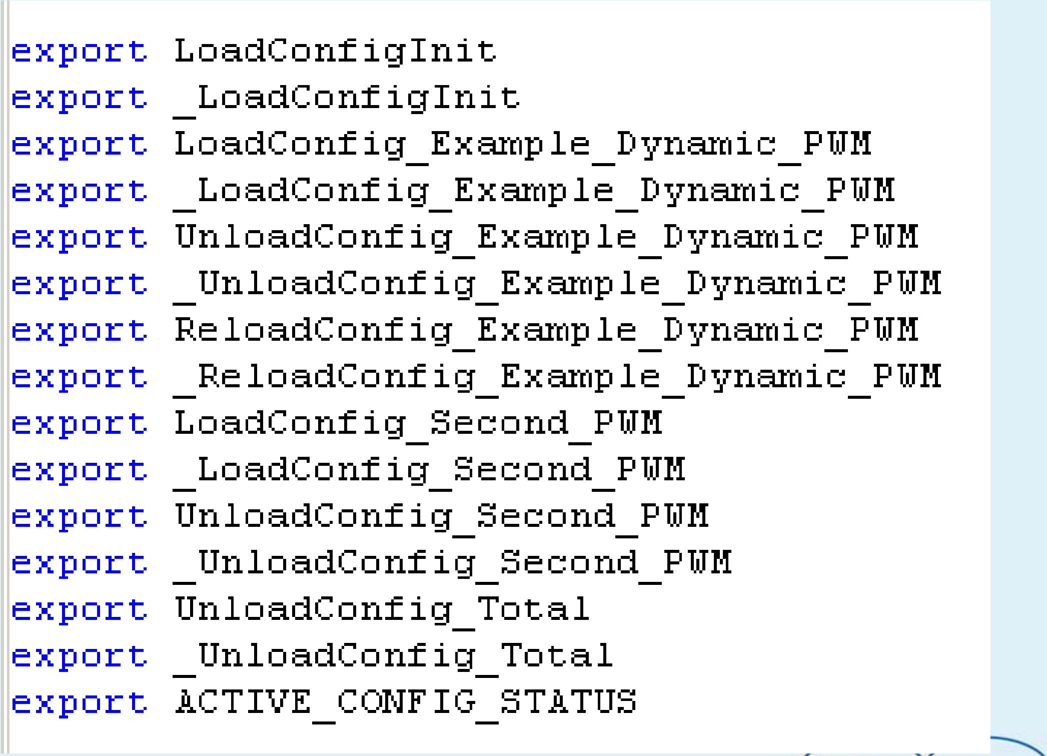


```
example_dynamic_pwm files
  Source Files
  Headers
  Library Source
    Second_PWM
      pwm16_5.asm
      pwm16_5int.asm
    psocconfig.asm
    psocconfigtbl.asm
    psocdynamic.asm
    psocdynamicint.asm
    pwm16_1.asm
    pwm16_1int.asm
    pwm16_2.asm
    pwm16_2int.asm
    pwm16_3.asm
    pwm16_3int.asm
    pwm16_4.asm
    pwm16_4int.asm
  Library Headers
    Second_PWM
      pwm16_5.h
      pwm16_5.inc
    example_dynamic_pwm_globalparams.inc
    example_dynamic_pwmapi.h
    psocdynamic.h
    psocdynamic.inc
    pwm16_1.h
    pwm16_1.inc
    pwm16_2.h
    pwm16_2.inc
    pwm16_3.h
    pwm16_3.inc
```

# psocconfig.asm

## psocconfig.asm

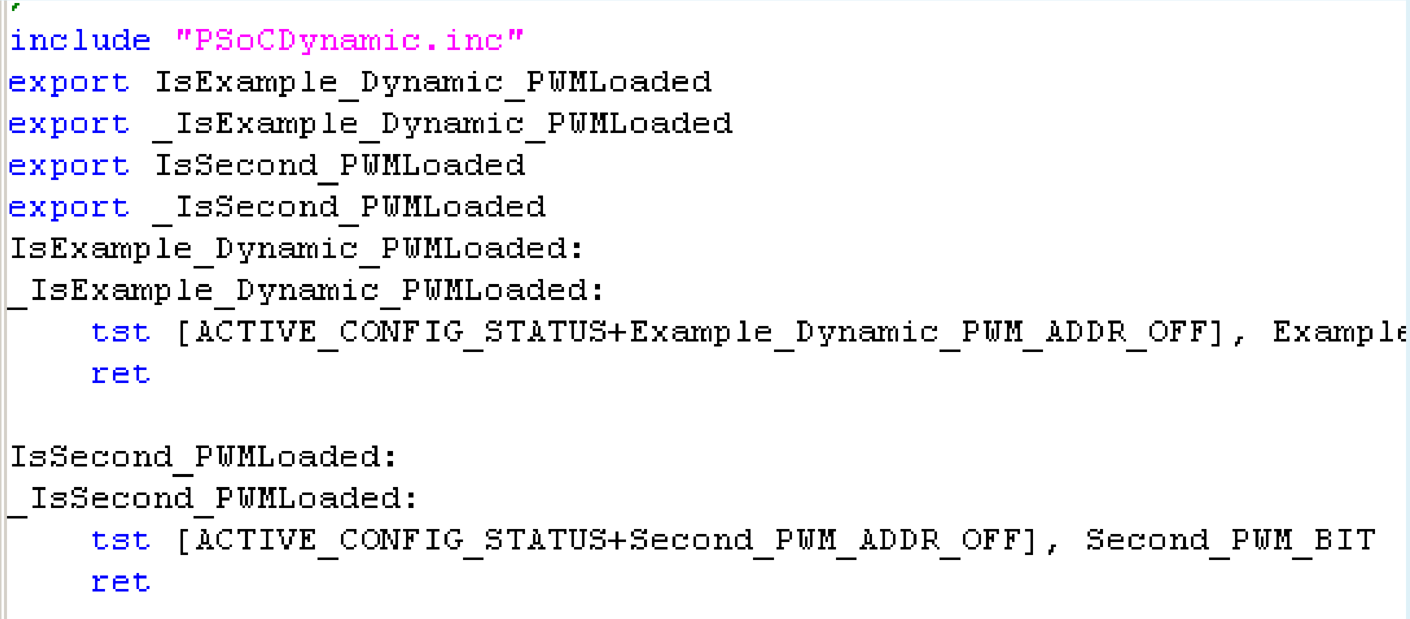
- Automatically loads base configuration
- Provides routines to load and unload all configurations



```
export LoadConfigInit
export _LoadConfigInit
export LoadConfig_Example_Dynamic_PWM
export _LoadConfig_Example_Dynamic_PWM
export UnloadConfig_Example_Dynamic_PWM
export _UnloadConfig_Example_Dynamic_PWM
export ReloadConfig_Example_Dynamic_PWM
export _ReloadConfig_Example_Dynamic_PWM
export LoadConfig_Second_PWM
export _LoadConfig_Second_PWM
export UnloadConfig_Second_PWM
export _UnloadConfig_Second_PWM
export UnloadConfig_Total
export _UnloadConfig_Total
export ACTIVE_CONFIG_STATUS
```

# psocdynamic.asm

## Routines to check which configurations are loaded

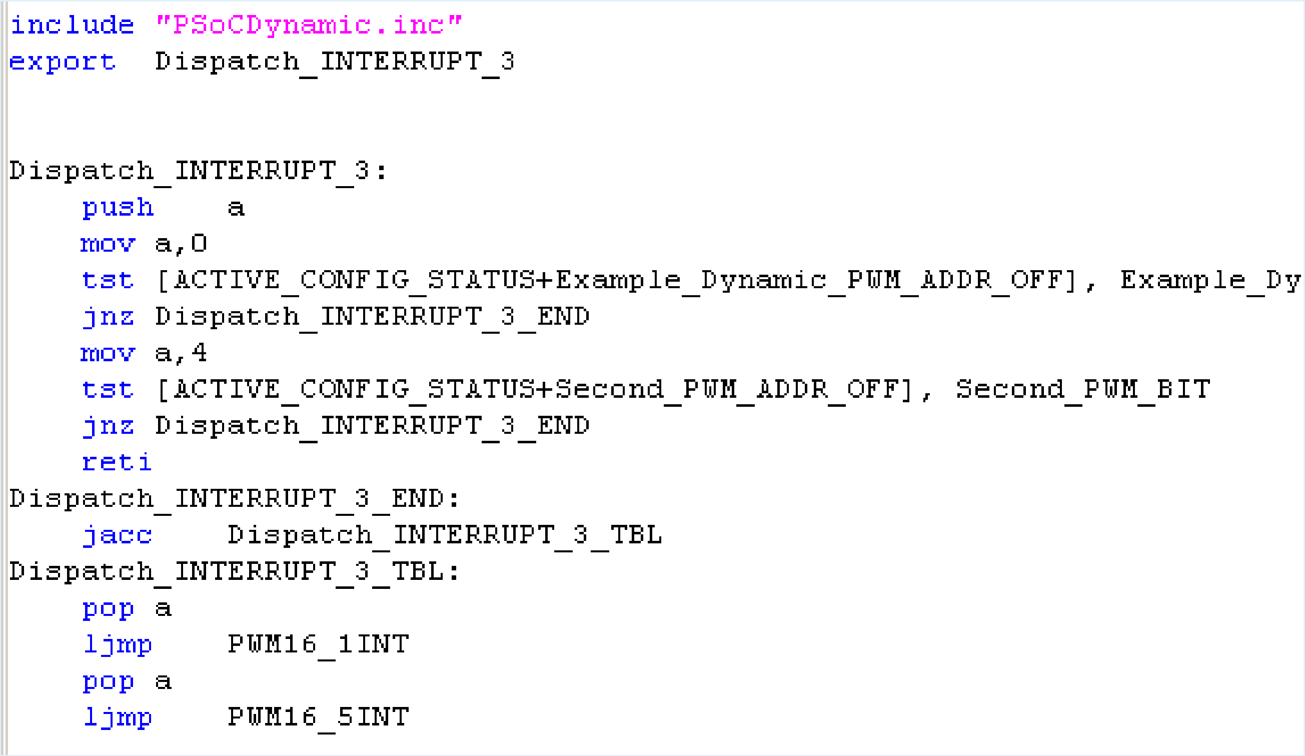


```
,
include "PSoCDynamic.inc"
export IsExample_Dynamic_PWMLoaded
export _IsExample_Dynamic_PWMLoaded
export IsSecond_PWMLoaded
export _IsSecond_PWMLoaded
IsExample_Dynamic_PWMLoaded:
_IsExample_Dynamic_PWMLoaded:
    tst [ACTIVE_CONFIG_STATUS+Example_Dynamic_PWM_ADDR_OFF], Example
    ret

IsSecond_PWMLoaded:
_IsSecond_PWMLoaded:
    tst [ACTIVE_CONFIG_STATUS+Second_PWM_ADDR_OFF], Second_PWM_BIT
    ret
```

# psocdynamicint.asm

## Routes interrupt vectors to current interrupt routine



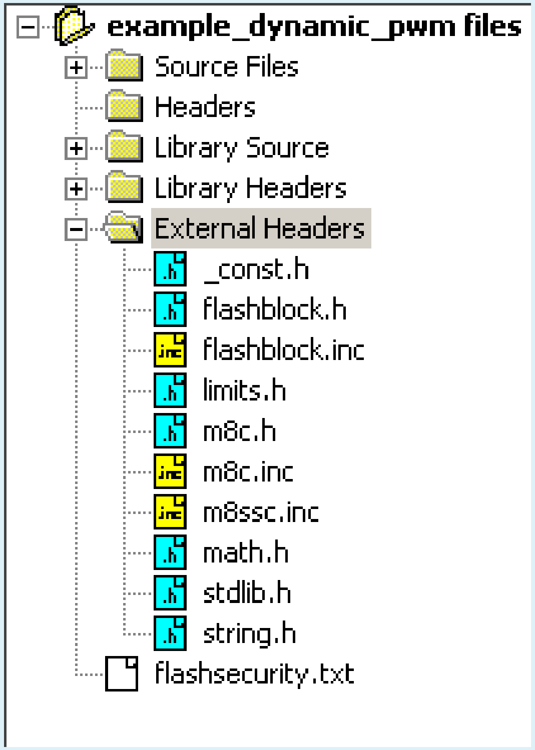
```
include "PSoCDynamic.inc"
export  Dispatch_INTERRUPT_3


Dispatch_INTERRUPT_3:
    push    a
    mov a,0
    tst [ACTIVE_CONFIG_STATUS+Example_Dynamic_PWM_ADDR_OFF], Example_Dy
    jnz Dispatch_INTERRUPT_3_END
    mov a,4
    tst [ACTIVE_CONFIG_STATUS+Second_PWM_ADDR_OFF], Second_PWM_BIT
    jnz Dispatch_INTERRUPT_3_END
    reti
Dispatch_INTERRUPT_3_END:
    jacc    Dispatch_INTERRUPT_3_TBL
Dispatch_INTERRUPT_3_TBL:
    pop a
    ljmp    PWM16_1INT
    pop a
    ljmp    PWM16_5INT
```

# Source Tree

## External Headers

- Flashblock R/W routines
- System Supervisor Commands (SSC)
- math.h
- stlib.h
- string.h



```
⊟ example_dynamic_pwm files
   ⊞ Source Files
     Headers
   ⊞ Library Source
   ⊞ Library Headers
   ⊟ External Headers
        _const.h
        flashblock.h
        flashblock.inc
        limits.h
        m8c.h
        m8c.inc
        m8ssc.inc
        math.h
        stdlib.h
        string.h
     flashsecurity.txt
```

## projectname_globalparams.inc

- All write-only registers are passed
- Simplifies the use of "shadow registers"

## flashsecurity.txt

- Provides block by block security settings for program and data in Flash

# Objectives:

Section 1: Understanding User Modules

- Data sheet
- Registers
- User module configuration code

Part 2: Dynamic reconfiguration

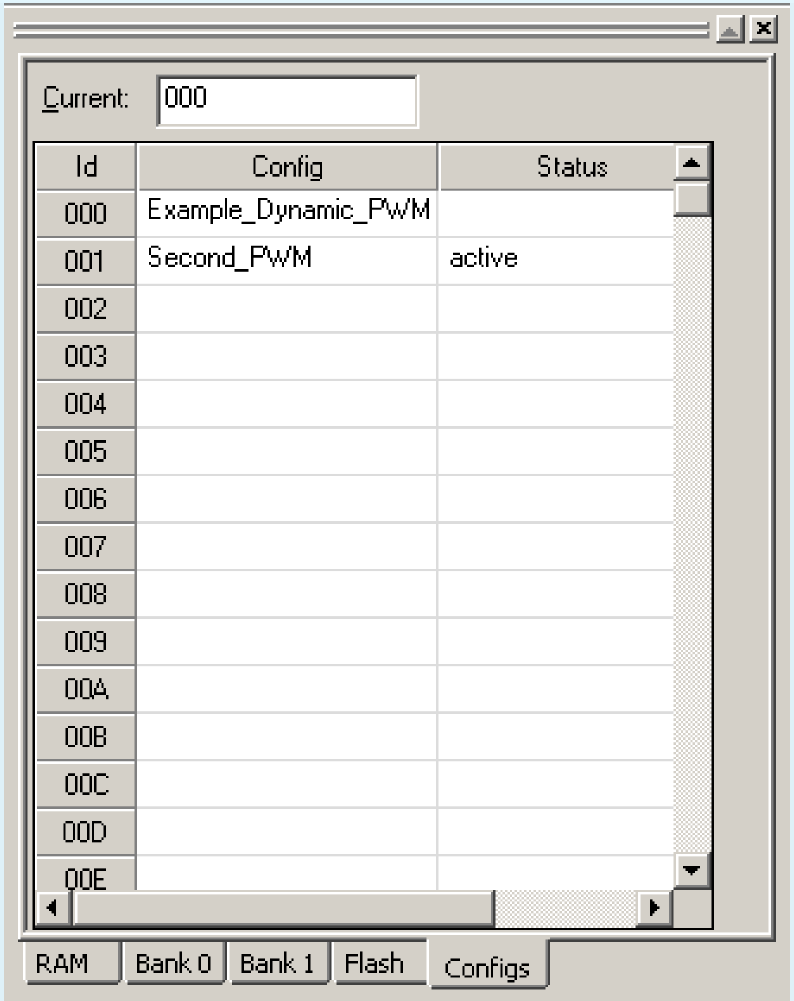
- Creating Project
- Device Editor
- Application Editor
- Debugger

Part 3: Dynamic reconfiguration Hands on

- Half-duplex UART

# Configs window

- List all project configurations
- Reports currently active configuration



Current: 000

| Id | Config | Status |
|---|---|---|
| 000 | Example_Dynamic_PWM | |
| 001 | Second_PWM | active |
| 002 | | |
| 003 | | |
| 004 | | |
| 005 | | |
| 006 | | |
| 007 | | |
| 008 | | |
| 009 | | |
| 00A | | |
| 00B | | |
| 00C | | |
| 00D | | |
| 00E | | |

# Objectives:

## Section 1: Understanding User Modules

- Data sheet
- Registers
- User module configuration code

## Part 2: Dynamic reconfiguration

- Creating Project
- Device Editor
- Application Editor
- Debugger

## Part 3: Dynamic reconfiguration Hands on

- Half-duplex UART

# Half Duplex UART

Full duplex UART requires 2 digital PSoC blocks

Half duplex UART requires only 1 digital PSoC block

Utilize generated User Module code to create a dynamically reconfigureable half duplex UART

# Create Project

Start a new project titled module3

Select the 28 PDIP default device and assembly

Create loadable configurations

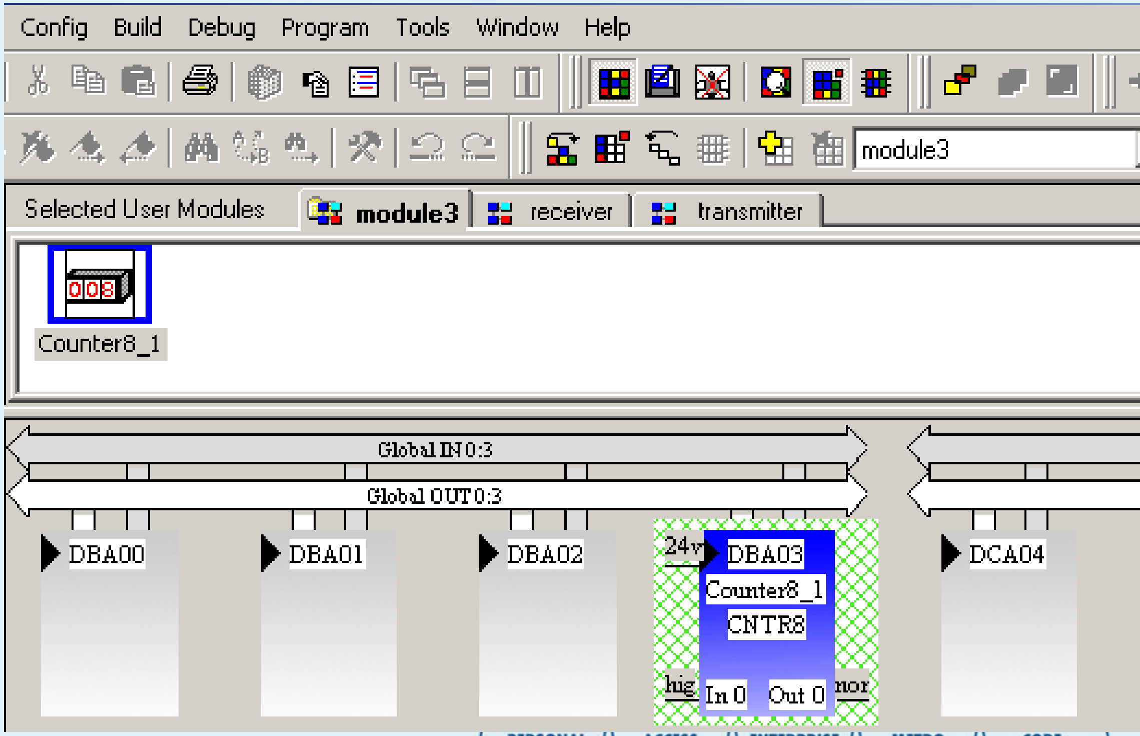
- receiver
- transmitter

Select User Module

- Counter8

Place User Module

- Counter8 – DBA03

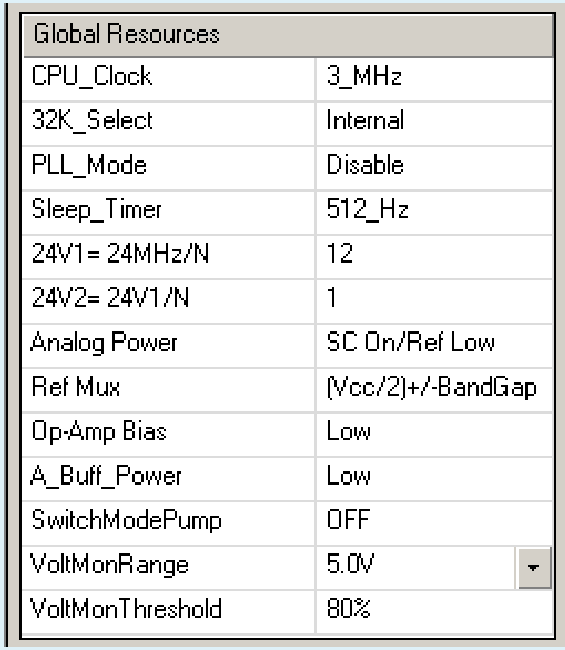


# Configure Base Configuration Resources

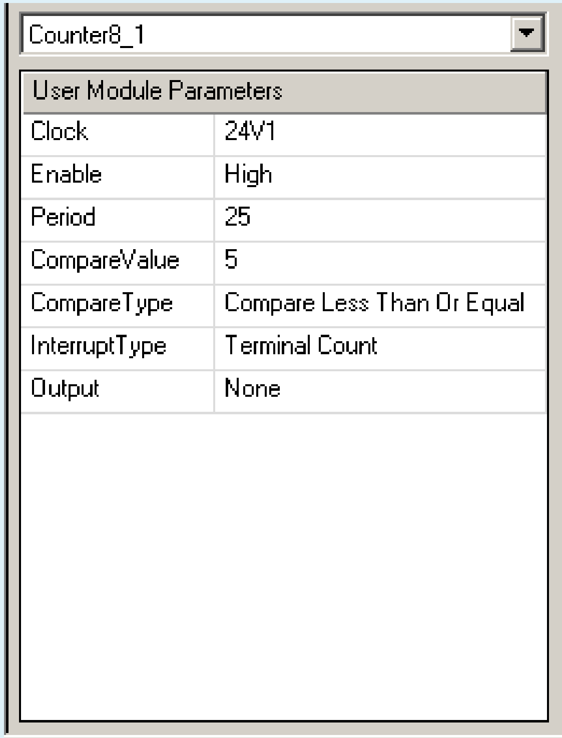
## Configure Global Resources

## Configure User Module parameters

- **9615 Baud = 24 MHz / 12 (24V1) / 26 (Counter8) / 8 (RX8)**



| Global Resources | |
|---|---|
| CPU_Clock | 3_MHz |
| 32K_Select | Internal |
| PLL_Mode | Disable |
| Sleep_Timer | 512_Hz |
| 24V1= 24MHz/N | 12 |
| 24V2= 24V1/N | 1 |
| Analog Power | SC On/Ref Low |
| Ref Mux | (Vcc/2)+/-BandGap |
| Op-Amp Bias | Low |
| A_Buff_Power | Low |
| SwitchModePump | OFF |
| VoltMonRange | 5.0V |
| VoltMonThreshold | 80% |



Counter8_1

| User Module Parameters | |
|---|---|
| Clock | 24V1 |
| Enable | High |
| Period | 25 |
| CompareValue | 5 |
| CompareType | Compare Less Than Or Equal |
| InterruptType | Terminal Count |
| Output | None |

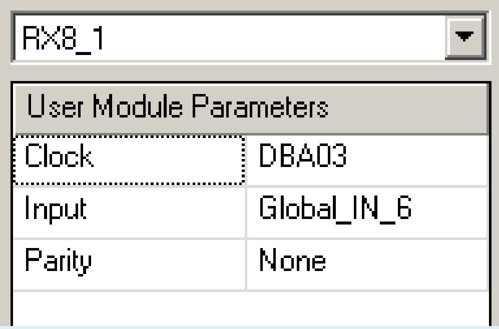
# Configure Receiver Configuration

## Select User Module

- RX8

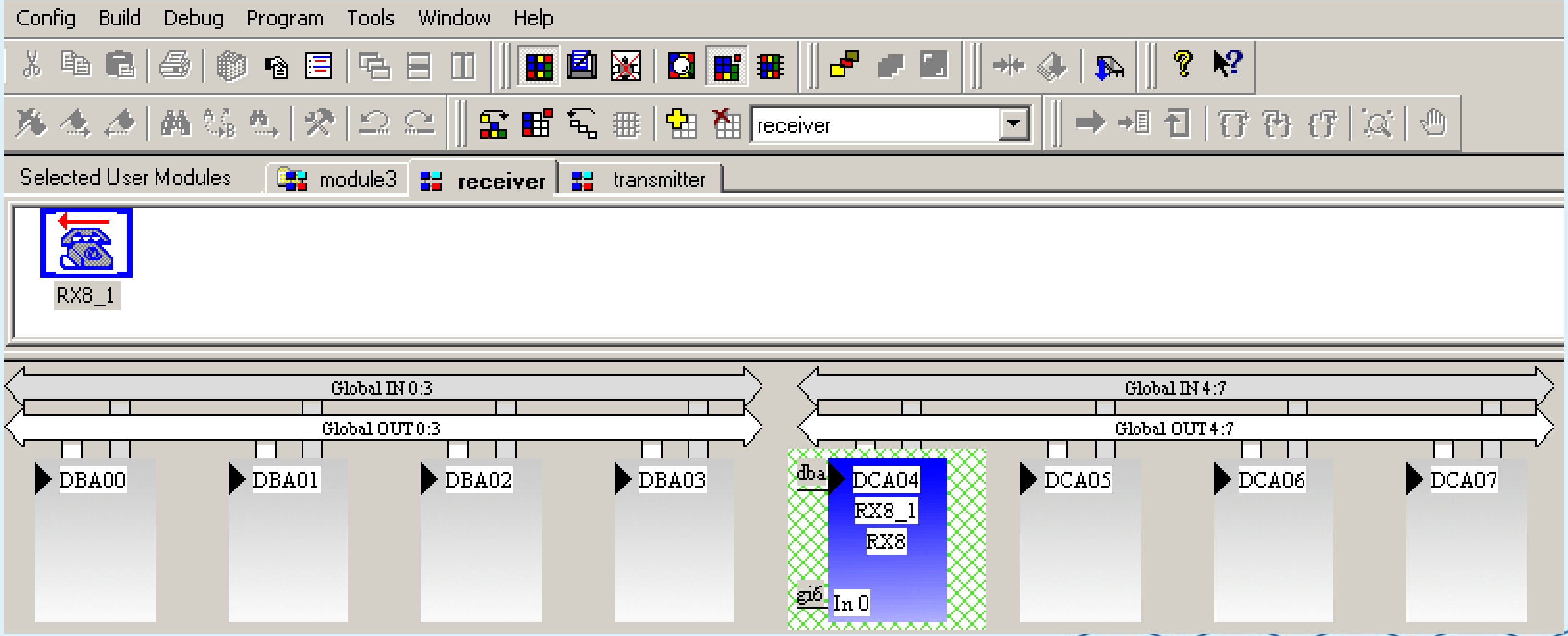
## Place User Module

- RX8 – DCA04



RX8_1

| User Module Parameters | |
|---|---|
| Clock | DBA03 |
| Input | Global_IN_6 |
| Parity | None |



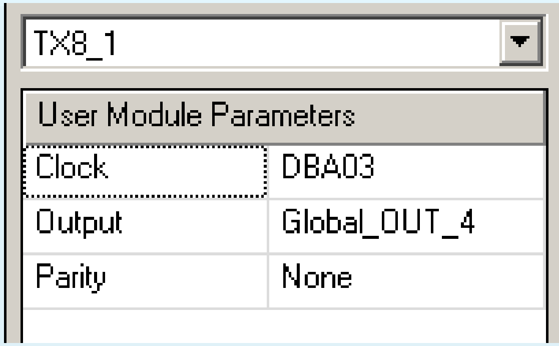
# Configure Transmitter Configuration

## Select User Module

- TX8

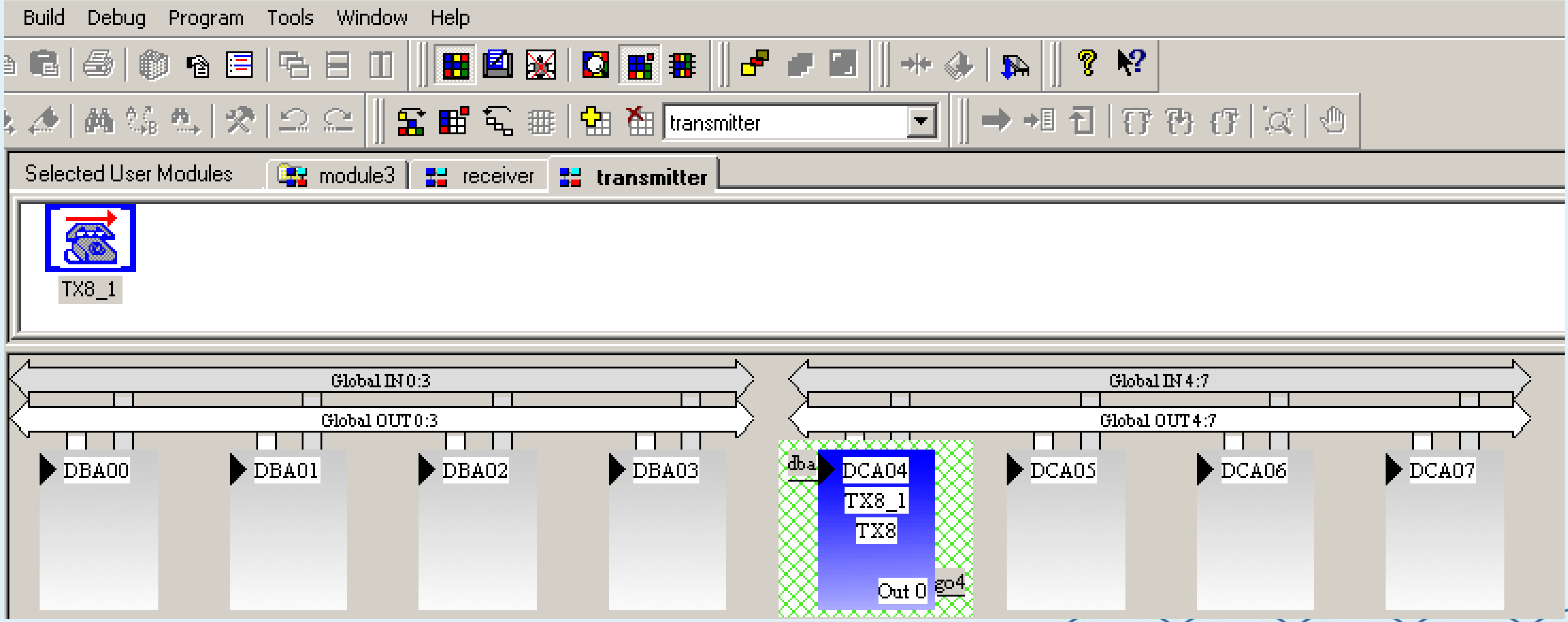
## Place User Module

- TX8 – DCA04



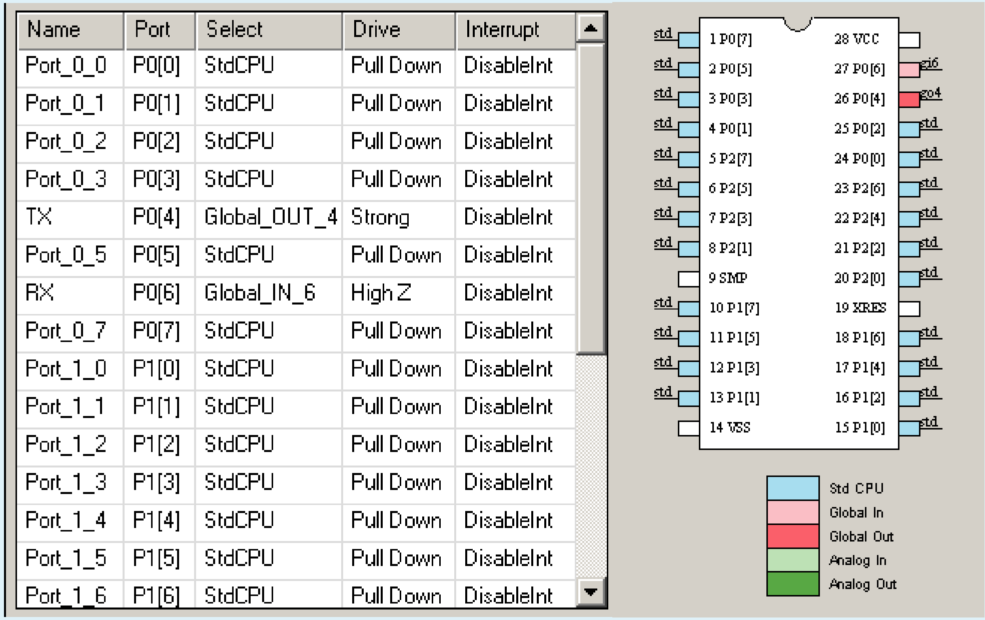
TX8_1

| User Module Parameters | |
|---|---|
| Clock | DBA03 |
| Output | Global_OUT_4 |
| Parity | None |



# Configure pinout

## RX = pin 0[6] = Global_IN_6
## TX = pin 0[4] = Global_OUT_4(Strong)



| Name | Port | Select | Drive | Interrupt |
|---|---|---|---|---|
| Port_0_0 | P0[0] | StdCPU | Pull Down | DisableInt |
| Port_0_1 | P0[1] | StdCPU | Pull Down | DisableInt |
| Port_0_2 | P0[2] | StdCPU | Pull Down | DisableInt |
| Port_0_3 | P0[3] | StdCPU | Pull Down | DisableInt |
| TX | P0[4] | Global_OUT_4 | Strong | DisableInt |
| Port_0_5 | P0[5] | StdCPU | Pull Down | DisableInt |
| RX | P0[6] | Global_IN_6 | High Z | DisableInt |
| Port_0_7 | P0[7] | StdCPU | Pull Down | DisableInt |
| Port_1_0 | P1[0] | StdCPU | Pull Down | DisableInt |
| Port_1_1 | P1[1] | StdCPU | Pull Down | DisableInt |
| Port_1_2 | P1[2] | StdCPU | Pull Down | DisableInt |
| Port_1_3 | P1[3] | StdCPU | Pull Down | DisableInt |
| Port_1_4 | P1[4] | StdCPU | Pull Down | DisableInt |
| Port_1_5 | P1[5] | StdCPU | Pull Down | DisableInt |
| Port_1_6 | P1[6] | StdCPU | Pull Down | DisableInt |

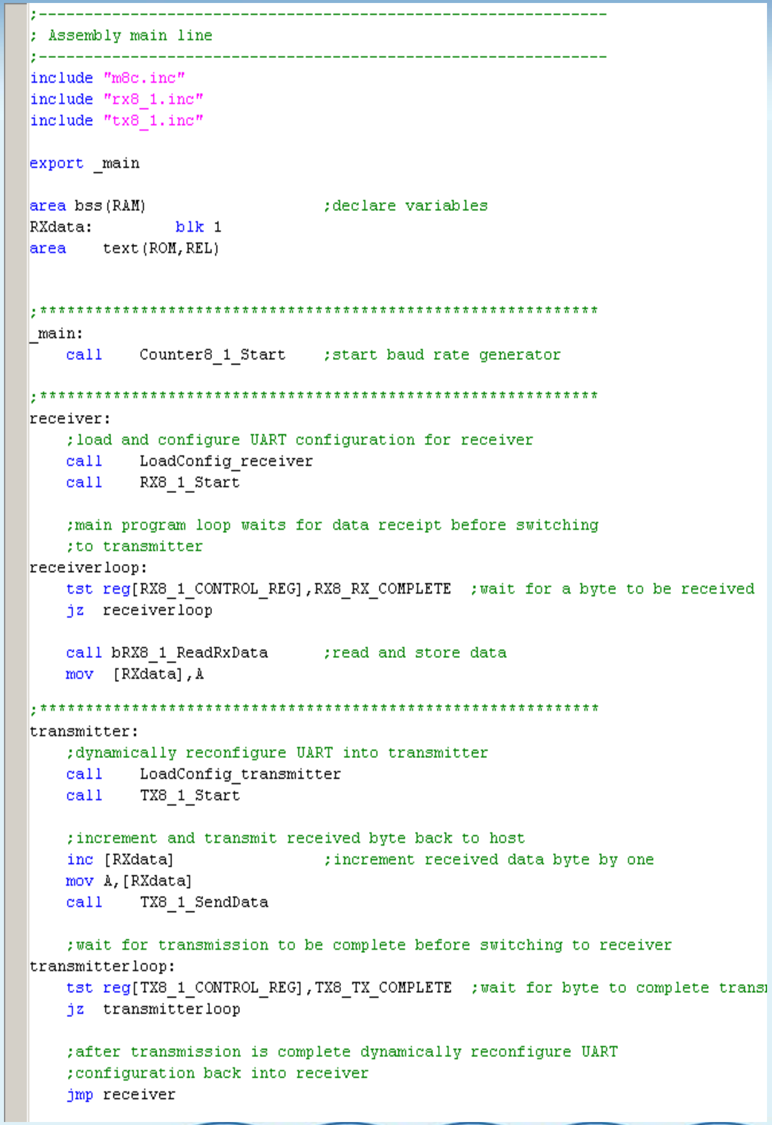
## Generate Application
## Enter Application Editor

# Complete UART

## Cut and Paste code into Application

- Delete all main.asm code
- The file CodeExcerpts3.txt contains assembly code to complete the project
- Select and insert all code into main.asm

## Build project



```
;--------------------------------------------------------------
; Assembly main line
;--------------------------------------------------------------
include "m8c.inc"
include "rx8_1.inc"
include "tx8_1.inc"

export _main

area bss(RAM)                   ;declare variables
RXdata:         blk 1
area    text(ROM,REL)


;*************************************************************
_main:
    call    Counter8_1_Start    ;start baud rate generator

;*************************************************************
receiver:
    ;load and configure UART configuration for receiver
    call    LoadConfig_receiver
    call    RX8_1_Start

    ;main program loop waits for data receipt before switching
    ;to transmitter
receiverloop:
    tst reg[RX8_1_CONTROL_REG],RX8_RX_COMPLETE  ;wait for a byte to be received
    jz  receiverloop

    call bRX8_1_ReadRxData      ;read and store data
    mov  [RXdata],A

;*************************************************************
transmitter:
    ;dynamically reconfigure UART into transmitter
    call    LoadConfig_transmitter
    call    TX8_1_Start

    ;increment and transmit received byte back to host
    inc [RXdata]                ;increment received data byte by one
    mov A,[RXdata]
    call    TX8_1_SendData

    ;wait for transmission to be complete before switching to receiver
transmitterloop:
    tst reg[TX8_1_CONTROL_REG],TX8_TX_COMPLETE  ;wait for byte to complete trans
    jz  transmitterloop

    ;after transmission is complete dynamically reconfigure UART
    ;configuration back into receiver
    jmp receiver
```

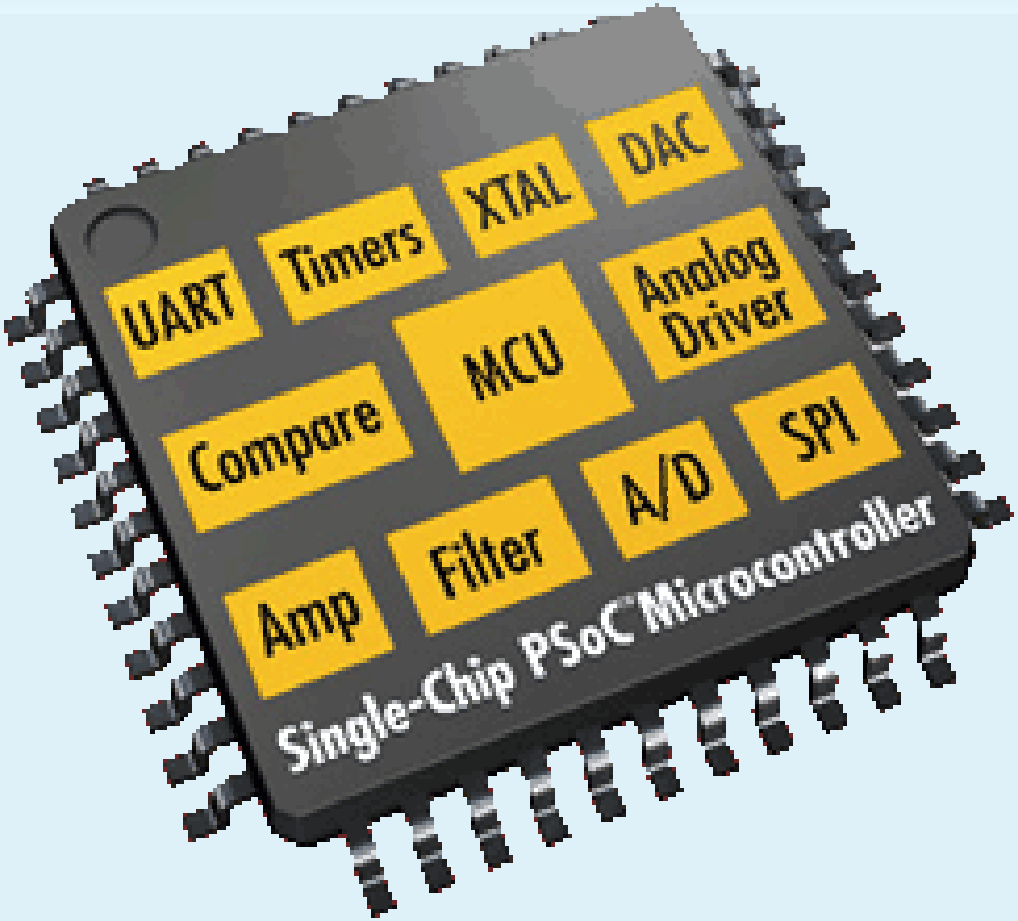
# Run Project

Connect half duplex UART to a serial port through a RS232 line driver.

Connect at 9600 baud, 8 data, no parity and 1 start bit with a terminal program

Single bytes of data transmitted to the PSoC microcontroller will be incremented by one and echoed back to the terminal program

If ‘A’ is sent ‘B’ will be echoed back


UART
Timers
XTAL
DAC
MCU
Analog Driver
Compare
SPI
A/D
Filter
Amp
Single-Chip PSoC™ Microcontroller