PSoC Hardware Control Versus Software Control

Aaron Thompson

November 10, 2011

Design Team 1

Executive Summery:

The Programmable System on a Chip or PSoC is a flexible platform that allows multiple components to be incorporated into a design without the need for external hardware. The virtual components can be configured as hardware enabled, software enabled or a mix of both. For this application note we will detail characteristics of a hardware controller counter and a software controlled time in PSoC Creator. The purpose of a counter is to be able to count to a maximum number and reset to the beginning similar to a clock divider. A timer can be used to measure the time elapsed between two events and trigger an interrupt signal similar to a timeout interrupt. This application note will cover building an 8-bit counter and an 8-bit timer with file and implementation functions that are pre-defined when you drag a virtual component onto the schematic. The similarities for both control methods require hardware connections to clock, input and output port as well initialization and starting function calls on the software side.

Keywords

PSoC, software, hardware, control, timer, counter, interrupt, schematic, C code

Introduction

The Programmable System on a Chip or PSoC is a flexible platform that allows multiple components to be incorporated into a design without the need for external hardware. These virtual hardware components include but are not limited to the use of timers, amplifiers, resistors and capacitors. To add even more flexibility to the PSoC Creator software individual components can be configured on a schematic layout similar to LabView or using C programming language. The description PSoC Creator uses to differentiate between the two configurations is hardware enable and software enabled.

The option between software versus hardware enabled components is set in its options menus. Not all characteristics for a virtual component have the option of being hardware versus software enabled. Using a virtual counter components signals such as reset and the main clock have to be triggered by hardware. For this application note we will detail characteristics of a hardware controller counter and a software controller timer. Each is built using virtual components of PSoC Creator. Below are images of these virtual components, on the Figure 1, for reference, is a counter that is software enabled. Figure 2 is a counter that is exclusively hardware enabled. Figure 3 is a timer that will be used to demonstrate the software controlling in PSoC Creator.

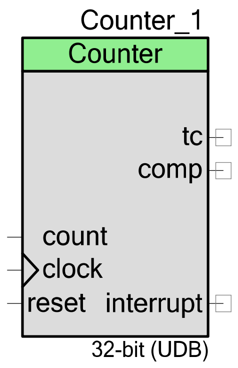


Figure 1

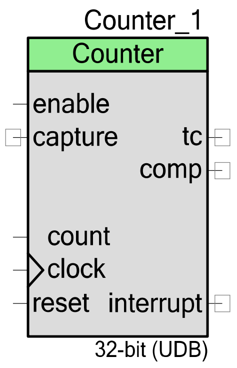


Figure 2

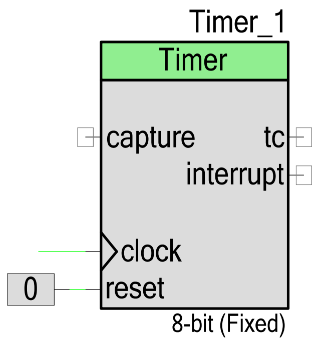


Figure 3

Objective

The purpose of a counter is to be able to count to a maximum number and reset to the beginning similar to a clock divider. A timer can be used to measure the timer elapsed between two events and trigger an interrupt signal similar to a timeout interrupt. With the appropriate circuit either can be implemented but for the purpose of this application note we will focus on how either can be configured for us in the circuit. We will implement the clock divider using hardware control and the timeout interrupt using software control methods.

Walkthrough

Building an 8-bit counter and an 8-bit timer will require the same initial schematic setup we will then divide into the individual application. Dragging a counter, seen in Figure 4, from the Component Catalog on the left side of the PSoC Creator window we will have to connect the reset pin to ‘0’ to ensure the counter continues to run and the clock pin to the BUS_CLK which can also be found in the Component Catalog. The timer will also require these initial steps. For the purpose of customization we will use a UDB Counter, this can be done by double clicking the counter on the schematic and clicking the radio button labeled UDB. At this point we will have to differentiate between the timer and the counter. Another clock signal should be used for the count port on the counter. For a clock divider we should input a clock signal of a lower frequency then the main bus clock, 1KHz will work.

Hardware control setting for the clock divider will require more configurations on the schematic, to do so double click on the counter as before. In the Configure tab change the number in “Compare Value” to “10” and ensure “Clock Mode” is set to “Up Counter”. In the Advanced tab change “Capture Mode” to “Rising Edge” and “Enable Mode” to “Hardware Only”, the other defaults are fine. Back on the schematic you can connect a ‘1’ the enable port to ensure that the counter is always on, a port can also be connected to enable in order to control if the counter is on or off from an outside source. The same clock connected to count can be branched to capture and you can connect an output port or another virtual component to the comp port which will be trigger at ten times slower than the speed of the bus clock. To initialize the clock and have a full functional clock divider in you main.c file you must call Counter_Init(), Counter_ClearFIFO(), Counter_Start(). All the functions are automatically generated in files built when you dragger the Counter onto the schematic.

Figure 4 Schematic

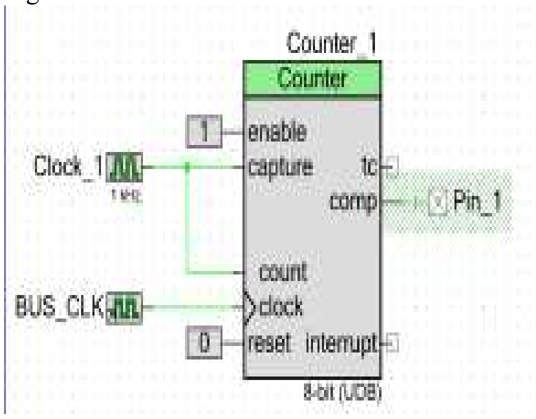
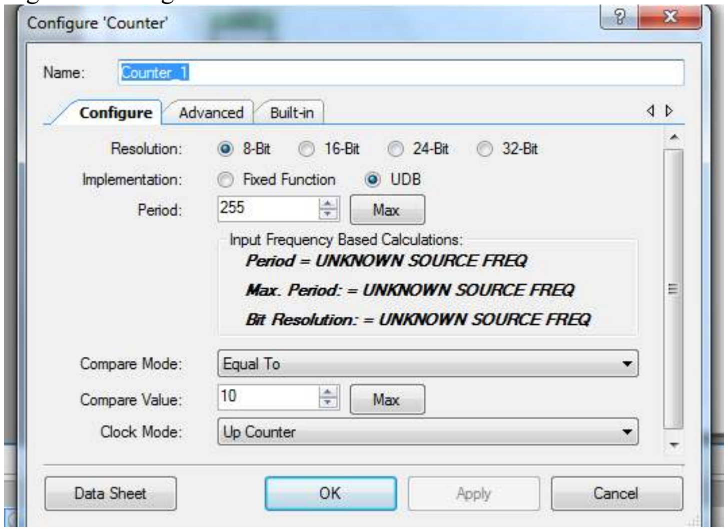


Figure 5 Configure Tab



Configure 'Counter'

Name: Counter_1

Configure | Advanced | Built-in

Resolution: [✓ 8-Bit] 16-Bit 24-Bit 32-Bit

Implementation: Fixed Function [✓ UDB]

Period: 255 Max

Input Frequency Based Calculations:

Period = UNKNOWN SOURCE FREQ

Max. Period: = UNKNOWN SOURCE FREQ

Bit Resolution: = UNKNOWN SOURCE FREQ

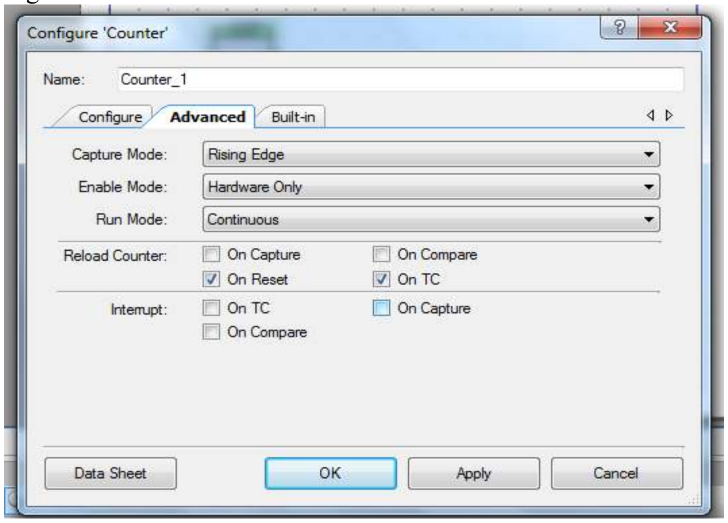
Compare Mode: Equal To

Compare Value: 10 Max

Clock Mode: Up Counter

Data Sheet | OK | Apply | Cancel

Figure 6 Advanced Tab



Configure 'Counter'

Name: Counter_1

Configure | Advanced | Built-in

Capture Mode: Rising Edge

Enable Mode: Hardware Only

Run Mode: Continuous

Reload Counter: ☐ On Capture ☐ On Compare ☑ On Reset ☑ On TC

Interrupt: ☐ On TC ☐ On Capture ☐ On Compare

Data Sheet | OK | Apply | Cancel

Figure 7 main.c
```
void main()
{
   Counter_1_Init();
   Counter_1_ClearFIFO();
   Counter_1_Start();
}
```

Software control, by definition, is more code intensive but still requires some hardware configuration. Once you have selected a timer from the Component Catalog connect the reset port to ‘0’, clock to BUS_CLK and an output port or another virtual component to the interrupt pin. These setting will ensure that the timer run uninterrupted. In order to use the hardware control for the timer a 1 KHz clock can be connected to capture. This would provide an error range of +/- 1msec of the specific time we will required for activity to begin. Next double click on the timer to access the configuration menu; under “Implementation” click the radio button labeled “UDB” so we can have full access to the timer controls. Under “Capture Mode” select “Software Controller” and check the box labels “Enable Capture Counter”. Under “Enable Mode” select “Software Only” and under “Interrupts” select the check box labeled “On Capture” and set the value to ‘1’ so that the timer triggers an interrupt on the first capture function call.

PSoC with its simplicity in design creates all files and functions needed to control the timer via software. Not covered in this application note but available for viewing when creating a project are the Timer.c, Timer.h and Timer_PM.c files which contain all function definitions and variables used to control the timer. Characteristics for the timer can be found using its data sheet, if the timer is selected in the Component Catalog a link will be at the bottom of the window. In main.c you have to first set all variable and initialize the timer. Call functions Timer_Init(), Timer_Enable() to initialize and enable the Timer for use. The function Time_SetInterruptMode(Timer_InterruptOnCaptureCount) must be called to specify that interrupt will occur when the count value is captured. The function Timer_SetCaptureMode(Timer_SoftwareCaptureMode) let the program know to look for a software capture instead of a hardware signal. The next step is to set the max count we want to count up using the function Timer_SetCaptureCount(100000000). This together with the 24 MHz BUS_CLK will ensure that the timer does not run longer than one minute without any activity, more specifically 41.6 seconds. See Figure 7 for code showing how to check if the timeout condition has been satisfied.

Figure 8 Schematic

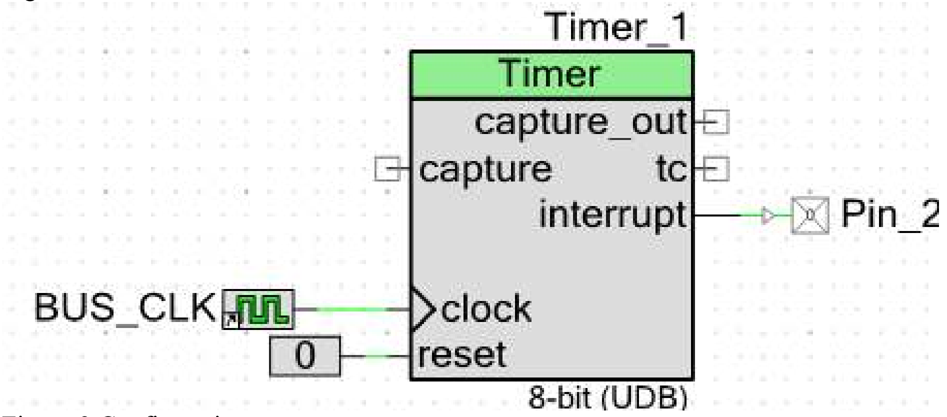
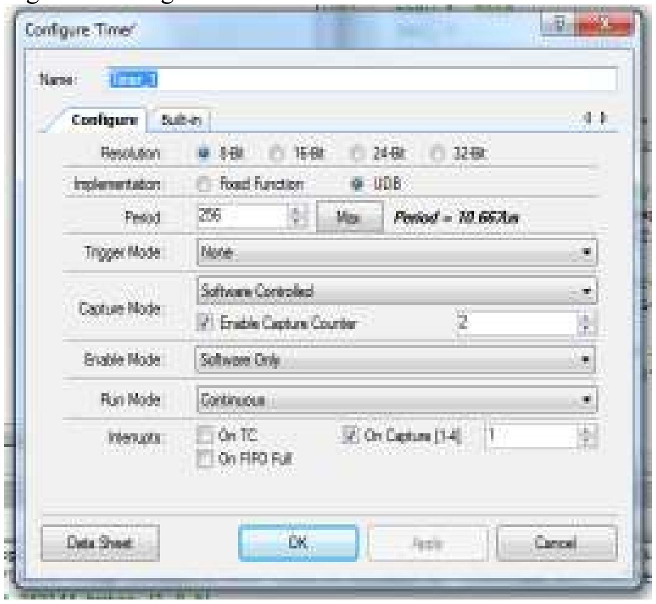


Figure 9 Configuration



Name: Timer_1

Resolution: [✓ 8-Bit] 16-Bit 24-Bit 32-Bit

Implementation: Fixed Function [✓ UDB]

Period: 256 Max *Period = 10.667us*

Trigger Mode: None

Capture Mode: Software Controlled

[✓] Enable Capture Counter 2

Enable Mode: Software Only

Run Mode: Continuous

Interrupts: [ ] On TC [✓] On Capture [1-4] 1

[ ] On FIFO Full

Figure 10 main.c

```
void main()
{
        Timer_1_Init();
        Timer_1_Enable();
        Timer_1_SetInterruptMode(Timer_1_InterruptOnCaptureCount);
        Timer_1_SetCaptureMode(Timer_1_SoftwareCaptureMode);
        Timer_1_SetCaptureCount(100000000);
        Timer_1_ClearFIFO(); //Clear memory
        Timer_1_Start(); //Start timer
        Timer_1_SetInterruptCount(0); //Interrupt when a capture occurs
        While(Timer_1_ReadCounter() !=  Timer_1_ReadCaptureCount()){
        //IF some condition is met reset counter
        // Timer_WriteCounter(0)
        }
        Timer_1_ReadCapture(); //Capture count
}
```

Depending on what ports or other virtual components are used to trigger the reset signal the interrupt will trigger after it has run for 42 seconds without any activity. It is left to the user to include error checking and handling of the interrupt.

Results

A port signal can be substituted in to the main.c's code if statement for the timer and we will have a fully functional interrupt timer that will run for 42 seconds without any activity. In the final design the trigger will have to be called every 42 second or the program will enter a different state or terminate. The counter can be used as a trigger that will keep the timer running because the counter will be able to trigger the timer every 416 nsec. Increasing the count in the counter to 1 million will adjust the count trigger to every 41.6 msec.

Summary

Hardware versus software control only refers to how each virtual component characteristics are set, including period and max count. The similarities for both control methods require hardware connections to clock input and output port as well as initialization and stating function calls on the software side. PSoC Creator makes it easy to implement each method, leaving the user to decide where they want to focus their energy most, on the design of a good circuit.