January 2010

FAQs on Designs with CY8C21x45/22xxx PSoC® 1 Devices

I. WATERPROOF DESIGN

1) How to design shield electrode in PCB layout?

Shield electrode can be simply designed as latticed ground plane near the sensors at the same side. Please refer to AN2292 to find guidance about CapSense layout design.

- Capacitance Sensing - Layout Guidelines for PSoC CapSense - AN2292
- Capacitance Sensing - Layout Guidelines for PSoC CapSense - AN2292 (ZH)

2) How to do waterproof design?

Basic understanding about Cypress waterproof design can be found in the user module datasheets for CSD2X and CSD, and please see application note AN2398 and AN2292 provides more details.

- User Module Datasheet: Dual CapSense® Sigma-Delta Data Sheet, CSD2x (CY8C21x45, CY8C22x45)
- User Module Datasheet: CapSense® Sigma-Delta Data Sheet, CSD (CY8C21x45, CY8C22x45)
- Capacitance Sensing - Waterproof Capacitance Sensing - AN2398
- Capacitance Sensing - Waterproof Capacitance Sensing - AN2398 (ZH)

II. SLIDER DESIGN

1) When adopting the dual-channels CapSense scanning CSD2X UM, how to assign 2 sliders on 1 board during PCB layout?

There are 2 thumb rules:

1) Assign sensors of the same slider to 1 channel as possible as you can
2) Balance the sensor counts of 2 channels to equal or nearly equal to reduce total scanning time as much as possible

2) Why should we keep sensors of 1 slider in the same channel when choosing 2 channels CSD2X?

To ensure that sensors of the same slider share the common- mode errors, so that there will be no errors for calling the wGetCentroidPos routine.

3) Can I assign sensors of the same slider to different channels?

Yes, however, because each channel has its own scanning circuit, this assignment may bring differential-mode error in raw count values, which may get imprecise central position based on the wGetCentroidPos calculation.

III. CSD2X Scanning Time

1) What are the time durations of scanning 1 pair of sensors in 2 channels and the total scanning duration for an asymmetry sensor assignment in 2 channels?

If the resolution and scanning speed settings are different for these two sensors, the scanning time depends on the sensor which takes the longer scanning time. The CSD2X hardware circuit allows each pair of sensors in 2 channels to be charged and sampled in parallel too; however, the firmware need to be modified to support the serial reading of sampled values.

For example, if the sensor in left channel is set up to be 12 resolution and ultrafast scanning speed, while the one in right channel is set up to be 13 resolution and normal scanning speed, the scanning duration

equals to scanning time of sensor on the right channel plus a small delta that is consumed by reading the additional 2 sampled values from registers (usually ignored).

If the sensor counts of 2 channels are asymmetry, such as 9 sensors in left and 6 sensors in right, supposed all have the same setting and scanning 1 sensor consumes T millisecond, then the total scanning duration should be about 9 * T.

2) How to reduce the CSD2X scanning time duration for applications that are very responding time sensitive?

One solution is to manually scan the sensors with ScanSensor() routine instead of automatically scanning all sensors with ScanAllSensors(). For example, ten sensors scanning can be simply divided into 5 groups of 2-sensor scanning, so that 1 group of sensors is scanned in the main routine during the each call. This allows the main routine to be able to handle other time sensitive jobs, and the system can process the key trigger event until the fifth group is scanned, so that an event that more than 2 buttons are pressed can be detected.

If the above method cannot meet the application requirements, designer can also try to increase the sensitivity of sensors, such as use larger area of sensor, higher reference to get smaller resolution and faster scanning speed setting. These two parameters are the critical factors to impact the scanning duration.

IV. RAM ACCESS

1) How to access data in specified address?

Below codes is an example to access data in specified address of RAM page 2 written in C.

```
…
BYTE t1 = 0;
BYTE t2 = 0;
t1 = *(BYTE *)(0x201);
*(BYTE *)(0x201) = 100;
t2 = *(BYTE *)(0x201);
…
```

Note that M8C uses the real address to access RAM and there is no protection to avoid conflicting valuable data with absolute address, so the programmer is obligated to ensure that the selected address is safe in the operation. Moreover, if the address exceeds the RAM actual size, the current version of PSoC Designer cannot check this error, which means that programmer should keep in mind that the address does not exceed the boundary of RAM size.

If programmer wants to use the assembly language to achieve the same goal, he/she should pay additional attention to the RAM page switching in assemble language. Please refer to AN2218 for more details.

Design Aids - Large Memory Model Programming for PSoC - AN2218

2) Why cannot I access RAM page 3?

Actually there is no limit of programming syntax for accessing data in RAM page 3. This constrain is due to the stack usage in CY8C22xxx (Neon) parts. CY8C22xxx (Neon) has 1K RAM which is divided into 4 pages. The page 3 is used for system stack. Any incorrect manipulation on this RAM space might cause system crash. Therefore it is highly recommended to leave this page alone.

V. EMI/EMC/ESD/ETF

1) What is EMI/EMC/ESD/ETF performance of CY8C21x45/22xxx Neon devices?

Please refer to CY8C21x45/22x45 datasheet for chip performance. Moreover, these test results highly depend on overall system design, thus the applications specific data is recommended before performing the tests.

2) How can I set up the parameters to optimize the EMC behavior (e.g. 96MHz) (IMO , Prescaler , Ref , IDAC trends => greater or lower...)

Please see TL cases 29ZI7VX and 3DGU6D that includes a lot of good information on various EMC issues:

Lower IMO = better RE.
Lower Prescaler frequency = better RE.
Lower Ref voltage = better RE
Lower IDAC values = better RE.

Note that the most important EMC aspect is the layout design, however, sometimes the pre-charge source selection can have an impact on RE too. The pick of timer can cause RE at known frequencies with higher peaks and the choice of pre-charge source can also cause RE at all frequencies, but lower.

Generally, improving RE will cause your EMI immunity to get worse. For example, lowering the reference voltage will get you lower RE, but is will also reduces the level of your CS signals, which will in turn make the design more susceptible to EM noise.

VI. CSD2X Setting

1) Do you have a complete description of the Auto-calibration algorithm?

The auto-calibration function modifies IDAC (and maybe reference settings) until the raw counts measured are close to 50% of the fill scale counts. So, if you are using 12 bit resolution, all of the sensor raw counts should be ~2048 after calling the Calibrate API.

2) Is the function raw_value (IDAC) linear or there exists local maxima/minima?

The IDAC values in the table do correspond linearly with the IDAC setting (at least if the IDAC were 100% linear itself).

3) Is the function raw_value (PreScaler) strong linear or there exists local maxima/minima?

These are not linear. The only prescaler settings available are SysClk/1, /2, /4, /8, /16, /32, /64, /128, /256. So, these values are not linear.

4) What reference source has lower noise crosstalk from the power supply?

The Vbg reference source contains much lower power supply noise than the Vdd reference source (since Vdd is the power supply)

5) What CSD2X configuration (IDAC, Rb , 1x, 2x) has lower noise crosstalk from the power supply?

There should be no difference in power supply crosstalk between the 1x and 2x configurations. The Rb configurations probably have lower power supply crosstalk, since a fixed, very stable resistor is used instead of the IDACs, which probably have some power supply noise. However, the IDAC configurations are generally better, since it give much more flexibility.

VII. Miscellaneous

1) Why does I2C User Module always timeout?

There are 2 possible causes for this problem:
a) There is a frequently interrupt source to interrupt I2C ISR routine (interrupt service routine), which makes SCL at low level and the master waits for the slave until timeout.
b) The PSoC Designer version is too low. Please get the latest version from www.cypress.com/psocdesigner.

2) How to apply watchdog reset?

There are 2 steps for watchdog setup:

a) Enable Watchdog setting in ***Global Resource*** tab;
b) Clear watchdog timer with ***M8C_ClearWDT*** before watchdog interrupt generated;

3) Why do the multiple resets happen when power supply drops down?

The reason for multiple reset is that the POR bits in the VLT_CR register are reset by PPOR. When boot.asm runs (e.g. Vdd = 5, CPU = 24MHz) the POR will be set to 4.75V.If Vdd drops below 4.75, then the POR occurs and the VLT_CR register is cleared, setting POR to 2.4V. Now the part is no longer reset and it starts running boot.asm until it reaches the code that sets POR = 4.75V, where it resets again. It will run this loop repeatedly (on power up and power down) as long as Vdd is between 4.75V and 2.4V.

There are several workarounds:

- Set any pins that are causing problems to HiZ and set them to the final drive mode in main.c. The loop of reseting happens before the program gets to main.c.
- Code could be added in boot.asm to set the LVD to the final voltage range and then poll the status of the LVD in the VLT_CMP register. Then wait in a loop until Vdd is high enough and then continue on.
- The set of code listed could be moved to be before the LoadConfig calls. Then the POR loop would happen before the PSoC configuration is loaded.

Here is an example solution. Insert the red code before LoadConfigInit is called in boot.tpl that is "generated" by PSoC5.0. This code will check if power supply can meet the system requirement, otherwise it will halt system to avoid any unexpected code execution.

```
M8C_SetBank1
CheckLVD:
   mov A, reg[VLT_CMP]
   and A, 02h
   jnz CheckLVD

   ;---------------------------
   ; Load Base Configuration
   ;---------------------------
   ; Load global parameter settings and load the user modules in the
   ; base configuration. Exceptions: (1) Leave CPU Speed fast as possible
   ; to minimize start up time; (2) We may still need to play with the
   ; Sleep Timer.
   ;
       lcall LoadConfigInit
```

4) How to apply the external crystal oscillator?

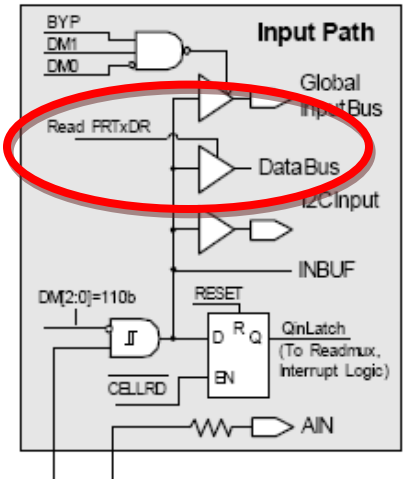
First, set the drive mode of pins of external crystal oscillator to be Hiz, and then use following codes to switch to external crystal oscillator dynamically.

```
void CPU_Switch32KHzToECO(void)
{
      unsigned char bSleepIntervalBackup;
      /* Backup OSC_CR0 register */
      bSleepIntervalBackup = OSC_CR0 & OSC_CR0_SLEEP;
      /* Set sleep timer interval to 1 second */
      OSC_CR0 |= OSC_CR0_SLEEP_1Hz;
      /* Reset the sleep timer to get a full second */
      M8C_ClearWDTAndSleep;
      /* Make sure that '32K_Select' equal to 'external' */
      CPU_SCR1 |= CPU_SCR1_ECO_ALLOWED;
      /* Select ECO */
      OSC_CR0 |= OSC_CR0_32K_SELECT;
      /* Test the SleepTimer Interrupt Status */
      while(!(INT_CLR0 & INT_MSK0_SLEEP));
      /* Restore OSC_CR0 register */
   OSC_CR0 &= ~OSC_CR0_SLEEP;
   OSC_CR0 |= bSleepIntervalBackup;
}
```

5) How to operate GPIO Data register?

When reading PRTxDR register, it just reads the electrical level on port wires rather than an internal register (see red marker in below figure). This has a potential risk when a signal is also changed by other external component. For example, PSoC sets PRT1DR[4] high level but then another MCU pulls this wire down, PSoC expects to get a high level by reading the port register, however a low level is actually read and a data error happens. To avoid this error, a shadow register is suggested to store port data register and each operation on port data register is divided into 2 steps:

1) Update/Check shadow register
2) Update data register with shadow register



6) How to apply software filter to CSD2X data?

Here is a FIR filter example to illustrate how to apply software filter to CSD2X raw count value.

```
// declare a buffer to store history input
#define          SENSOR_CNT          CSD2X_TotalSensorCount
```

```
WORD FIRBuf[SENSOR_CNT][3];
……
void main()
{
   BYTE i = 0;
      ……
   // initialize buffer
   for(i=0; i<SENSOR_CNT; i++)
   {
      FIRBuf[i][0] = FIRBuf[i][1] = FIRBuf[i][2] = CSD2X_waSnsResult[i];
   }
   ……
   while(1)
   {
      CSD2X_ScanAllSensors();
      for(i=0; i<SENSOR_CNT; i++)
      {
         WORD temp;
         temp = FIRBuf[i][0] >> 2 + FIRBuf[i][1] >> 2 +
                FIRBuf[i][2] >> 2 + CSD2X_waSnsResult[i] >> 2;
         FIRBuf[i][0] = FIRBuf[i][1];
         FIRBuf[i][1] = FIRBuf[i][2];
         FIRBuf[i][2] = CSD2X_waSnsResult[i];
         CSD2X_waSnsResult[i] = temp;
      }
      CSD2X_UpdateAllBaselines();
      if(CSD2X_bIsAnySensorActive())
      {
                  ……
      }
      ……
   }
}
……
```