

Microcontrôleurs

Travaux Pratiques – HAE404E – Licence 2 EEA / 2022

Mikhaël MYARA
mikhael.myara@umontpellier.fr



“La plupart des problèmes d'informatique proviennent de l'interface chaise-clavier.”

Klaus Klages

Microcontrôleurs - Travaux Pratiques - HAE404E - Licence 2 EEA/2022

26 avril 2022

Programmation des Microcontrôleurs en C - Application au STM32.
par Mikhaël Myara, pour le Department EEA, Université de Montpellier, France



This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

Table des matières

Séance I – Environnement de travail • Utilisation des GPIO	5
I.1 L'environnement de travail	5
I.2 Utilisation des GPIOs	7
I.3 Avec la bibliothèque "ST standard library"	7
I.4 Cabler une LED supplémentaire	8
Séance II – Gestion du temps	9
II.1 Prise en main des Timer	9
II.2 Mini projet : Code morse	10
Séance III – Acquisition de signaux • Debugger	12
III.1 Prise en main de la conversion analogique–numérique	12
III.2 Le Debugger	13
III.3 Mini Projet : rapport cyclique imposé par un potentiomètre	16
Séance IV – Génération de signaux analogiques : PWM	17
IV.1 Prise en main des signaux PWM	17
IV.2 Mini Projet : "LED RGB"	18
Séance V – Interruptions	21
V.1 Prise en main des interruptions externes	21
V.2 Prise en main des interruptions par Timer	22
V.3 Générer un sinus	24
V.4 Un mini synthétiseur	26
V.5 Mini Projet : Filtrage numérique d'un signal	27
Annexe A – Création d'un projet en mode "Bare-Metal"	33
Annexe B – Création d'un projet en mode "ST Standard Library"	38
Annexe C – Cablage	42
C.1 Fonctionnement des plaquettes d'essai	42
C.2 Exemple	42

Annexe D – Circuit "Entrée audio"	44
Annexe E – Filtre passe-bas numérique	47
E.1 Etablir l'équation du filtre	47
E.2 Code C du filtre	48
Annexe F – Documentation de l'amplificateur opérationnel MCP601	49

Séance I – Environnement de travail • Utilisation des GPIO

⌚ Motivations et objectifs

On va prendre en main l'environnement de travail "STM32 C/C++ System Workbench" ainsi que la carte STM32L152C, puis écrire nos premiers programmes utilisant des GPIO.

I.1 L'environnement de travail

On va prendre pour support un code déjà donné dans le cours, qui fait simplement clignoter une LED, en code "bare-metal" :

```
1 int main() {  
2     long* RCC_AHBENR = (long*) (0x40023800 + 0x1C) ;  
3     (*RCC_AHBENR) = (*RCC_AHBENR) | (1<<1); // code definitif  
4  
5     long* GPIOB_MODER = (long*) (0x40020400 + 0x0);  
6     (*GPIOB_MODER) = (*GPIOB_MODER) & (~(0b11<<14));  
7     (*GPIOB_MODER) = (*GPIOB_MODER) | (0b01<<14);  
8  
9     long* GPIOB_ODR = (long*) (0x40020400 + 0x14);  
10    while(1) {  
11        (*GPIOB_ODR) = (*GPIOB_ODR) | (0b1<<7);  
12        for(int k=0; k<100000;k++) { } // perdre du temps pour "attendre"  
13        (*GPIOB_ODR) = (*GPIOB_ODR) & (~(0b1<<7));  
14        for(int k=0; k<100000;k++) { } // perdre du temps pour "attendre"  
15    }  
16    return 0;  
17}
```

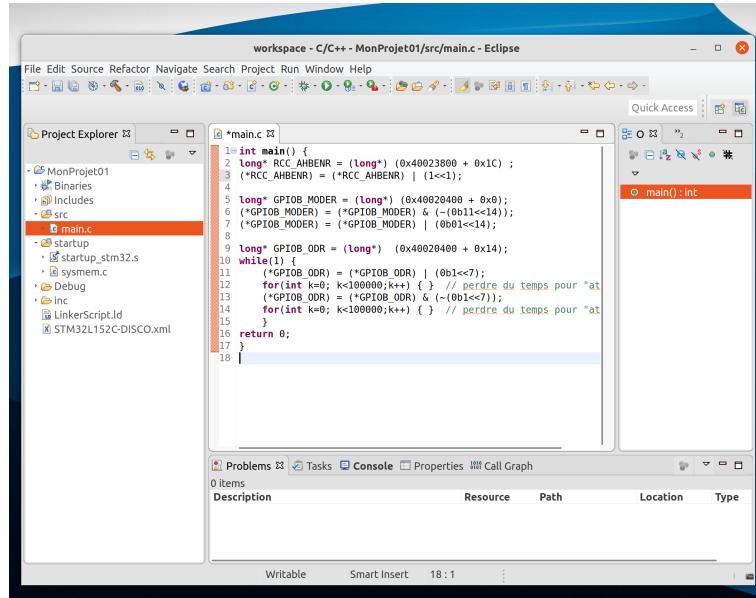
🌐 [codeTP/codeTP-I-1-1.c]

Et on va commencer à l'implanter dans la carte STM32L152C. Voici les étapes :

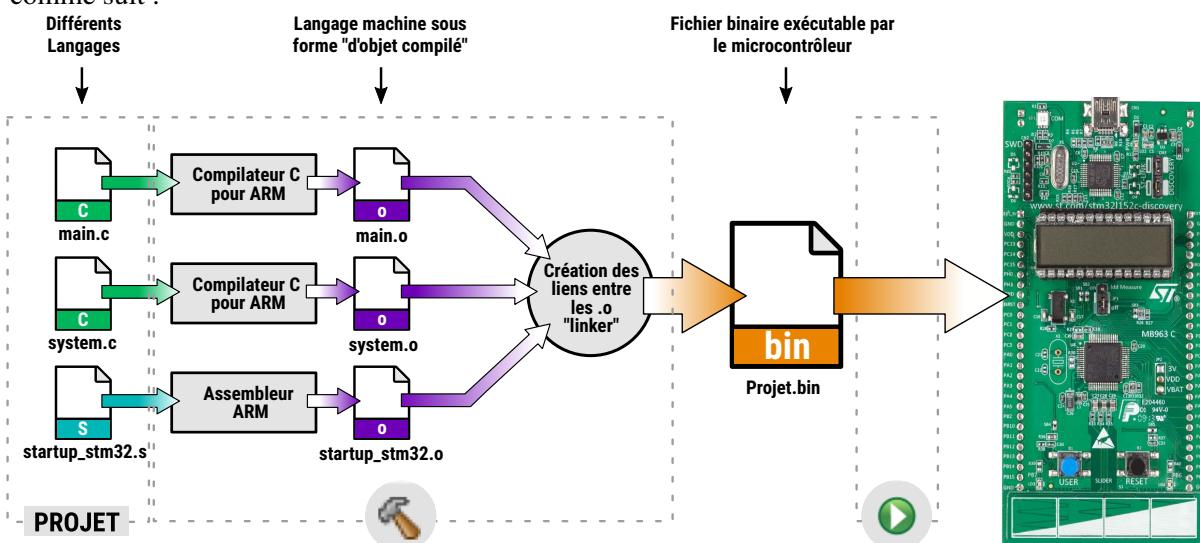
1	2	3
Connecter la carte au PC via le port USB 	Créer un projet en mode "bare-metal", voir Annexe A  Voir Annexe A	Copier/coller le code, compiler et transférer  "Compiler" (Construire = Build)  Transférer à la carte

Compilation et transfert

Quand vous créez un projet pour STM32, il y a un ensemble de fichiers qui sont ajoutés au projet et qui implémentent des besoins fondamentaux. Notamment, il y a la séquence de démarrage de la carte, qui réalise une première configuration. C'est ce que l'on voit une fois le projet créé, dans la liste des fichiers, sur la gauche :



Notamment il y a les fichiers **startup_stm32.s** et **system.c**. Le premier est la séquence de démarrage du STM32, écrite en langage assembleur : son rôle est de donner une première configuration de base au STM32 (configuration de l'horloge par exemple) et de lancer la fonction **main()** du projet. L'autre contient un code élémentaire pour aider la fonction **malloc()** à fonctionner. En effet, il faut avoir conscience que nous ne disposons pas ici de système d'exploitation, et que donc **malloc()** est une fonction écrite "en partant de zéro" (on dit "*from scratch*"). La construction puis le transfert du projet à la carte va se dérouler comme suit :



- ② **Q-1.1.1:** Créez un projet "bare-metal", collez dedans le code qui fait clignoter la LED, transférez le à la carte, puis vérifiez que vous obtenez bien une LED qui clignote.
- ② **Q-1.1.2:** Si vous disposez d'un chargeur de téléphone, déconnectez la carte du PC et branchez la au

chargeur pour alimenter la carte. Qu'observez-vous ? Peut-on dire qu'une fois programmée, la carte est indépendante du PC ?

I.2 Utilisation des GPIOs

Vous avez donc un code de base en mode "bare-metal" pour faire clignoter une LED. A partir de là, vous devriez pouvoir réaliser ce qui est demandé ci-dessous.

- ② **Q-1.2.1 :** Modifiez le code pour faire clignoter l'autre LED. Pour cela, reprenez les documentations associées au microcontrôleur, comme montré pendant le cours.
- ② **Q-1.2.2 :** Modifiez le code pour sélectionner la LED qui clignote lorsque le bouton est appuyé.

I.3 Avec la bibliothèque "ST standard library"

On a vu en cours que l'approche "bare-metal", si elle est tout à fait abordable pour quelque chose de simple, demande une connaissance trop intime du microcontrôleur dès que l'on veut faire des choses un peu évoluées. Alors on passe par l'utilisation de bibliothèques. Sur STM32, il y a la "*ST Standard Library*" qui gère la plupart des sous-ensembles du STM32.

On donne le même code que précédemment, mais réécrit pour la bibliothèque "*ST Standard Library*" :

```
1 #include "stm32l1xx.h"
2
3 int main() {
4     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);
5
6     GPIO_InitTypeDef leds_PB;
7     GPIO_StructInit(&leds_PB);
8     leds_PB.GPIO_Mode = GPIO_Mode_OUT;
9     leds_PB.GPIO_Pin = GPIO_Pin_7;
10    GPIO_Init(GPIOB, &leds_PB);
11
12    while(1) {
13        GPIO_SetBits(GPIOB, GPIO_Pin_7);
14        for(int k=0; k<100000; k++){}
15        GPIO_ResetBits(GPIOB, GPIO_Pin_7);
16        for(int k=0; k<100000; k++){}
17    }
18    return 0;
19 }
```

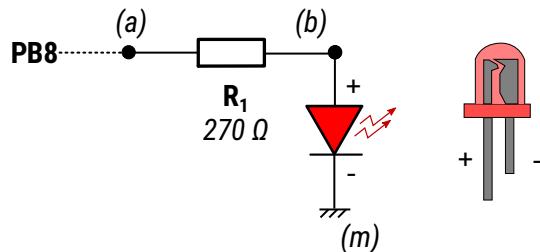
 [codeTP/codeTP-I-3-2.c]

- ② **Q-1.3.1 :** En suivant l'Annexe B, créez un projet de type "*ST Standard Library*", copiez le code donné ci-dessus à l'intérieur, puis exécutez le pour vérifier son fonctionnement.
- ② **Q-1.3.2 :** Regardez dans la documentation (le fichier CHM) la documentation du module GPIO, et vérifiez que vous comprenez ce que fait chaque ligne de code.
- ② **Q-1.3.3 :** Comme précédemment, modifiez le code pour faire clignoter l'autre LED.
- ② **Q-1.3.4 :** Comme précédemment, modifiez le code pour changer la LED qui clignote lorsque le bouton est appuyé.

I.4 Cabler une LED supplémentaire

On a vu en cours qu'on pouvait utiliser la pin PB8 pour ajouter une LED, que l'on a choisi rouge. On a calculé aussi qu'il fallait une résistance de protection d'environ 250Ω .

- ② **Q-1.4.1:** Compte tenu des valeurs qui existent, on va choisir $R = 270\Omega$. La LED éclairera un peu moins (mais juste un peu), ce n'est pas très grave. Cablez cette résistance avec la LED comme montré ci-dessous :



Important

Remarque 1 : Les microcontrôleurs qui vous sont fournis ont été reliés à des plaques d'essai. Si vous n'avez jamais utilisé de plaquette d'essai ou si vous avez oublié, le fonctionnement d'une plaque d'essai est décrit en *Annexe C*.

Remarque 2 : Lors du câblage de la LED, vous ferez attention au fait que la LED est un composant **polarisé**, comme représenté sur le schéma : la "cathode" (patte -) de la LED est celle qui est du côté où le plastique transparent présente un morceau de surface "platte". Vous pouvez vous en rendre compte soit en touchant avec le doigt les deux côtés, soit en regardant attentivement la LED comme présenté sur l'image ci-dessus. Interrogez votre enseignant si vous n'êtes pas sûr du câblage. ■

- ② **Q-1.4.2:** Ecrivez un code pour tester cette LED seule.
- ② **Q-1.4.3:** Ecrivez un code qui permet de sélectionner, avec le bouton, laquelle des 3 LED clignote : initialement c'est la verte qui doit clignoter, après un appui la bleue, après encore un appui la rouge, et le prochain appui sélectionnera à nouveau la verte, et le cycle recommencera.

Séance II – Gestion du temps

⌚ Motivations et objectifs

On va jouer avec traiter deux sujets : les Timers du microcontrôleur et nous allons aussi utiliser un outil puissant pour analyser les programmes et en comprendre les erreurs : le Debugger.

II.1 Prise en main des Timer

On redonne le code de base vu en cours pour la gestion des Timer :

```
1 #include "stm32l1xx.h"
2
3 // fonctions pour configurer et utiliser le timer 5.
4 // voir apres le main
5 void TIM5_Config();
6 void TIM5_delay_ms(int delay);
7
8 int main(void)
9 {
10     TIM5_Config();
11
12     /* Activer GPIOB sur AHB */
13     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE);
14     /* Configurer PB7 */
15     GPIO_InitTypeDef gpio_b;
16     GPIO_StructInit(&gpio_b);
17     gpio_b.GPIO_Mode = GPIO_Mode_OUT;
18     gpio_b.GPIO_Pin = GPIO_Pin_7;
19     GPIO_Init(GPIOB,&gpio_b);
20
21     while(1) {
22         GPIO_SetBits(GPIOB,GPIO_Pin_7);
23         TIM5_delay_ms(500);
24         GPIO_ResetBits(GPIOB,GPIO_Pin_7);
25         TIM5_delay_ms(500);
26     }
27 }
28
29 // ### TIM5 pour delai
30 // Configuration
31 void TIM5_Config()
32 {
33     /* Activer TIM5 sur APB1 */
34     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5,ENABLE);
35     /* Configurer TIM5 : prescaler a 1 ms, periode au maximum*/
36     TIM_TimeBaseInitTypeDef DelayTimer;
37     TIM_TimeBaseStructInit(&DelayTimer);
```

```

38     DelayTimer.TIM_Prescaler = 16000-1;
39     DelayTimer.TIM_Period = 0xFFFFFFFFU;
40     TIM_TimeBaseInit(TIM5,&DelayTimer);
41 }
42
43 // Fonction Delai en millisecondes
44 void TIM5_delay_ms(int delay)
45 {
46     TIM_SetCounter(TIM5,0);
47     TIM_Cmd(TIM5, ENABLE);
48     while(TIM_GetCounter(TIM5)<delay)
49     {
50     }
51     TIM_Cmd(TIM5, DISABLE);
52 }
```

 [codeTP/codeTP-II-1-3.c]

- ② **Q-2.1.1:** Prenez en main ce code : créez un projet adapté et exécutez le.
- ② **Q-2.1.2:** Prenez ma documentation (fichier CHM) et vérifiez que vous comprenez le paramétrage qui est fait.
- ② **Q-2.1.3:** Dessinez grossièrement sur papier le signal que l'on est supposé observer sur PB7. Vérifiez avec un oscilloscope que cela correspond et que les constantes de temps données sont précises.

II.2 Mini projet : Code Morse

On donne le début de code ci-dessous qui définit le code Morse :

```

1 typedef struct
2 {
3     char lettre;
4     char morse[10];
5 }code;
6
7
8 int main()
9 {
10 //int NB_CODES = 26;
11 code alphabet[26]=
12     { {'A', ". _"}, {"B", " _ . ."}, {"C", " _ . _"}, {"D", " _ _ ."}, {"E", ". _ . . ."}, {"F", " _ . _ ."}, {"G", " _ _ _"}, {"H", " _ . . ."}, {"I", " _ _"}, {"J", ". _ _ ."}, {"K", " _ _ . _"}, {"L", " _ . _ ."}, {"M", " _ _"}, {"N", " _ _ ."}, {"O", " _ _ _"}, {"P", " . _ _ ."}, {"Q", " _ _ . _ ."}, {"R", " _ . _ ."}, {"S", " _ . . ."}, {"T", " _ _ _"}, {"U", " _ . _ . ."}, {"V", " _ . . _ ."}, {"W", " _ _ . ."}, {"X", " _ . _ _ ."}, {"Y", " _ _ . _ ."}, {"Z", " _ _ _ ."} };
21
22
23
24 }
```

 [codeTP/codeTP-II-2-4.c]

Deux remarques :

- Dans la mesure où les lettres sont dans l'ordre alphabétique, le champ **lettre** de la structure ne sert fonctionnellement à peu près à rien. Il participe surtout à la lisibilité du code et au débogage si besoin,

- Si on suppose que le code ASCII de la lettre dont on veut connaître le codage Morse s'appelle **maLettre**, il suffit de taper :

```
alphabet[maLettre - 'A'].morse
```

En effet, cette expression retranche à "**maLettre**" le code ASCII du 'A', ce qui revient à obtenir une valeur entre 0 et 26, soit l'indice de la bonne case dans le tableau **alphabet**.

- ② **Q-2.2.1:** Donnez vous, dans le main, une chaîne de caractères qui contient un mot de quelques lettres
- ② **Q-2.2.2:** Ecrivez un code qui permet de faire clignoter la LED pour générer le code Morse de la chaîne de caractères donnée. On donne les constantes de temps :

- un "point" dure 0.1 s
- un "trait" dure 0.3 s
- il faut attendre 0.1 s entre 2 éléments morse (point ou trait)
- il faut attendre 0.3 s entre 2 lettres
- il faut attendre 1 s entre 2 mots

Remarque : Vous avez intérêt à découper le code en deux fonctions :

- une qui envoie une sur la LED une seule lettre en code morse, dont le prototype serait par exemple :

```
void morseUneLettre(char lettre, code* alphabet);
```

- une autre qui envoie la totalité d'une chaîne en code morse sur la LED, en utilisant **morseUneLettre()** décrite ci-dessus, et dont le prototype serait par exemple :

```
void morseChaine(char* s, code* alphabet);
```

- ② **Q-2.2.3:** Envoyez des SOS en boucle, avec une attente de 1 seconde entre deux SOS.
- ② **Q-2.2.4:** Vérifiez à l'oscilloscope que cela fonctionne. Divisez tous les temps par 10 et vérifiez à nouveau sur l'oscilloscope.

Remarque : Vous venez de programmer votre premier protocole de communication, en utilisant un codage normalisé. Limité au texte en effet, mais il faut un début à tout. Renseignez vous sur le "lifi", inspiré du "wifi", et faites l'analogie.

- ② **Q-2.2.5:** Regardez à l'oscilloscope le code morse associé à votre prénom ou nom ou ce que vous voudrez.

Séance III – Acquisition de signaux • Debugger

⌚ Motivations et objectifs

On va commencer par regarder l'acquisition de signaux analogiques avec le microcontrôleur. Le problème que l'on va rencontrer, c'est que lorsque l'on fait l'acquisition de signaux analogiques, il est difficile de savoir si "ça marche", dans la mesure où nous n'avons jusqu'ici aucun moyen d'en connaître la valeur : il n'y a pas de "printf" sur un microcontrôleur. On travaille donc un peu "en aveugle".

Cela va suggérer l'utilisation d'un outil puissant qui fait partie du cycle de développement sur microcontrôleur : le debugger.

III.1 Prise en main de la conversion analogique-numérique

On redonne le code vu en cours, un peu modifié pour lire 5 fois l'ADC au lieu d'une (lines 11 à 16), ce sera pratique pour la suite.

```
1 #include "stm32l1xx.h"
2
3 void ADC1_Config();
4 uint16_t ADC1_Get(uint8_t ch);
5
6 int main(void)
7 {
8     ADC1_Config();
9     for(;;)
10    {
11        volatile uint16_t adcVal = 0;
12        adcVal = ADC1_Get(ADC_Channel_5);
13        adcVal = ADC1_Get(ADC_Channel_5);
14        adcVal = ADC1_Get(ADC_Channel_5);
15        adcVal = ADC1_Get(ADC_Channel_5);
16        adcVal = ADC1_Get(ADC_Channel_5);
17
18        // ici adcVal vaut la valeur lue sur PA5
19    }
20}
21
22 // ### ADC1 sur PA5
23 // Configuration
24 void ADC1_Config()
25 {
26     /* Activer GPIOA sur AHB */
27     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
28     /* Parametrer PA5 en mode analogique */
29     GPIO_InitTypeDef gpio_a;
30     gpio_a.GPIO_Pin = GPIO_Pin_5;
31     gpio_a.GPIO_Mode = GPIO_Mode_AN;
32     gpio_a.GPIO_PuPd = GPIO_PuPd_NOPULL;
```

```

32     GPIO_Init(GPIOA, &gpio_a);
33
34     /* Activer ADC1 sur APB2 */
35     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
36     /* Configurer ADC1 en 12 bit, simple acquisition */
37     ADC_InitTypeDef adc_1;
38     ADC_StructInit(&adc_1);
39     adc_1.ADC_NbrOfConversion = 1;
40     adc_1.ADC_Resolution = ADC_Resolution_12b;
41     ADC_Init(ADC1, &adc_1);
42     ADC_Cmd(ADC1, ENABLE);
43 }
44
45 // Acquisition d'une valeur
46 // PA5 : ch = ADC_Channel_5
47 uint16_t ADC1_Get(uint8_t ch)
48 {
49     ADC-RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_4Cycles);
50     ADC_SoftwareStartConv(ADC1);
51     while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET)
52     ;
53
54     return ADC_GetConversionValue(ADC1);
55 }
```

 [codeTP/codeTP-III-1-5.c]

- A. Implantez le dans la carte à microcontrôleur, testez le et constatez qu'en apparence "il ne fait rien".
- B. Regardez le code lui même et consultez la documentation (fichier CHM et architecture), et vérifiez que vous comprenez bien ce qui se passe.
- C. Constatez que c'est compliqué de dire si il fait vraiment ce que l'on croit ...

III.2 Le Debugger

On est face à une situation typique : on a écrit un code, on peut avoir confiance dans l'idée qu'il marche mais c'est de l'auto-persuasion. A part le fait que "*c'est le prof qui l'a fait alors ça marche sûrement*", ce qui n'est pas vraiment un argument factuel. Alors on aimeraient bien se rassurer sur le fait que "ça marche" avant d'aller plus loin.

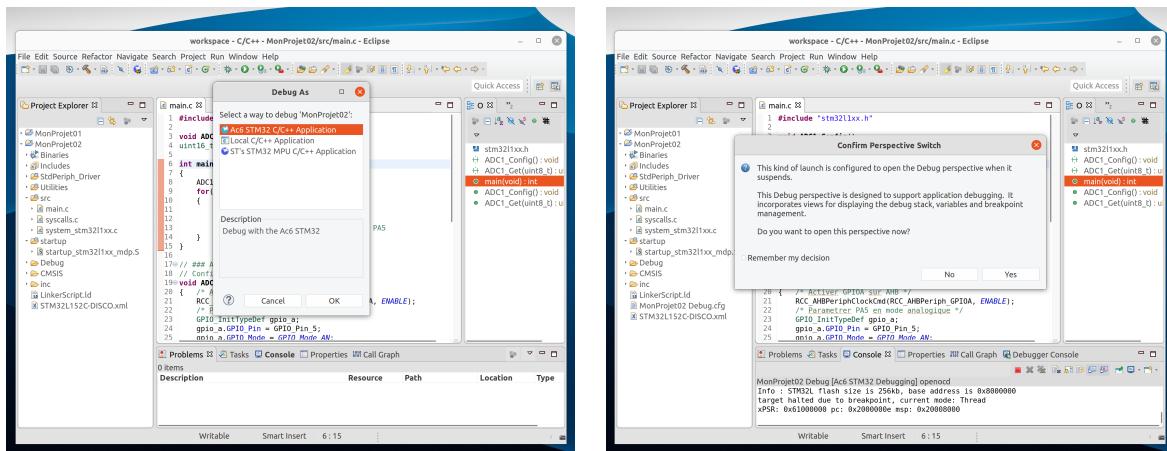
Pour cela on va utiliser un outil que l'on n'a jamais utilisé jusqu'ici : le **debugger**. C'est un outil qui n'est pas dédié aux microcontrôleurs et qui existe en développement sur PC. Mais sur PC on peut souvent s'en passer parce qu'on a au moins accès à **printf** pour "y voir". Ici on a juste une LED qui peut clignoter, c'est pas terrible ...

Commençons par lancer le debugger. Une fois que vous avez implanté le code dans le microcontrôleur, cliquez sur l'icone :

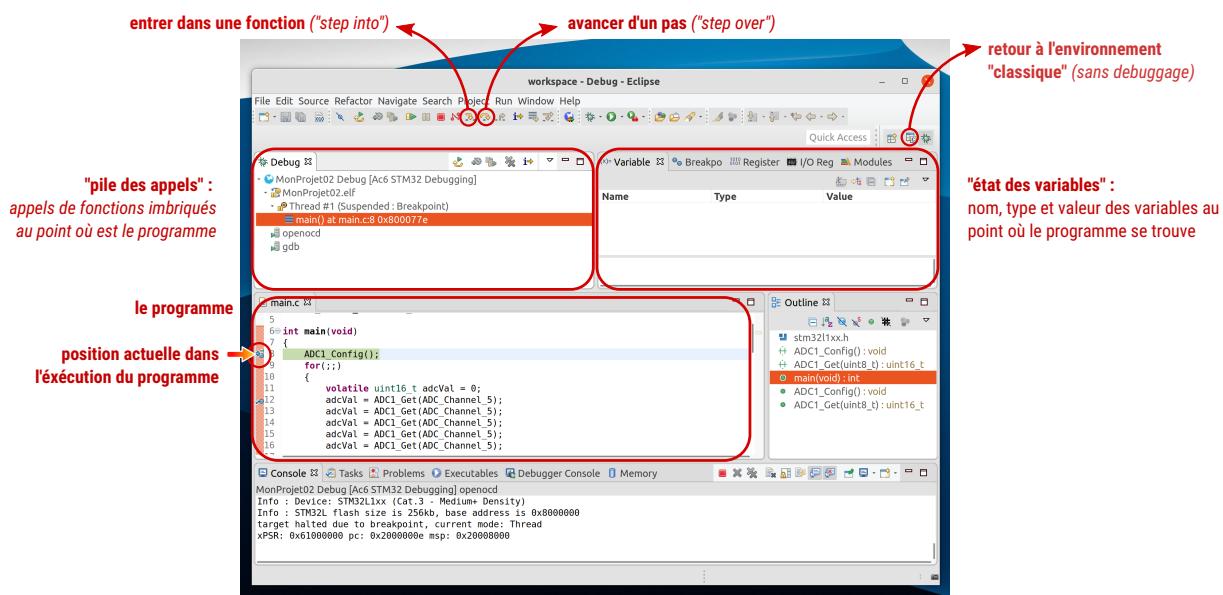


Debugger

L'environnement va vous demander deux confirmations que l'on a représenté ci-dessous. Validez les deux :



L'environnement va ensuite recompiler le code pour qu'il soit approprié pour le debugger, et l'environnement de travail prendra une nouvelle forme :



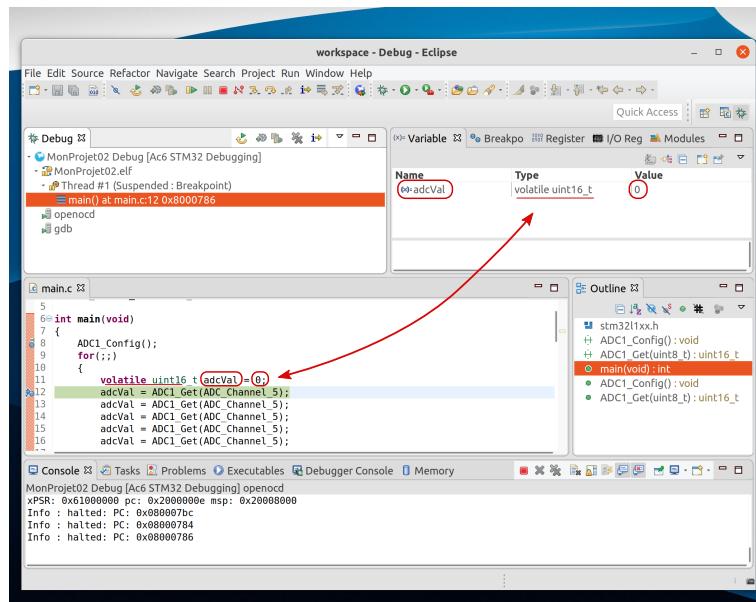
Au lancement, le debuggeur s'est placé au début de la fonction main, ligne 8, puisque c'est le début du programme. A ce stade, il n'a pas encore exécuté cette ligne.

Si on appuie sur le bouton "avancer d'un pas" (voir copie d'écran ci-dessus) :

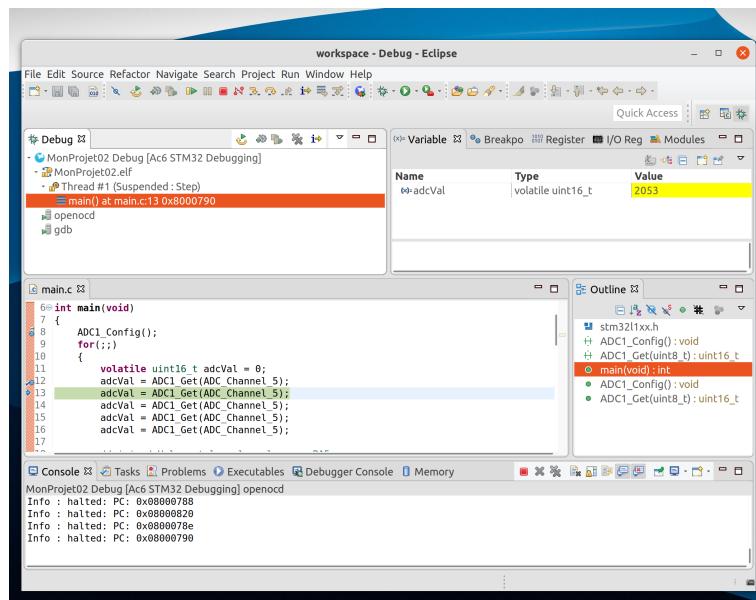


Alors le programme va exécuter la ligne 8, se positionner en ligne 9, et attendre. Pour l'instant ça n'est pas très intéressant, mais allons, avec la même icône, jusqu'à la ligne 12, en appuyant plusieurs fois, mais sans exécuter la ligne 12 : ainsi, nous aurons successivement les lignes 10 et 11.

Or à la ligne 11, il est indiqué la création de la variable `adVal`, et son initialisation à 0. Une fois fait, on se retrouve dans la situation ci-dessous :



On voit bien que dans la partie "variables", la variable `adcVal` apparait, et sa valeur est bien 0. Maintenant exécutons la ligne 12 elle-même. Compte tenu de son code, elle doit réaliser une mesure sur l'ADC. Nous avons injecté une tension de 1.5 V, et souvenons nous que l'ADC est 12 bit. Vous voyez ceci :

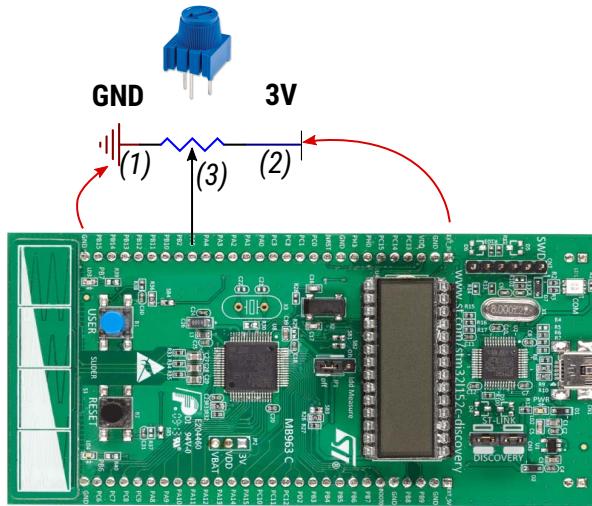


On voit une valeur de 2053.

- ② **Q-3.2.1:** Est-ce que cette valeur est cohérente, au moins approximativement ?
- ② **Q-3.2.2:** Si vous n'y êtes pas, faites en sorte de vous mettre dans la même situation.
- ② **Q-3.2.3:** Lancez maintenant l'exécution de la ligne suivante (ligne 13), puis de la ligne suivante, et encore la suivante, mais n'exécutez pas encore la ligne 16. Qu'observez-vous ? Et pour la ligne suivante ? D'où proviennent ces variations à votre avis ?
- ② **Q-3.2.4:** Maintenant, au lieu d'utiliser "avancer d'un pas", cliquez sur l'option "entrez dans une fonction" (revoyez la copie d'écran qui comporte des commentaires). Qu'observez-vous ?

III.3 Mini Projet : rapport cyclique imposé par un potentiomètre

On veut faire clignoter la LED du port PB7 en fonction de la tension analogique vue sur le port PA5. On va utiliser un potentiomètre pour choisir la tension analogique. On veut réaliser le circuit ci-dessous :



Le potentiomètre qui vous est fourni a une résistance de $1\text{ k}\Omega$ entre les bornes notées (1) et (2) du schéma ci dessus.

- ② **Q-3.3.1:** Avec un ohmètre, identifiez quelles pattes du potentiomètre correspondent aux noeuds (1), (2) et (3) du schéma. Si ce n'est pas clair demandez à votre enseignant.
- ② **Q-3.3.2:** Avec cette information, réalisez le câblage demandé. Il ne faut **surtout pas** que (3) se retrouve branchée directement à GND ou à 3 V, sinon vous risquez de griller le potentiomètre et peut-être la carte (Si vous ne voyez pas pourquoi demandez à votre enseignant)
- ② **Q-3.3.3:** En reprenant le code plus-haut, changez la position du curseur du potentiomètre et observez, dans le debugguer, que l'état du convertisseur change.
- ② **Q-3.3.4:** Maintenant, réalisez un code qui fait clignoter la LED selon les 4 rapports cycliques ci-dessous, en fonction de la tension vue que PA5 :

- pour $\text{PA5} < 0.75 \text{ V}$, elle doit être allumée pendant 100 ms et éteinte pendant 900 ms
- pour $0.75 \text{ V} \leq \text{PA5} < 1.5 \text{ V}$, elle doit être allumée pendant 200 ms et éteinte pendant 800 ms
- pour $1.5 \text{ V} \leq \text{PA5} < 2.25 \text{ V}$, elle doit être allumée pendant 500 ms et éteinte pendant 500 ms
- pour $\text{PA5} \geq 2.25 \text{ V}$, elle doit être allumée pendant 900 ms et éteinte pendant 100 ms

- ② **Q-3.3.5:** Divisez par 40 toutes les constantes de temps du code précédent. Qu'observez-vous alors ?
- ② **Q-3.3.6:** Reprenez le code avec les constantes de temps initiales. A partir de ces ordres de grandeur, faites en sorte que le rapport cyclique s'adapte de façon graduelle à la tension lue sur PA5. On ne cherchera pas à multiplier les cas, mais à trouver une relation "intelligente", mathématique (pas compliquée) entre la tension lue et le rapport cyclique imposé.
- ② **Q-3.3.7:** A nouveau, divisez par 40 les constantes de temps mises en jeu. Qu'avez-vous réalisé ?
- ② **Q-3.3.8:** Visualisez les signaux sur PB7 avec l'oscilloscope.

Remarque : Ce que vous venez de faire : vous avez programmé "à la main" une façon de piloter l'amplitude d'un signal en ne disposant que d'amplitudes logiques 0 ou 1. Nous verrons que c'est un mode de fonctionnement très classique sur microncontrôleur, que les microcontrôleurs disposent de sous-ensembles pour gérer cela directement, sans avoir à vraiment le programmer. C'est ce que l'on appelle le "PWM" pour Pulse Width Modulation.

Séance IV – Génération de signaux analogiques : PWM

⌚ Motivations et objectifs

Générer des signaux analogiques en mode "tout-ou-rien"

IV.1 Prise en main des signaux PWM

On redonne le code de base vu en cours pour la génération de signaux PWM :

```
1 #include "stm32l1xx.h"
2
3 void TIM4_PWM_Config();
4 void TIM4_PWM_Set(uint16_t pulseWidth);
5
6 int main(void)
7 {
8     TIM4_PWM_Config();
9
10    float duty = 0;
11    float step;
12    while(1)
13    {
14        duty *= step;
15        if (duty > 32000) {
16            duty = 32000;
17            step = 0.99;
18        }
19        if (duty < 0.001*32000) {
20            duty = 0.001*32000;
21            step = 1.01;
22        }
23
24        TIM4_PWM_Set(duty);
25
26        int k;
27        for(k=0; k<3000; k++) {
28        }
29    }
30}
31
32 // ## PWM via TIM4 sur PB7
33 // Configuration
34 void TIM4_PWM_Config()
35 {
36     /*Activer TIM4 sur APB1 */
37     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4 ,ENABLE);
38     /* Configurer TIM4 a 20 ms */
39     TIM_TimeBaseInitTypeDef timer_4;
```

```

40     TIM_TimeBaseStructInit(&timer_4);
41     timer_4.TIM_Period = 32000;
42     timer_4.TIM_Prescaler = 1;
43     TIM_TimeBaseInit(TIM4,&timer_4);
44     TIM_Cmd(TIM4, ENABLE);

45
46     /* Configurer le comparateur de sortie OC2 de TIM4 */
47     /* pour faire du PWM */
48     TIM_OCInitTypeDef timer_4_oc_2;
49     TIM_OCStructInit(&timer_4_oc_2);
50     timer_4_oc_2.TIM_OCMode =           TIM_OCMode_PWM1;
51     timer_4_oc_2.TIM_Pulse = 1000;
52     timer_4_oc_2.TIM_OutputState = TIM_OutputState_Enable;
53     TIM_OC2Init(TIM4,&timer_4_oc_2);

54
55     /*Activer GPIOB sur AHB */
56     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE);
57     /* Configurer PB7 comme "Alternative Function" pour preparer son
58      utilisation en PWM */
59     GPIO_InitTypeDef gpio_b;
60     GPIO_StructInit(&gpio_b);
61     gpio_b.GPIO_Mode = GPIO_Mode_AF;
62     gpio_b.GPIO_Pin = GPIO_Pin_7;
63     gpio_b.GPIO_Speed = GPIO_Speed_10MHz;
64     GPIO_Init(GPIOB, &gpio_b);

65     /* relier PB7 a TIM4/OC2 */
66     GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_TIM4);
67 }

68
69 // Changer rapport cyclique
70 void TIM4_PWM_Set(uint16_t pulseWidth)
71 {
72     TIM_SetCompare2(TIM4,pulseWidth);
73 }

```

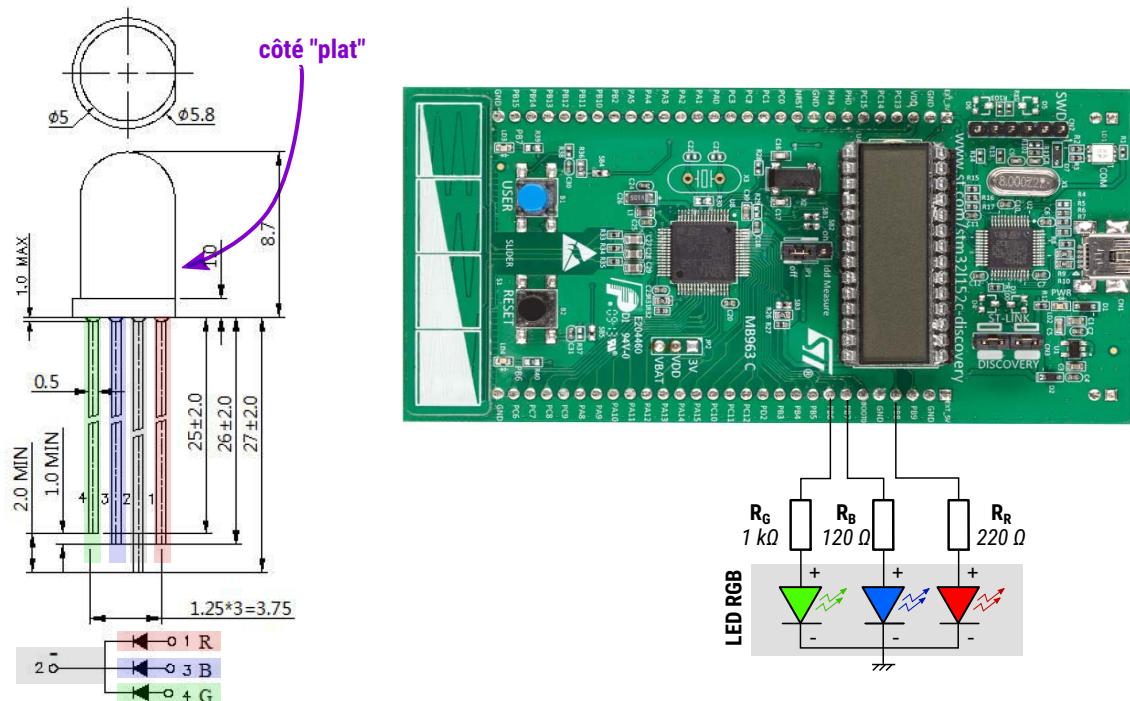
 [codeTP/codeTP-IV-1-6.c]

- ② **Q-4.1.1:** Testez ce code et regardez dans la documentation (fichier CHM) si tout est clair pour vous.
- ② **Q-4.1.2:** Testez ce code en paramétrant un timer bien plus lent en passant le prescaler à 10. Modifiez le prescaler et relancez le programme plusieurs fois, déterminez à partir de quelle valeur la lumière est stable. Déduisez-en à quelle fréquence cela correspond.
- ② **Q-4.1.3:** Vérifiez la valeur de cette fréquence avec un oscilloscope.

IV.2 Mini Projet: "LED RGB"

On se donne une LED "RGB", c'est à dire une LED qui contient 3 LED, une rouge, une verte et une bleue. La combinaison des 3 permet d'obtenir une lumière de la couleur que l'on veut, en mettant une certaine proportion de rouge, une proportion de vert et une proportion de bleu. Mais pour cela il faut pouvoir obtenir une quantité analogique pour chaque couleur, et on va utiliser le PWM pour cela.

- ② **Q-4.2.1:** Réalisez le cablage de cette LED en utilisant les ports **PB6** **PB7** et **PB8**, respectivement pour le Bleu, le Vert et le Rouge dans cet ordre, comme montré ci-dessous :



Résumé du cablage :

Couleur	Résistance de protection	Pin STM32
Vert	1 000Ω	PB6
Bleu	120Ω	PB7
Rouge	220Ω	PB8

Note : les valeurs des résistances ont été choisies, "en essayant", pour équilibrer l'intensité du rouge, du vert et du bleu pour obtenir un "blanc" à peu près neutre lorsque les 3 GPIO sont à 3V continu.

- ② **Q-4.2.2 :** Créez un nouveau projet C, indépendant du premier, dans lequel vous vous contentez de considérer PB6 PB7 et PB8 comme des sorties tout-ou-rien. Allumez tour à tour puis en les combinant les différentes LED, pour valider votre cablage et le pilotage de la LED. Par exemple vérifiez qu'en allumant uniquement **PB6** vous avez du bleu, et qu'en allumant **PB6** et **PB7** vous avez du jaune. Essayez aussi d'autres configurations sans y passer trop de temps.
- ② **Q-4.2.3 :** Adaptez le code donné dans la partie §IV.1 pour piloter en PWM les 3 LED en même temps. Dans un premier temps, vous mettrez des valeurs arbitraires fixes et relancerez le programme plusieurs fois pour tester. Observez pour certaines configuration les signaux vus sur **PB6** **PB7** et **PB8** à l'oscilloscope. Remarque : Vous n'avez pas besoin de gérer 3 timers, un seul suffit. En revanche vous devrez paramétrier 3 comparateurs de sortie ("OC") sur un seul et même timer. En effet, le STM32 peut en gérer jusqu'à 4 par timer. Regardez dans le "**datasheet**" du STM32L52 (et PAS dans le "document de référence" ni dans la documentation de la bibliothèque) pour voir la correspondance entre les différents "OC" (aussi appelés "**TIMx_CH**") et les différentes PIN du microcontrôleur.
- ② **Q-4.2.4 :** Enfin faites évoluer en fonction du temps les valeurs associées aux 3 couleurs. Vous pourrez utiliser les lois d'évolution suivantes :

$$\begin{cases} PB6(t) = OC_{max} \sin^2\left(\frac{2\pi}{30}t\right) \\ PB7(t) = OC_{max} \cos^2\left(\frac{2\pi}{22}t\right) \\ PB8(t) = OC_{max} \sin^2\left(\frac{2\pi}{12}t\right) \end{cases}$$

où OC_{max} est la valeur maximale que peut prendre le compteur PWM, et t est le temps en secondes. Dans cette version, vous réaliserez cette évolution dans la boucle du programme principal. Vous pourrez interroger la valeur du temps en regardant la valeur courante de TIM5 en récupérant le code écrit au TP II.

Séance V – Interruptions

⌚ Motivations et objectifs

On va regarder la gestion des interruptions avec les microcontrôleurs. L'objectif final est ici de pouvoir synchroniser des entrées/sorties pour traiter des signaux analogiques.

V.1 Prise en main des interruptions externes

On redonne le code vu en cours pour les interruptions externes :

```
1 #include "stm32l1xx.h"
2
3 void IRQ_EXTI0_Config();
4
5 int main(void)
6 {
7     // # LED sur PB7
8     /* Activer GPIOB sur AHB */
9     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);
10    /* Configurer PB7 comme sortie tout-ou-rien */
11    GPIO_InitTypeDef gpio_b;
12    GPIO_StructInit(&gpio_b);
13    gpio_b.GPIO_Mode = GPIO_Mode_OUT;
14    gpio_b.GPIO_Pin = GPIO_Pin_7;
15    GPIO_Init(GPIOB, &gpio_b);
16    /* allumer la LED*/
17    GPIO_SetBits(GPIOB, GPIO_Pin_7);
18
19    // configuration de l'interruption
20    IRQ_EXTI0_Config();
21
22    while(1) {
23
24    }
25}
26
27 // callback pour l'interruption externe EXTI0_IRQHandler
28 void EXTI0_IRQHandler(void)
29 {
30     if(EXTI_GetITStatus(EXTI_Line0) != RESET)
31     {
32         /* Clear the EXTI line 0 pending bit */
33         EXTI_ClearITPendingBit(EXTI_Line0);
34         GPIO_ToggleBits(GPIOB, GPIO_Pin_7);
35     }
36 }
```

```

38 // ### EXTI0 sur PA0
39 // Configuration
40 void IRQ_EXTI0_Config()
41 {
42     // # Interrupteur
43     /* Activer GPIOA sur AHB */
44     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE);
45     /* Configurer PB7 comme entrée tout-ou-rien */
46     GPIO_InitTypeDef gpio_a;
47     GPIO_StructInit(&gpio_a);
48     gpio_a.GPIO_Mode = GPIO_Mode_IN;
49     gpio_a.GPIO_Pin = GPIO_Pin_0;
50     GPIO_Init(GPIOA,&gpio_a);
51
52     /* Activer SYSCFG sur APB2
53      * pour permettre l'utilisation des interruptions externes */
54     RCC_APB2PeriphClockCmd (RCC_APB2Periph_SYSCFG,ENABLE);
55     /* Déclarer PA0 comme source d'interruption */
56     SYSCFG_EXTITLineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);
57     /* Param. des signaux qui déclencheront l'appel de EXTI0_IRQHandler()
58     * autrement dit on paramètre les signaux associés à la "ligne 0"
59     * ici : sur front montant ("Trigger_Rising")
60     */
61     EXTI_InitTypeDef EXTI0_params;
62     EXTI_StructInit(&EXTI0_params);
63     EXTI0_params.EXTI_Line = EXTI_Line0;
64     EXTI0_params.EXTI_LineCmd = ENABLE;
65     EXTI0_params.EXTI_Trigger = EXTI_Trigger_Rising;
66     EXTI_Init(&EXTI0_params);
67
68     /* Activer l'interruption dans le NVIC */
69     NVIC_InitTypeDef nvic;
70     NVIC_Init(&nvic);
71     nvic.NVIC_IRQChannel = EXTI0_IRQn;
72     nvic.NVIC_IRQChannelCmd = ENABLE;
73     NVIC_Init(&nvic);
74 }

```

 [codeTP/codeTP-V-1-7.c]

② **Q-5.1.1 :** Prenez en main ce code, testez le, injectez le dans la carte à microcontrôleur

② **Q-5.1.2 :** Modifiez le pour :

- Inverser l'état de la LED PB6 à la fois sur les fronts montants et descendants
- Inverser l'état de la LED PB7 à la fois sur les fronts montants et descendants
- Faire en sorte de respecter les deux conditions précédentes tout en ayant l'état de PB6 qui correspond à l'état inverse de PB7.

V.2 Prise en main des interruptions par Timer

On redonne le code des interruptions par Timer

```

1 #include "stm32l1xx.h"
2
3 void TIM2_IRQHandler();
4
5 int main(void)
6 {

```

```

7         TIM2_IRQHandler();
8
9     // # LED sur PB7
10    /* Activer GPIOB sur AHB */
11    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE);
12    /* Configurer PB7 comme sortie tout-ou-rien */
13    GPIO_InitTypeDef gpio_b;
14    GPIO_StructInit(&gpio_b);
15    gpio_b.GPIO_Mode = GPIO_Mode_OUT;
16    gpio_b.GPIO_Pin = GPIO_Pin_7;
17    GPIO_Init(GPIOB,&gpio_b);
18
19    while(1) {
20
21    }
22}
23
24// callback pour l'interruption periodique associee a TIM2
25void TIM2_IRQHandler() {
26    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
27    {
28        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
29        GPIO_ToggleBits(GPIOB, GPIO_Pin_7);
30    }
31}
32
33// ### TIMER 2 + IRQ a 500 ms
34// Configuration Timer 2 a 500 ms
35// avec emission d'IRQ : execute periodiquement TIM2_IRQHandler()
36void TIM2_IRQHandlerConfig()
{
37
38    /*Activer TIM2 sur APB1 */
39    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
40    /* Configurer TIM2 a 500 ms */
41    TIM_TimeBaseInitTypeDef timer_2;
42    TIM_TimeBaseStructInit(&timer_2);
43    timer_2.TIM_Prescaler = 16000-1;
44    timer_2.TIM_Period = 500;
45    TIM_TimeBaseInit(TIM2,&timer_2);
46    TIM_SetCounter(TIM2,0);
47    TIM_Cmd(TIM2, ENABLE);
48
49    /* Associer une interruption a TIM2 */
50    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
51
52    NVIC_InitTypeDef nvic;
53    /* Configuration de l'interruption */
54    nvic.NVIC_IRQChannel = TIM2_IRQn;
55    nvic.NVIC_IRQChannelPreemptionPriority = 0;
56    nvic.NVIC_IRQChannelSubPriority = 1;
57    nvic.NVIC_IRQChannelCmd = ENABLE;
58    NVIC_Init(&nvic);
59}

```

 [codeTP/codeTP-V-2-8.c]

- ② **Q-5.2.1:** Testez ce code et vérifiez que vous comprenez ce qu'il fait
- ② **Q-5.2.2:** Modifiez le pour augmenter la fréquence de l'interruption à 44kHz environ, en ajustant le Prescaler à 0 et en choisissant la période du Timer adaptée pour cela. Vérifiez que vous avez la bonne

fréquence à l'oscilloscope, en testant la pin PB7. Expliquez pourquoi il est normal, au regard du code qui a été écrit, que l'on se retrouve avec une fréquence de seulement 22kHz sur PB7. Pouvez-vous trouver un moyen de modifier cela ?

V.3 Générer un sinus

On veut maintenant générer une onde sinusoïdale avec le DAC. Malheureusement, un microcontrôleur comme le STM32 n'a pas la puissance de calcul qui permet de générer, pour chaque "top" d'interruption à 44kHz, la valeur d'un sinus (c'est lié au fait que le microcontrôleur que nous utilisons ne dispose pas de "FPU" pour "floating Point Unit"). Mais ça n'est pas perdu pour autant.

L'idée principale est de précalculer les valeurs : on va calculer 100 valeurs sur une période d'un sinus et on va les stocker dans un tableau. Ainsi, quand la fonction d'interruption arrivera, on aura juste à sortir la bonne valeur du tableau, ce qui coûte un temps de calcul très faible. Allons-y.

- ② **Q-5.3.1:** Ecrivez dans le main un code qui remplit un tableau avec les valeurs du sinus sur 100 points. Vous utiliserez malloc pour réserver la mémoire. Le tableau qui contient les valeurs du sinus sera une variable globale. Comme le sinus peut être négatif et que nous disposons d'une sortie entre 0 et +3V, on ajoutera une composante continue. Ainsi, si on note k l'indice des cases du tableau, on utilisera la formule ci-dessous pour le remplir :

$$T[k] = 511 \sin(k \times 2 \times \pi / 100) + 2047$$

Justifiez que ces valeurs sont correctes pour un convertisseur 12 bits, et calculez la tension de la composante continue, ainsi que la tension maximale et la tension minimale du signal que l'on devrait obtenir quand ces valeurs seront fournies au DAC.

- ② **Q-5.3.2:** On vous redonne ci-dessous le code vu en cours pour le DAC. En combinant le code précédent qui concerne les interruptions Timer, et le code ci-dessous qui concerne le DAC, faites en sorte de générer un sinus de fréquence 440Hz. Pour cela, vous créez un compteur nommé n, déclaré en variable globale, que vous incrémenterez à chaque fois que l'interruption timer est générée. Pour gérer la périodicité du sinus, vous utiliserez une syntaxe du type T[n%100] pour obtenir la bonne case du tableau pendant l'interruption.

```

1 #include "stm32l1xx.h"
2
3 void DAC1_Config();
4 void DAC1_Set(uint16_t value);
5
6 int main(void)
7 {
8     DAC1_Config();
9     uint16_t dac_value = 0;
10    while(1) {
11        if(dac_value <= 0) {
12            dac_value = 0xffff;
13        }
14        DAC1_Set(dac_value);
15        for(int k=0; k<200; k++) { } // perdre du temps pour "attendre"
16        dac_value = dac_value -10;
17    }
18 }
19
20 // ### DAC1 (DAC Channel 1) sur PA4
21 // Configuration
22 void DAC1_Config()
23 {
24     /*Activer GPIOA sur AHB */
25     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA , ENABLE);

```

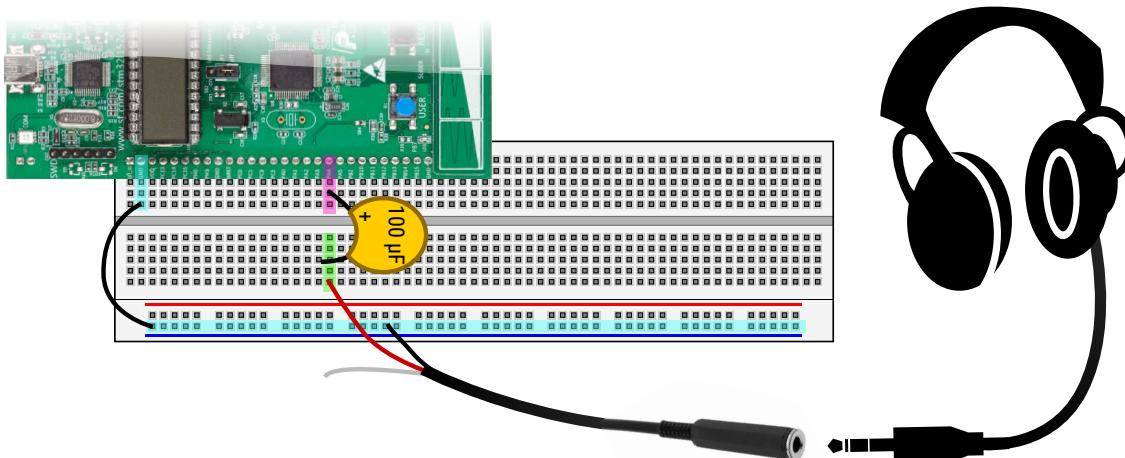
```

26  /* Configurer PA4 en mode analogique*/
27  GPIO_InitTypeDef gpio_a;
28  GPIO_StructInit(&gpio_a);
29  gpio_a.GPIO_Mode = GPIO_Mode_AN;
30  gpio_a.GPIO_Pin = GPIO_Pin_4;
31  GPIO_Init(GPIOA, &gpio_a);
32
33  /*Activer DAC sur APB1 */
34  RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
35  /* Configurer DAC1 avec parametres par defaut */
36  DAC_InitTypeDef dac_1;
37  DAC_StructInit(&dac_1);
38  DAC_Init(DAC_Channel_1, &dac_1);
39  /* Activer DAC1 */
40  DAC_Cmd(DAC_Channel_1, ENABLE);
41 }
42
43 void DAC1_Set(uint16_t value)
44 {
45  DAC_SetChannel1Data( DAC_Align_12b_R, value );
46  DAC_SoftwareTriggerCmd( DAC_Channel_1, ENABLE );
47 }
48 }
```

 [codeTP/codeTP-V-3-9.c]

- ② **Q-5.3.3:** Observez à l'oscilloscope le signal que vous obtenez, vérifiez que sa fréquence, sa composante continue, son maximum et son minimum sont conformes à ce qui a été calculé plus haut. Récupérez une trace de l'oscilloscope, et insérez la courbe correspondant dans votre rapport.
- ② **Q-5.3.4:** Une sinusoïde à 440 Hz correspond à la note "la" telle qu'elle est donnée par un diapason, autrement dit la note de référence pour les musiciens. On a donc envie d'écouter ce que ça fait ! Pour connecter un casque audio, ajoutez une capacité de forte valeur (de l'ordre de $100\mu F$), comme montré sur le cablage ci-dessous. Dans votre rapport : essayez de comprendre et expliquer pourquoi cette capacité est indispensable pour ne pas détériorer irrémédiablement, ou peut-être même détruire le haut parleur du casque.

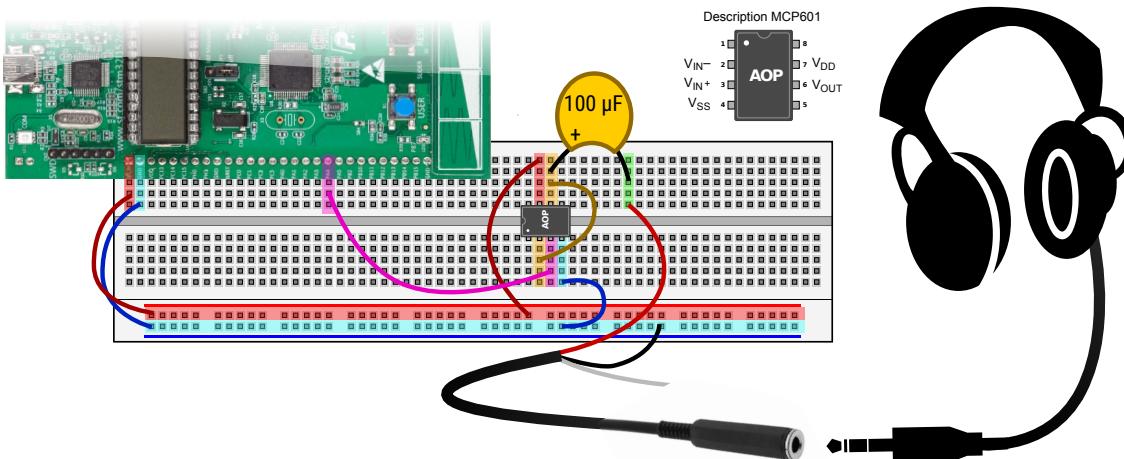
Comme c'est une capacité de forte valeur, elle est polarisée¹. Mettez *forcément* la patte polarisée "+" de la capacité au microcontrôleur, et la patte polarisée "-" sur le fil rouge de la sortie casque. Dans votre rapport, justifiez pourquoi la capacité doit être dans ce sens et pas dans l'autre. Connectez la masse de la sortie casque à la masse du microcontrôleur. Tout cela doit être conforme au schéma ci-dessous.



1. Ca n'est pas une "règle absolue" mais c'est fréquent : une capacité supérieure à $10\mu F$ est presque toujours polarisée, c'est lié à des limites technologiques dans la fabrication des condensateurs

Ecoutez maintenant "ce qui sort" par le casque. Au moment précis où vous branchez le casque, regardez le signal se détériorer à l'oscilloscope. Avez-vous une idée de ce qui se passe ? Dans votre rapport, essayez de l'interpréter en réfléchissant à la source de tension qu'est une sortie analogique du microcontrôleur, et à la charge que représente l'impédance du casque.

- ⑤ **Q-5.3.5:** Pour résoudre ce problème, on veut améliorer la qualité de la source de tension du DAC, pour qu'elle puisse "encaisser" des impédances plus petites. Ajoutez donc un montage suiveur basé sur un amplificateur opérationnel (AOP), référence MCP601. Demandez à votre enseignant cet AOP. A la sortie de l'AOP, placez la capacité, comme précédemment, puis le connecteur du casque, comme représenté ci-dessous :



Dans votre rapport, vous expliquerez pourquoi ce montage suiveur peut résoudre le problème, même si avec l'AOP choisi l'amélioration n'est pas totale (mais évidente malgré tout) : il faudrait un "meilleur" AOP (plus "cher" ou nous amenant à faire des montages plus risqués pour les cartes à microcontrôleur).

En utilisant la description des pattes du MCP601 donnée ci-dessous, essayez de faire la relation entre le montage suiveur que vous connaissez d'après vos cours théoriques (si vous ne vous souvenez pas, regardez sur internet), et le câblage réel représenté ici. Vous pouvez aussi regarder la documentation "datasheet" du MCP601 en Annexe F.

Observez maintenant que ça "se passe mieux" pour les signaux au moment où vous connectez le casque. Essayez de réaliser la différence que cela fait, au niveau du son (cela est éventuellement un peu subtil, l'AOP donné ici n'est pas si bon que ça.)

Pour finir, regardez ce qui se passe à l'oscilloscope lorsque vous branchez aussi le fil blanc du câble "jack" (qui correspond au 2ième haut-parleur du casque). Est-ce logique par rapport à ce que l'on a observé avant ?

Maintenant redébranchez le fil blanc et passez à la suite.

Remarque : Vous venez de créer un "diapason numérique".

V.4 Un mini synthétiseur

- ⑥ **Q-5.4.1:** En matière de musique, la fréquence de 440Hz correspond à un "la". On voudrait jouer la gamme de Do majeur (les touches blanches d'un piano, dans l'ordre), voici la liste des fréquences :

note	do	ré	mi	fa	sol	la	si	do
fréquence (Hz)	262	294	330	350	392	440	494	524
Période du Timer pour Prescaler = 0	611	544	484	458	408	363	323	306

Créez un tableau avec ces valeurs de prescalers, et faites en sorte qu'à chaque appui sur le bouton poussoir (**PA0**), on passe à la note suivante. Quand on est au bout du tableau, on doit revenir à la première note. Ecoutez ce que ça fait.

On donne un code de gestion d'un bouton sur **PA0**, que vous pouvez télécharger ci-dessous :

 [codeTP/codeTP-V-4-10.c]

Dans votre rapport, vous justifierez, par une formule que vous établirez, la relation qu'il y a entre la fréquence voulue pour la note et la valeur du prescaler.

- ② **Q-5.4.2:** Si vous avez envie, vous pouvez faire "do mineur" en remplaçant :
- le *mi* par *mi bémol*, fréquence = 312 Hz, période Timer = 513
 - le *la* par *la bémol*, fréquence = 416 Hz, période du Timer = 385
- ② **Q-5.4.3:** Générez des signaux "dents de scie" à la place du sinus. On donne un code qui vous aidera beaucoup, à adapter au contexte de votre programme :

```

1 //note : Nmax=100 est suppose etre une variable globale
2 //          y est suppose etre un pointeur en variable globale
3 void buildSawTooth() {
4     y = malloc(sizeof(float));
5     float t = 0;
6     for(int k=0; k<Nmax; k++) {
7         y[k] = 2047-511 + (k*1022/Nmax);
8     }
9 }
```

 [codeTP/codeTP-V-4-11.c]

- ② **Q-5.4.4:** Générez des signaux "triangles" à la place du sinus. On donne un code qui vous aidera beaucoup, à adapter au contexte de votre programme :

```

1 //note : Nmax=100 est suppose etre une variable globale
2 //          y est suppose etre un pointeur en variable globale
3 void buildTriangle() {
4     y = malloc(sizeof(float));
5     float t = 0;
6     int k;
7     for( k=0; k<Nmax/2; k++) {
8         y[k] = 2047-511 + (k*1022*2/Nmax);
9     }
10    for( ; k<Nmax; k++) {
11        y[k] = 2047 + 511 -((k-Nmax/2)*1022*2/Nmax);
12    }
13 }
```

 [codeTP/codeTP-V-4-12.c]

- ② **Q-5.4.5:** Comparez successivement le son des 3 (sinus, triangle, dent de scie), lequel semble "le plus agressif" à l'oreille, c'est à dire "contenir le plus d'aigus"? Pouvez-vous donner une interprétation (sans faire de calcul) en matière de séries de Fourier ?

V.5 Mini Projet : Filtrage numérique d'un signal

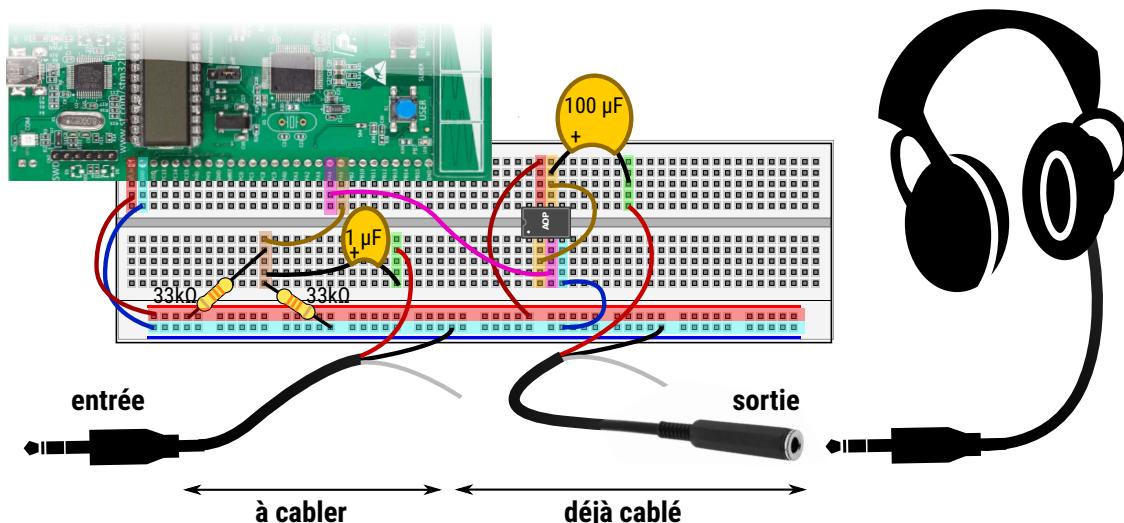
On veut maintenant réaliser le filtrage numérique d'un signal. On a donc besoin d'un ADC, pour faire l'acquisition du signal, et du DAC, pour régénérer le signal après transformation.

Aspect Circuit Electronique/Cablage

Les signaux audio sont des signaux alternatifs centrés autour de 0V, et dont la bande passante est $20\text{Hz}-20\text{kHz}$ ². Une sortie "casque", sur un PC ou sur un téléphone, baladeur, etc, fournit un signal dont l'amplitude évolue environ dans la gamme $\pm 1\text{V}$. On peut donc assimiler une sortie "casque" à une source de tension qui délivre un signal dans la gamme $\pm 1\text{V}$. La spécification d'une sortie "casque" indique aussi que l'impédance de sortie doit être entre 50Ω et 600Ω .

Pour pouvoir utiliser une sortie casque avec le DAC du microcontrôleur, il faut que le signal soit entre 0V et 3V. On doit donc ajouter une composante continue au signal audio, typiquement de 1.5V, pour centrer le signal dans la plage permise par le DAC. Pour mémoire, nous utilisons la puce **PA5** pour l'entrée DAC.

On va donc se retrouver avec le montage ci-dessous :



Si vous le souhaitez, vous pouvez aller voir le schéma électrique son étude en annexe D (*il est utile pour votre "background" d'électronicien de passer un peu de temps à comprendre comment marche un tel circuit, mais pour des raisons de temps c'est une démarche à avoir hors des séances de TP, ce qui ne vous empêche pas de poser des questions*).

- ⑤ **Q-5.5.1:** Câblez la nouvelle partie du montage. Connectez l'entrée audio de votre circuit à la sortie casque du PC (elle est à l'avant ou à l'arrière selon le PC), et lancez une musique sur Youtube, peu importe ce qu'elle est. Mettez le son à environ 50% du volume maximum sur le PC pour éviter les saturations. Observez alors à l'oscilloscope le signal vu directement sur l'entrée (par exemple au niveau du fil rouge de l'entrée "jack"), et comparez le avec le signal vu sur PA5. Vérifiez que le circuit que nous avons câblé joue bien son rôle. Vous ajouterez les traces de l'oscilloscope dans votre rapport.

Aspect Informatique/Programmation

On veut maintenant simplement copier l'entrée audio sur la sortie audio, pour vérifier que l'acquisition et la résolution des signaux fonctionne : ce sera une base qui nous permettra, plus loin, de réaliser des transformations sur le signal. Vous pouvez simplement injecter le code de l'ADC donné dans un TP précédent. Mais nous préférons donner un code "nettoyé", spécialisé dans cette opération, et qui contient quelques optimisations. En effet, comme maintenant les fonctions d'acquisition par ADC et de génération de signal par DAC seront appelées 44000 fois par seconde, il a été utile de les optimiser en utilisant les registres au lieu des bibliothèques. Ce sont les fonctions **DAC1_Set_Quickly** et **Get_Adc_Quickly**. Ces fonctions sont intégrées au code que vous pouvez télécharger ci-dessous :

[codeTP/codeTP-V-5-13.c]

2. C'est la raison pour laquelle on échantillonne depuis le début de ce TP à 44kHz : c'est un peu plus que 2 fois la fréquence maximale, et cela correspond au "critère de Shannon" que vous verrez ou avez vu en traitement du signal.

- ② **Q-5.5.2:** (*A faire chez vous :*) Si vous le pouvez, essayez de trouver, en analysant le code de la bibliothèque et/ou le datasheet, le code donné pour `Get_Adc_Quickly()` et pour `DAC1_Set_Quickly`. Notamment, expliquez pourquoi on voit sur le code que ces fonctions sont écrites en mode "baremetal", et si vous pouvez le faire, quels registres elles modifient et pourquoi. (*c'est une question technique et difficile, mais la démarche derrière est formatrice*)
- ② **Q-5.5.3:** Prenez ce code, mettez le dans un nouveau projet, et injectez le dans le microcontrôleur. Vérifiez en injectant une entrée avec le PC et en écoutant le son avec le casque. En jouant un peu avec le code de l'interruption et pour se rassurer, divisez la valeur lue par l'ADC par 10 avant de l'injecter sur le DAC. Vérifiez que vous en entendez l'effet dans le casque. Regardez aussi à l'oscilloscope ce que cela fait. Vous insérerez les traces de l'oscilloscope dans votre rapport.

Filtre passe-bas numérique

Maintenant on va jouer avec des filtres numériques. On donne le code d'un filtre passe-bas numérique :

```
/***
 *  PasseBas
 *  Entrées :
 *      - input est la dernière valeur récupérée sur le ADC
 *      - freq_norm : fréquence de coupure exprimée en pourcentage de la
 *        bande passante totale.
 *  Sorties :
 *      - prevOutput est l'amplitude précédente. Doit pointer sur une
 *        variable globale qui vaut 0 initialement, ensuite elle évolue sans
 *        besoin d'intervenir dessus
 *  Retour :
 *      - nouvelle amplitude, après filtrage
 */
uint16_t PasseBas(uint16_t input, uint16_t freqNorm, uint16_t* prevOutput)
{
    uint16_t output = *prevOutput + freqNorm*(input - *prevOutput)/200;
    *prevOutput = output;
    return output;
}
```

 [codeTP/codeTP-V-5-14.c]

- ② **Q-5.5.4:** Insérez le dans votre code et écoutez l'effet que cela fait. Essayez des valeurs très différentes comme 10 et 100 pour `freqNorm`.

note : Dans cette question, vous avez le droit de vous contenter d'insérer cette fonction dans votre code et de l'appeler à l'endroit idoine, mais si vous en avez envie la conception et une étude rapide de ce code sont discutés en Annexe E. Si "le temps presse" gardez cette exploration théorique pour chez vous.

- ② **Q-5.5.5:** Observez simultanément la trace temporelle à l'entrée de la carte et à la sortie de la carte, et essayez de comprendre la "logique", dans l'espace des temps, de l'effet d'un filtre passe-bas sur le son. Vous insérerez les traces de l'oscilloscope dans votre rapport.

Note : on voit plus facilement ce qui se passe quand il y a de la batterie. Par exemple utilisez "I'm no good" de Amy Winehouse, que vous trouverez forcément sur youtube.

- ② **Q-5.5.6:** (optionnelle) Proposez une méthode pour comparer à l'oscilloscope le temps de calcul utilisé en présence et en absence de calcul du filtre passe-bas.

- ② **Q-5.5.7:** Discutez des avantages et inconvénients d'un filtre passe-bas analogique accompagné d'un suiveur, par rapport à la solution à microcontrôleur que nous venons d'implémenter ?

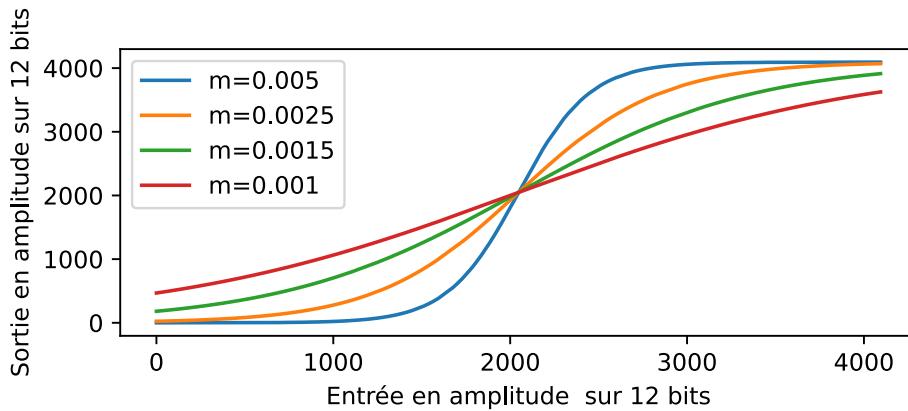
Filtre "compresseur de dynamique audio"

Un "compresseur" de son est un filtre qui amplifie les sons les plus faibles et seulement les sons les plus faibles.

 **Remarque**

Nous ne parlons pas **du tout** de compression au sens "compression des données" en informatique ! On parle ici d'un concept qui concerne les signaux analogiques.

Cette "compression audio" est utile dans beaucoup de situations. Par exemple si vous écoutez de la musique dans une voiture, le bruit du moteur peut masquer les sons faibles. Alors un compresseur va pouvoir "remonter" les sons faibles tout en maintenant les sons forts au même niveau. C'est donc un filtre "non-linéaire". On a représenté la fonction de transfert en amplitude (attention, ce n'est pas un diagramme de Bode !!) entre les niveaux d'entrée et de sortie :



Essayez de visualiser pourquoi cette courbe représente bien une compression du son, et que cette compression est forte pour $m = 0.005$ et presque inexiste pour $m = 0.001$. Notamment : pourquoi le simple tracé de ces courbes montre un "gain" pour les faibles amplitudes et une atténuation pour les fortes amplitudes ?

On donne aussi son équation, qui une appelée "sigmoïde" un peu modifiée pour l'adapter aux niveaux de l'ADC :

$$S = \frac{4095}{1 + \exp(-m \times (E - 2047))}$$

où S est l'amplitude de sortie, E l'amplitude d'entrée, et m le coefficient de compression, qui vaut ici entre 0.001 (pas de compression) et 0.005 (compression très forte). Notez les valeurs 4095, le maximum que l'on peut avoir avec 12 bit, et 2047, qui est la moitié.

On donne le code d'un compresseur :

```
/**
 * initialise_Compresseur : - prepare la memoire pour un compresseur pour
 * un DAC 12 bit
 *                      - precalcule la fonction de compression
 * Entrées : - m : coefficient de compression, doit etre situe entre
 *            0.001 et 0.005
 * Sorties :
 * Retour : tableau contenant la fonction de transfert d'un compresseur
 *
 */
uint16_t* initialise_Compresseur(float m)
{
    uint16_t nbPts = 2<<12;
    uint16_t* stock = malloc(nbPts*sizeof(uint16_t));
    for(int k=0;k< nbPts; k++) {
        stock[k] = 4095 /(1+exp(-m*(k-2047)));
    }
    return stock;
}
```

```
/**  
 *  compresseur : applique la fonction de compression au signal fourni en  
 *  entree  
 *  Entrees : - input : nouvelle valeur fournie par le DAC  
 *             - stock : fonction de compression precalculee par  
 *               initialise_Compresseur  
 *  Sorties :  
 *  Retour : valeur de 'input' filtree par la fonction de compression  
 *  
 */  
uint16_t compresseur(uint16_t input,uint16_t* stock)  
{  
    input = input & 0xFFFF; // s'assurer que l'entree est bien sur 12  
    bit  
    return stock[input];  
}
```

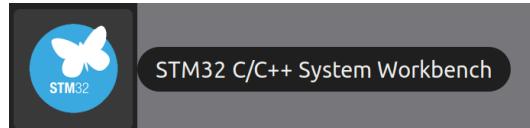
 [codeTP/codeTP-V-5-15.c]

- ② **Q-5.5.8:** Essayez ce filtre et écoutez son effet dans un casque. Ensuite, cherchez sur internet la video "1kHz sine tone youtube", qui génère un sinus, et regardez l'effet sur l'oscilloscope, pour bien comprendre ce que fait ce filtre. Vous ajouterez les traces de l'oscilloscope dans votre rapport.
- ② **Q-5.5.9:** Expliquez pourquoi on dit que ce filtre est "non-linéaire", alors que l'on dit du passe-bas qu'il est linéaire.

Annexes

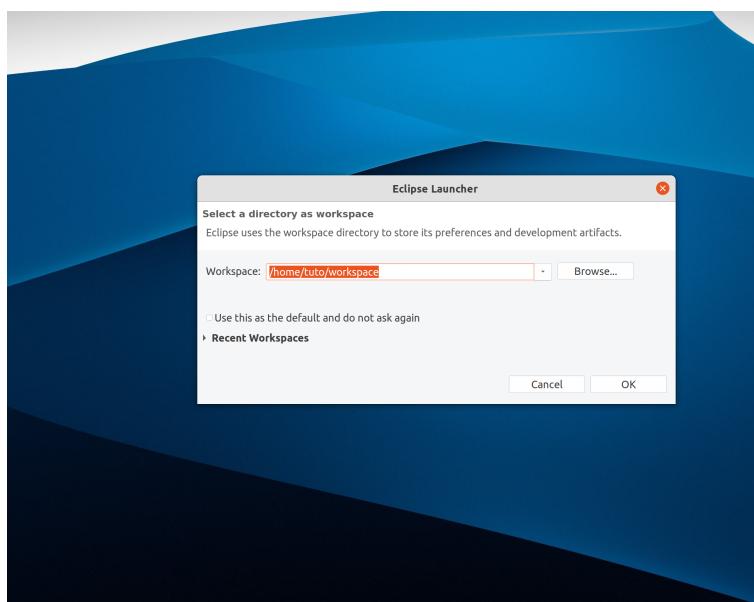
Annexe A - Création d'un projet en mode "Bare-Metal"

1



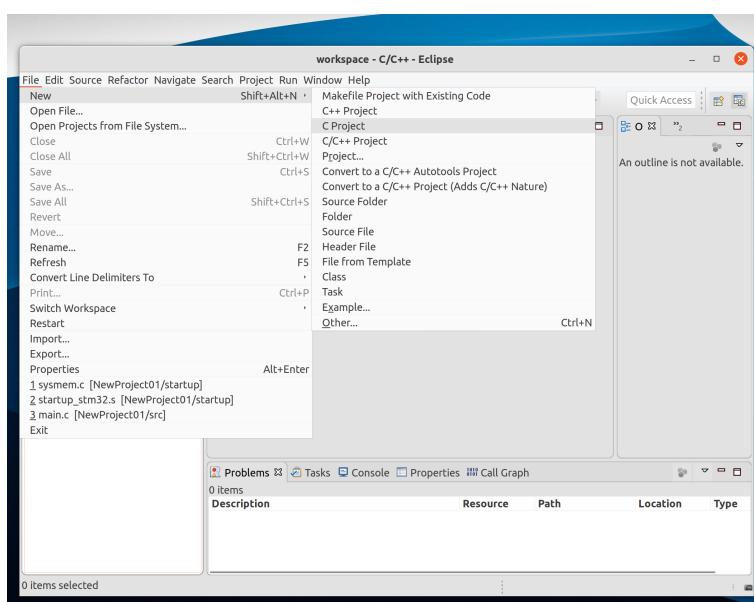
Lancer le logiciel "*STM32 C/C++ System Workbench*"

2

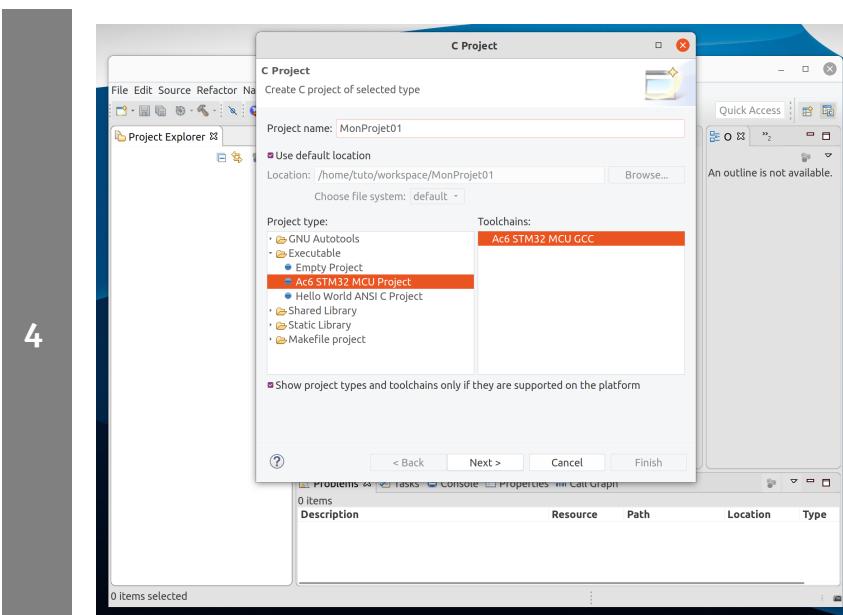


Le logiciel demande où les projets doivent être créés. Vous pouvez garder la proposition par défaut : validez en appuyant sur *OK*.

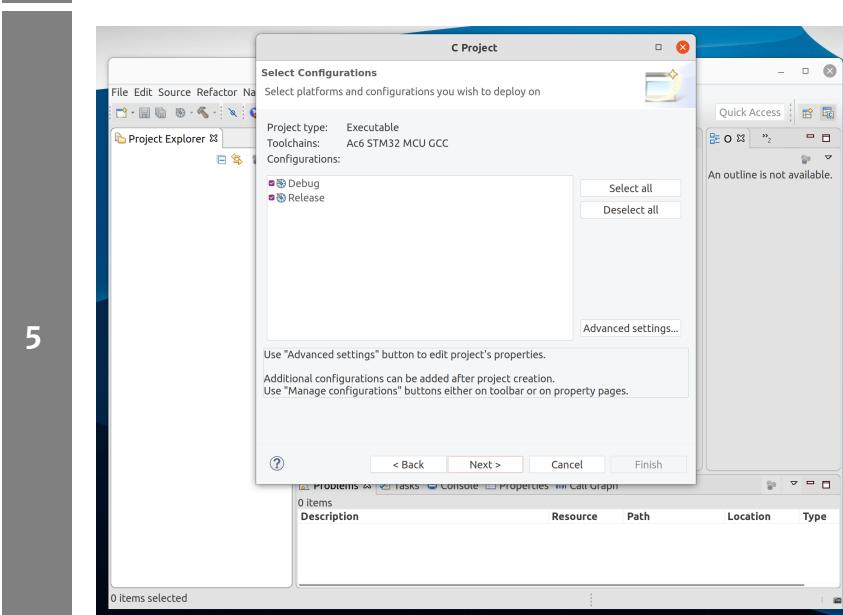
3



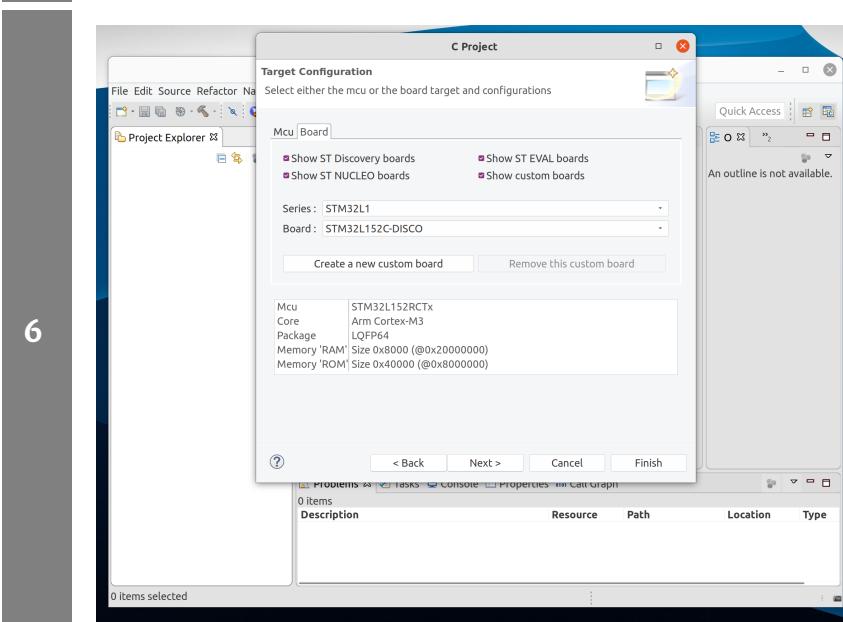
Le logiciel est maintenant chargé. Créez un projet en allant dans le menu *File* ▶ *New* ▶ *C Project*.



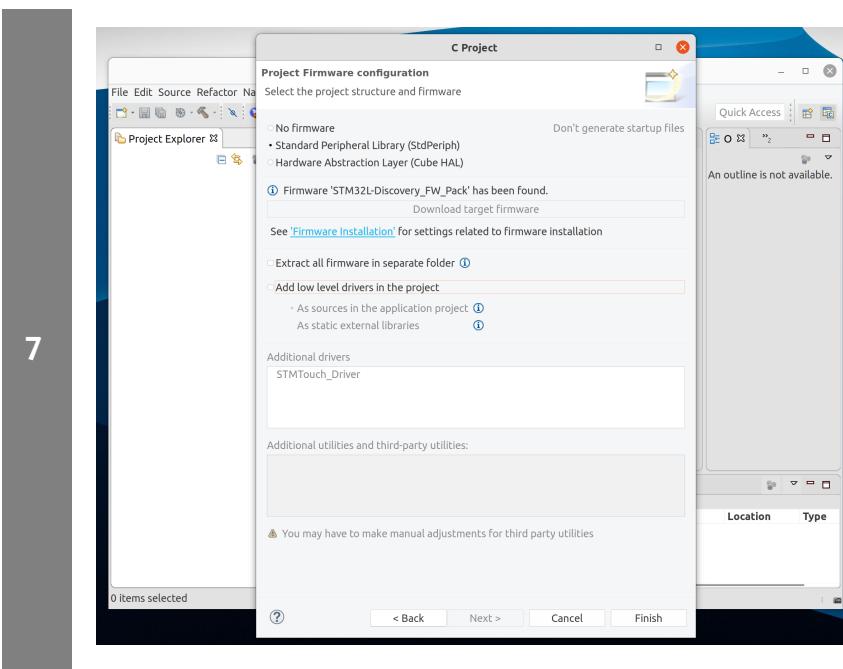
Le logiciel demande quel type de projet il faut créer. Pour faire un projet pour STM32, vous devez sélectionner *Ac6 STM32 MCU Project* dans la partie "*Projet*" (à gauche), et *Ac6 STM32 MCU GCC* dans "*Toolchains*" (à droite). Dans "*Project Name*" en haut, tapez un nom pour le projet. Ici on a choisi "*MonProjet01*" mais ça pourrait être autre chose. Puis appuyez sur "*Next >*".



Le logiciel demande les configurations de compilation à créer. En l'état il va créer la configuration *Release* et *Debug*, ce qui est parfait. Appuyez sur "*Next >*".

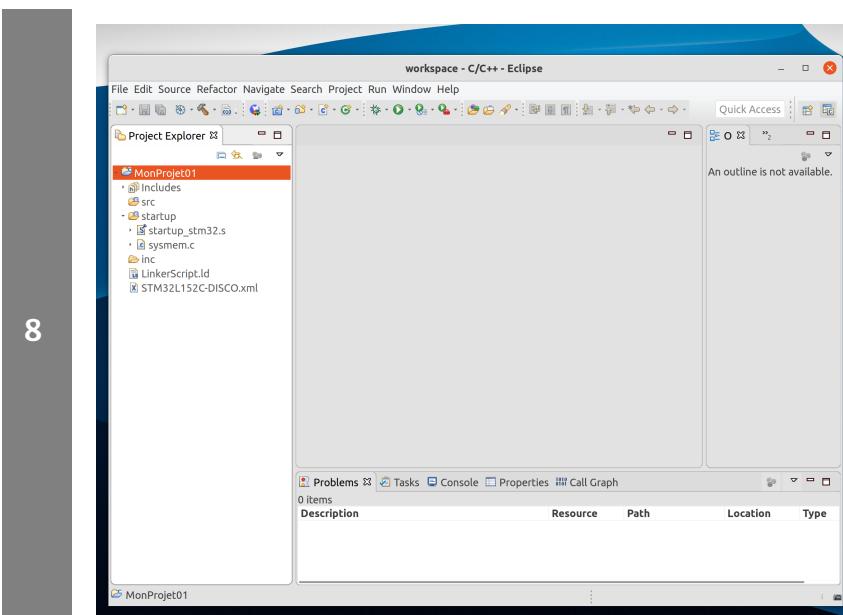


Le logiciel demande pour quel microcontrôleur et pour quelle carte à microcontrôleur le projet doit être préparé. C'est important parce que c'est ce qui va notamment permettre au logiciel de générer la séquence de démarrage du microcontrôleur (que du coup on n'a pas à écrire et c'est tant mieux parce que ça se fait en assembleur et c'est pas simple du tout!). On va sans surprise sélectionner la famille "*STM32L1*" et la carte "*STM32L152C-DISCO*". Puis appuyez sur "*Next >*".



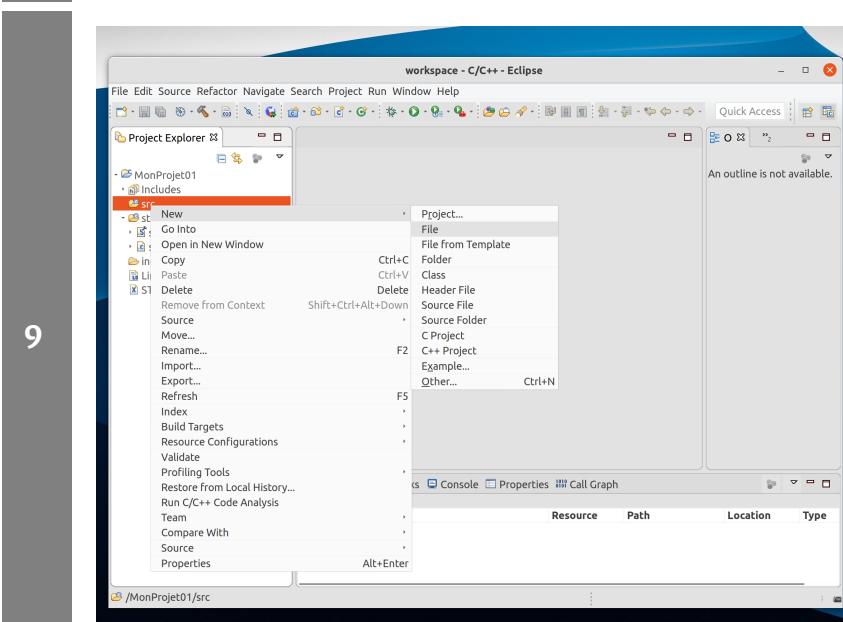
7

Maintenant on choisit quels codes préparés (bibliothèques, drivers, etc) doivent être intégrés au projet. Il faut commencer par cliquer sur "Download target firmware" si vous ne l'avez jamais fait, pour télécharger les bibliothèques et la séquence de démarrage pour le microcontrôleur et la carte que l'on utilise. Puis, pour disposer de la séquence de démarrage MAIS pas des bibliothèques (puisque l'on veut faire un projet "Bare-Metal"), il faut **ACTIVER** (paradoxalement) "*Standard Peripheral Library StdPeriph*" et **DESACTIVER** "*Add low level drivers in the project*". Puis appuyez sur "*Finish*".



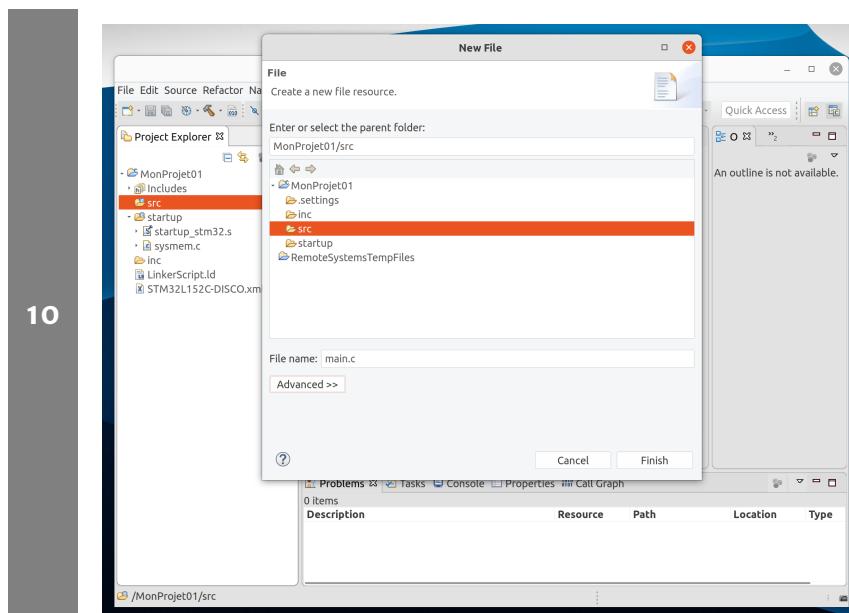
8

Le projet est maintenant créé, vous en voyez la structure sur la gauche, dans la colonne "*Project Explorer*". Vous pouvez ouvrir les différents éléments (*src*, *startup*, *inc* ...), comme montré ici. Vous verrez qu'il n'y a pas de programme principal. En revanche vous pouvez jeter un œil à "startump_stm31.s" qui est la séquence de démarrage écrite en assembleur. Ne la modifiez surtout pas, c'est juste de la curiosité, pour voir déjà à quoi ressemble de l'assembleur, et pourquoi pas quelques ingrédients on trouve dans une séquence de démarrage.

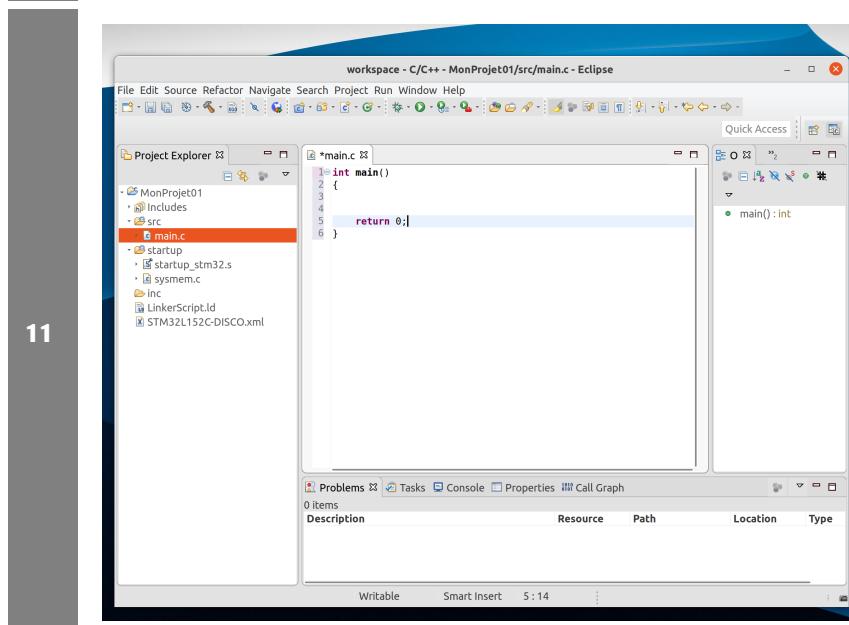


9

Il faut maintenant ajouter un programme principal. Cliquez avec le bouton droit de la souris sur *src*, dans la colonne de gauche "*Project Explorer*". Cette sous-section du projet servira à contenir tous les fichier C que vous créez vous-même. Dans le menu qui s'ouvre alors, sélectionnez *New ▶ File*.



Le logiciel demande quel nom donner au fichier C que l'on veut ajouter. On va choisir (ce n'est pas une obligation mais c'est plus clair) `main.c`. Et donc tapez `main.c`, puis cliquez sur "*Finish*"



Tapez un code minimum pour créer une fonction `main`, comme montré sur la copie d'écran. Votre projet est créé !.

Pour valider votre projet et faire un test concret, vous pouvez utiliser un code de test en mode "bare-metal". On en donne un ci-dessous qui doit faire clignoter une LED. On en donne un ci-dessous qui doit faire clignoter une LED. Ne faites pas un copier/coller directement à partir du PDF : utilisez le système de téléchargement qui apparaît après le code en bleu, et utilisez le fichier téléchargé pour faire un copier/coller dans votre `main.c` :

```

1 int main() {
2     long* RCC_AHBENR = (long*) (0x40023800 + 0x1C) ;
3     (*RCC_AHBENR) = (*RCC_AHBENR) | (1<<1); // code definitif
4
5     long* GPIOB_MODER = (long*) (0x40020400 + 0x0);
6     (*GPIOB_MODER) = (*GPIOB_MODER) & (~(0b11<<14));
7     (*GPIOB_MODER) = (*GPIOB_MODER) | (0b01<<14);
8
9     long* GPIOB_ODR = (long*) (0x40020400 + 0x14);
10    while(1) {
11        (*GPIOB_ODR) = (*GPIOB_ODR) | (0b1<<7);

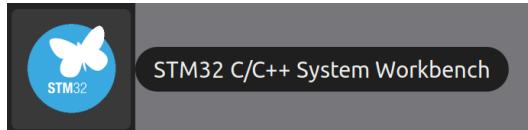
```

```
12     for(int k=0; k<100000;k++) { } // perdre du temps pour "attendre"
13     (*GPIOB_ODR) = (*GPIOB_ODR) & (~(0b1<<7));
14     for(int k=0; k<100000;k++) { } // perdre du temps pour "attendre"
15 }
16 return 0;
17 }
```

 [codeTP/codeTP-A-0-16.c]

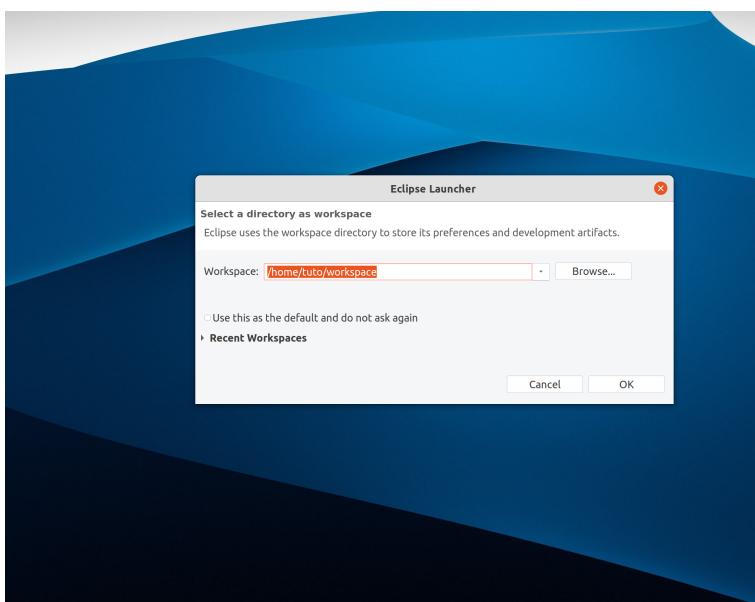
Annexe B – Création d'un projet en mode "ST Standard Library"

1



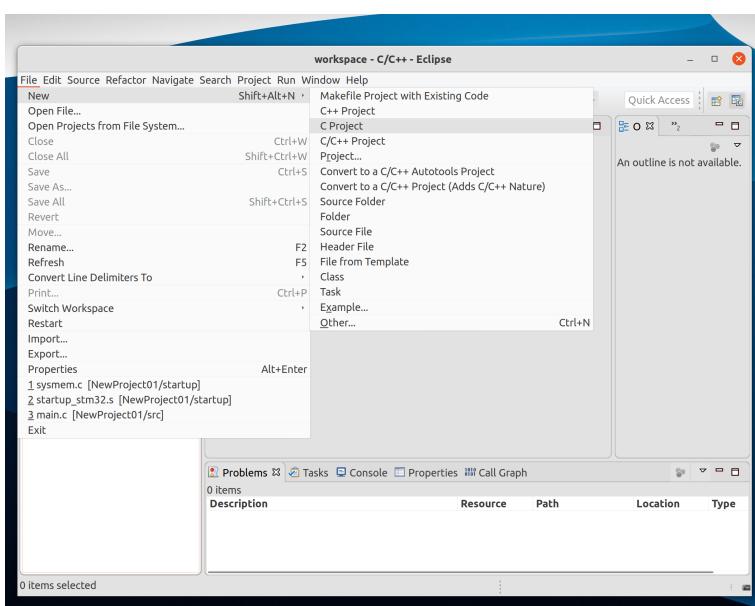
Lancer le logiciel "*STM32 C/C++ System Workbench*"

2

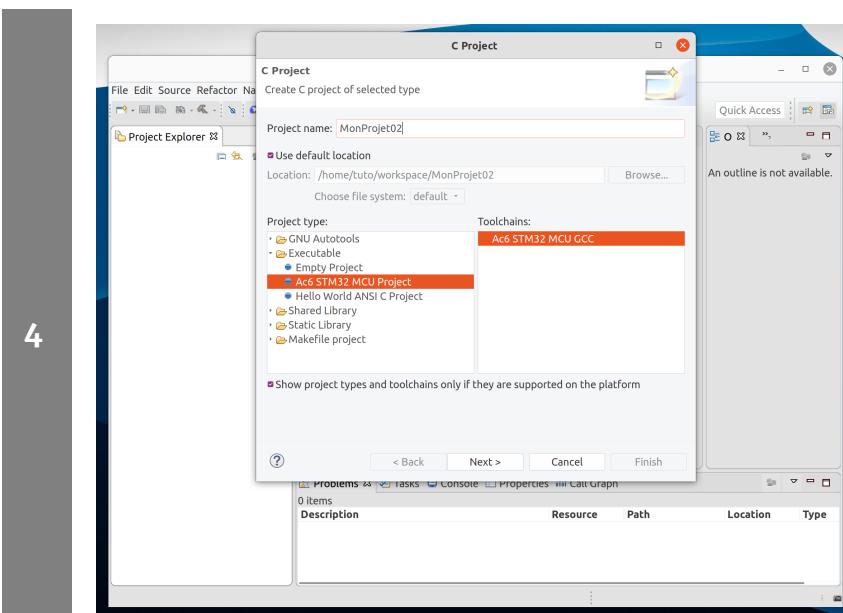


Le logiciel demande où les projets doivent être créés. Vous pouvez garder la proposition par défaut : validez en appuyant sur *OK*.

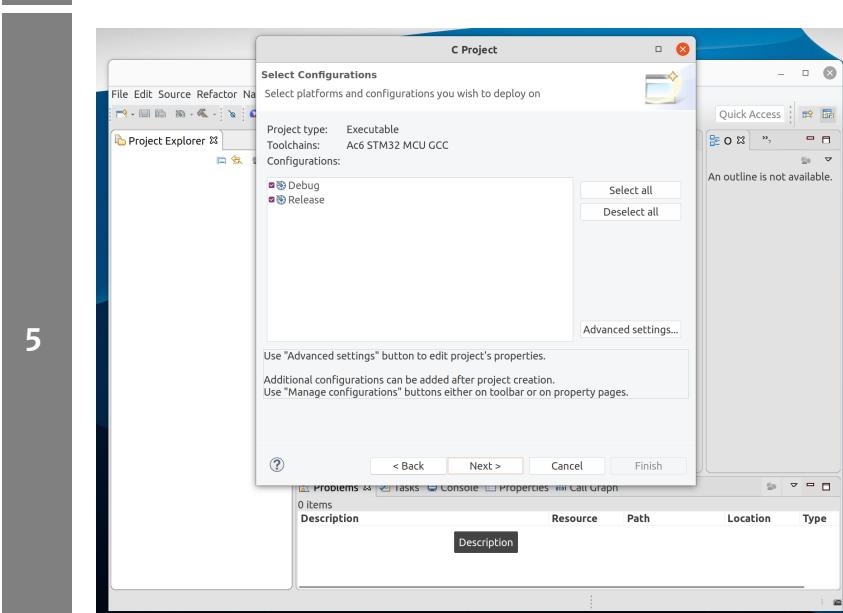
3



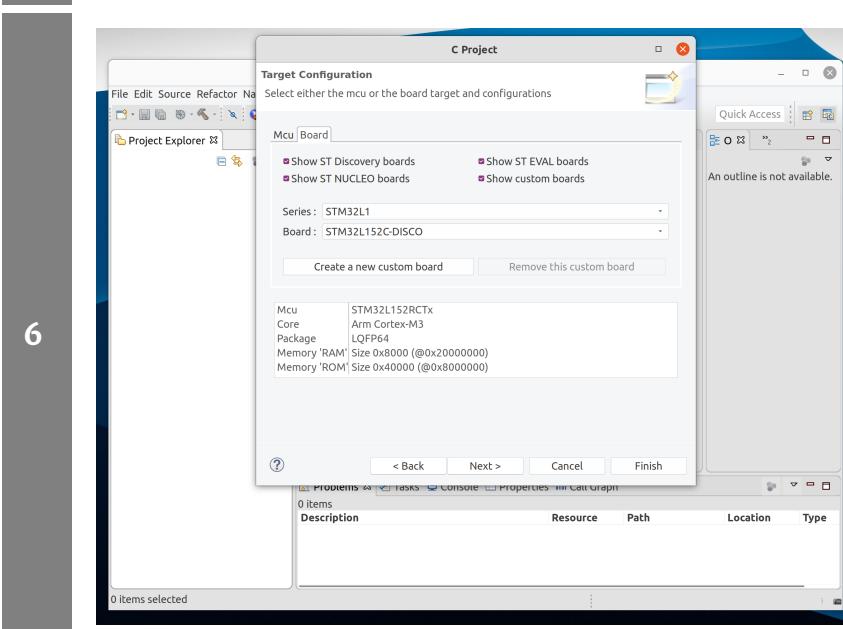
Le logiciel est maintenant chargé. Créez un projet en allant dans le menu *File* ▶ *New* ▶ *C Project*.



Le logiciel demande quel type de projet il faut créer. Pour faire un projet pour STM32, vous devez sélectionner *Ac6 STM32 MCU Project* dans la partie "*Projet*" (à gauche), et *Ac6 STM32 MCU GCC* dans "*Toolchains*" (à droite). Dans "*Project Name*" en haut, tapez un nom pour le projet. Ici on a choisi "*MonProjet02*" mais ça pourrait être autre chose. Puis appuyez sur "*Next >*".

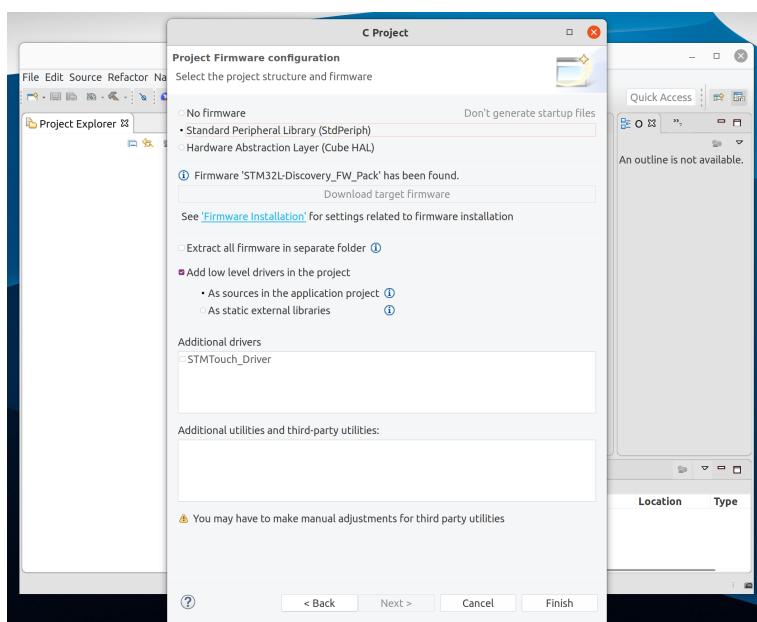


Le logiciel demande les configurations de compilation à créer. En l'état il va créer la configuration *Release* et *Debug*, ce qui est parfait. Appuyez sur "*Next >*".

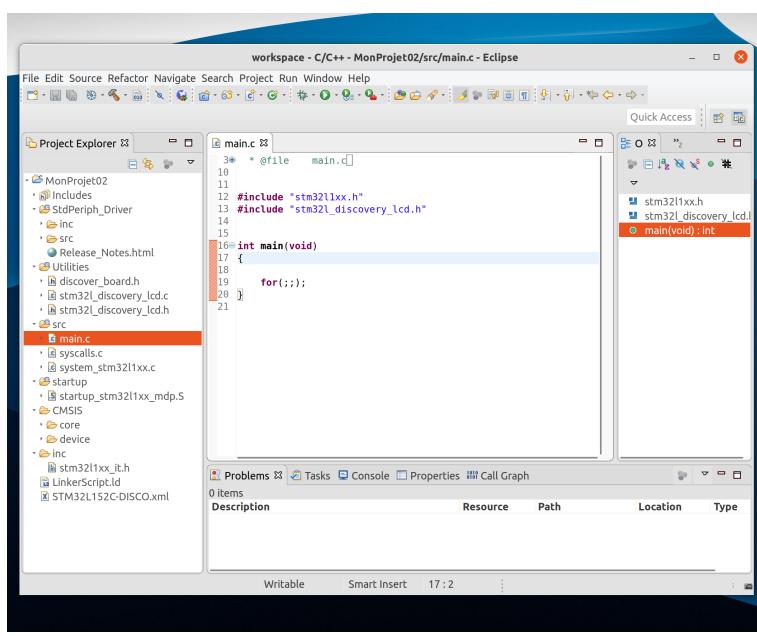


Le logiciel demande pour quel microcontrôleur et pour quelle carte à microcontrôleur le projet doit être préparé. C'est important parce que c'est ce qui va notamment permettre au logiciel de générer la séquence de démarrage du microcontrôleur (que du coup on n'a pas à écrire et c'est tant mieux parce que ça se fait en assembleur et c'est pas simple du tout!). On va sans surprise sélectionner la famille "*STM32L1*" et la carte "*STM32L152C-DISCO*". Puis appuyez sur "*Next >*".

7



8



Maintenant on choisit quels codes préparés (bibliothèques, drivers, etc) doivent être intégrés au projet. Il faut commencer par cliquer sur "Download target firmware" si vous ne l'avez jamais fait, pour télécharger les bibliothèques et la séquence de démarrage pour le microcontrôleur et la carte que l'on utilise. Puis, pour disposer de la séquence de démarrage et des bibliothèques (puisque l'on veut faire un projet "ST standard library"), il faut ACTIVER "Standard Peripheral Library StdPeriph" et ACTIVER "Add low level drivers in the project" (c'est normalement actif par défaut). Puis appuyez sur "Finish".

Le projet est maintenant créé, vous en voyez la structure sur la gauche, dans la colonne "Project Explorer". Vous pouvez ouvrir les différents éléments (*src*, *startup*, *inc* ...), comme montré ici. Vous pouvez jeter un œil à "startump_stm31.s" qui est la séquence de démarrage écrite en assembleur. Ne la modifiez surtout pas, c'est juste de la curiosité, pour voir déjà à quoi ressemble de l'assembleur, et pourquoi pas quels ingrédients on trouve dans une séquence de démarrage. Vous trouverez aussi un programme principal de base dans *main.cpp*, qui ne fait juste rien.

Pour valider votre projet et faire un test concret, vous pouvez utiliser un code de test en mode "ST Standard Library". On en donne un ci-dessous qui doit faire clignoter une LED. Ne faites pas un copier/coller directement à partir du PDF : utilisez le système de téléchargement qui apparaît après le code en bleu, et utilisez le fichier téléchargé pour faire un copier/coller dans votre *main.c* :

```

1 #include "stm32l1xx.h"
2
3 int main() {
4     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);
5
6     GPIO_InitTypeDef leds_PB;
7     GPIO_StructInit(&leds_PB);
8     leds_PB.GPIO_Mode = GPIO_Mode_OUT;
9     leds_PB.GPIO_Pin = GPIO_Pin_7;

```

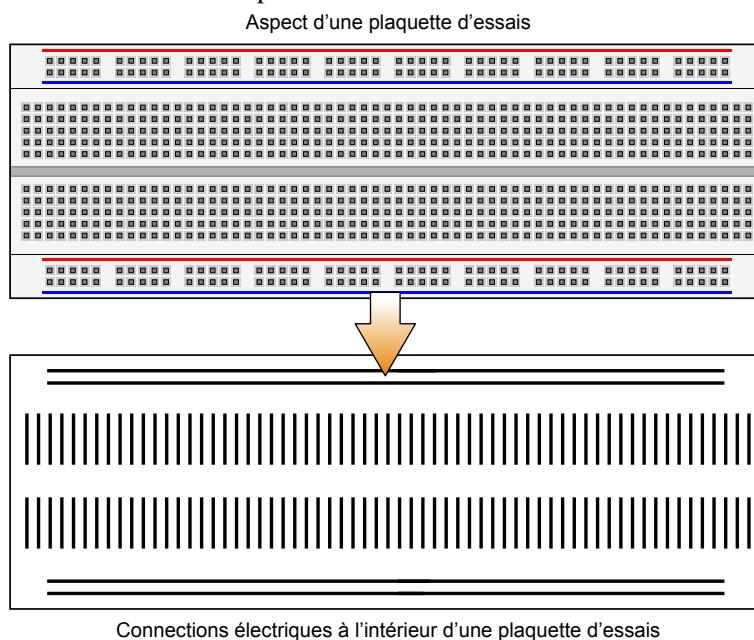
```
10 | GPIO_Init(GPIOB ,&leds_PB);  
11 |  
12 | while(1) {  
13 |     GPIO_SetBits(GPIOB ,GPIO_Pin_7);  
14 |     for(int k=0; k<100000;k++){}  
15 |     GPIO_ResetBits(GPIOB ,GPIO_Pin_7);  
16 |     for(int k=0; k<100000;k++){}  
17 | }  
18 | return 0;  
19 | }
```

 [codeTP/codeTP-B-0-17.c]

Annexe C – Cablage

C.1 Fonctionnement des plaquettes d'essai

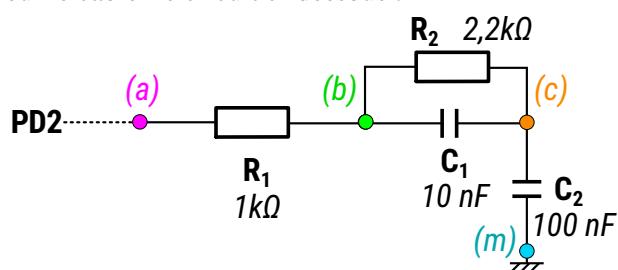
Les plaquettes d'essai vous permettront de câbler les circuits électroniques. Il faut d'abord bien comprendre comment elles sont câblées pour bien les utiliser :



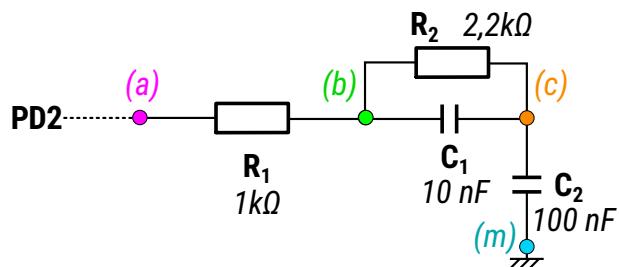
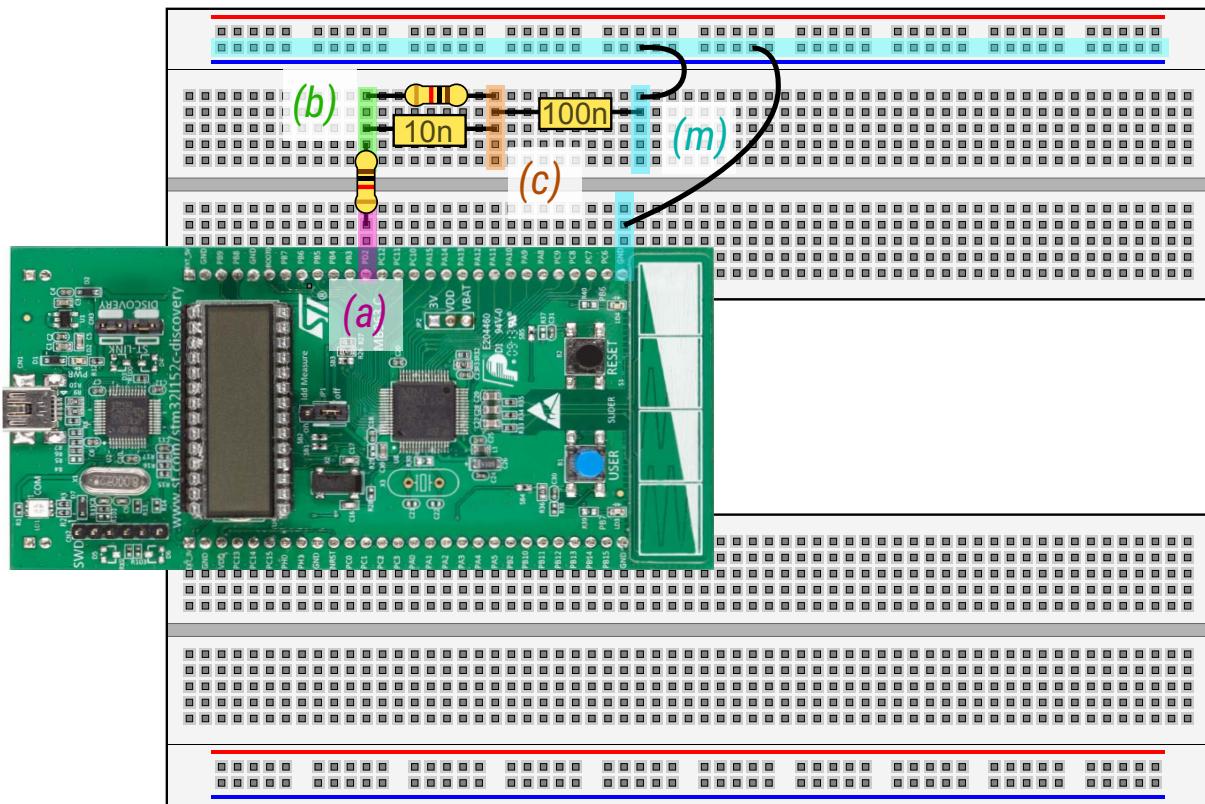
Il y a 4×2 jeux de connexions horizontales longues, typiquement pensées pour les alimentations et la masse, et un grand nombre de connexions verticales, typiquement utilisées pour les connexions entre les composants.

C.2 Exemple

Imaginons que l'on veuille cabler le circuit ci-dessous :

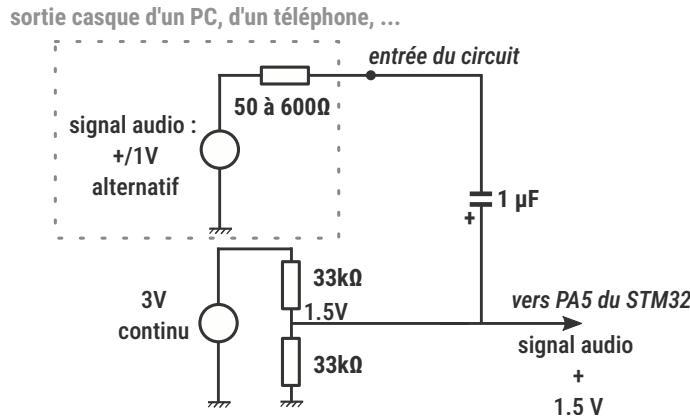


Peu importe ce qu'il est. On veut le relier à la patte PD2 du microcontrôleur. Voici ce que l'on ferait :



Annexe D – Circuit "Entrée audio"

Le schéma du circuit qui est proposé en câblage pour l'entrée audio dans le TP V est donné ci-dessous :



Le principe du circuit est le suivant : on cherche à faire la "somme" d'un signal alternatif et d'un signal continu. On pourrait penser que l'on va utiliser un sommateur à amplificateur opérationnel, et en effet parfois on n'a pas le choix. Mais ici on peut utiliser le fait que les deux signaux que l'on va additionner ne sont pas à la même fréquence : on va utiliser les principes de "filtrage" type RC ou CR, mais "à l'envers", pour combiner des signaux à des fréquences différentes au lieu de séparer des composantes de fréquence dans un signal. L'idée est la suivante :

- D'abord on construit la tension continue dont on a besoin. Pour cela on utilise la tension d'alimentation du microcontrôleur, soit 3V, et on utilise 2 résistances égales pour former un pont diviseur, pour obtenir la moitié, soit 1.5V.
- On a représenté dans l'encadrement en pointillés la source de tension équivalente à un circuit de "sortie casque" tel qu'on pourrait l'avoir en sortie d'un PC, d'un téléphone, etc. Les caractéristiques de ce signal sont qu'il a une amplitude encadrée par $\pm 1\text{V}$ fonctionnant entre $20,\text{Hz}$ et $20,\text{kHz}$, et une impédance en série entre 50Ω et 600Ω . Le "pire cas" est donc 600Ω , c'est le cas où la source de tension est de plus mauvaise qualité. Si vous ne comprenez pas pourquoi, demandez à votre enseignant.
- Pour coupler ces deux sources de tension qui travaillent dans des gammes fréquences différentes, on les relie simplement par un condensateur. Pourquoi ? Rappelons que l'impédance d'un condensateur est, en module :

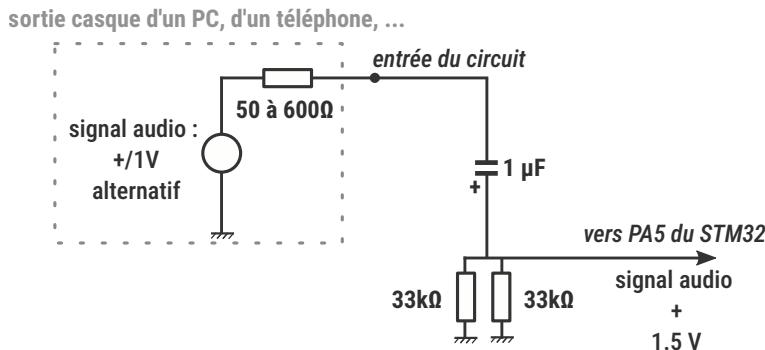
$$Z_C = \left| \frac{1}{jC2\pi f} \right| = \frac{1}{C2\pi f}$$

Cette impédance est donc infinie pour du continu ($f = 0$), et diminue au fur et à mesure que la fréquence augmente. Par exemple, pour la capacité de $1\mu\text{F}$ l'impédance à 20Hz vaut environ $8\text{k}\Omega$, et à l'autre extrémité, à 20kHz , elle vaut 8Ω environ. Ces valeurs sont assez inférieures au $33\text{k}\Omega$ du pont diviseur qui a été choisi, et donc l'impédance de la capacité a forcément un impact assez faible.

- Pour pousser un peu plus loin, regardons le point de vue du continu : comme l'impédance de la capacité est infinie en continu, alors on est certain que la source de tension continue n'a aucune interaction au noeud "vers PA5" sur le schéma : en continu, la seule source de tension vue est

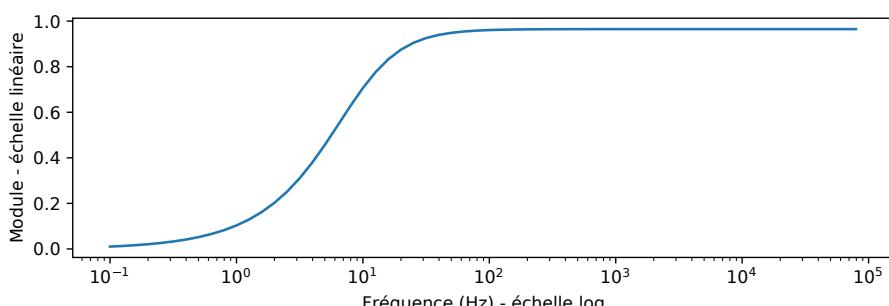
la source continue. Pourquoi c'est important ? Si la source de tension alternative était vue, elle imposerait une tension de $0V$, vu du continu, en ce point, et ça c'est quelque chose qui n'est pas faisable : on ne peut pas avoir 2 sources de tension qui imposent une valeur au même point du circuit, c'est contradictoire avec l'idée de source de tension, puisqu'une source de tension impose la valeur qu'elle décide. Donc la tension continue est bel et bien imposée par la source de tension continue, **sans interaction** avec la source audio.

- Regardons maintenant le point de vue de l'alternatif. En alternatif, on considère que les sources continues sont à $0V$, donc elles sont des courts-circuits. Donc le schéma devient :



Et donc on voit bien que vu de l'alternatif, on a un circuit C-R, donc passe-haut, formé par la capacité de $1 \mu F$ et la résistance équivalente aux 2 résistances de $33 k\Omega$ en parallèle, qui sont donc équivalentes à une résistance unique de $16.5 k\Omega$. Et c'est bien comme ça que le $33 k\Omega$ a été choisi : Comme on veut une fréquence de coupure minimum inférieure à $20 Hz$, on voit bien que $1/(2\pi RC) \approx 5 Hz$, est bien inférieur.

- On peut aussi calculer le diagramme de Bode, vu de la source audio, pour regarder quelles fréquences sont correctement communiquées à PA5 à travers ce circuit :



Et donc tout va bien !

- Pour finir, il faut bien voir que ce calcul est mené en gardant en tête que l'entrée PA5 est équivalente à une impédance infinie. C'est pas totalement juste mais c'est loin d'être faux : son impédance est très très supérieure à $1 M\Omega$, donc bien supérieure à toutes les impédances que l'on considère ici, et donc elle n'a pas d'importance pour nous en pratique.



Remarque

Au sujet de la polarité du condensateur : Il existe des condensateurs polarisés et d'autres non polarisés. Il n'y a aucun avantage à ce qu'un condensateur soit polarisé. Mais il se trouve qu'il est difficile de fabriquer des condensateurs de "grande valeur" ($> 100 nF$) non polarisés. Ce n'est pas que ça n'existe pas, on trouve des condensateurs non polarisés jusqu'à des valeurs aussi fortes que $1000 \mu F$, mais cela dépend de la tension maximale et coûte cher dans tous les cas.

Pour un condensateur de $1\mu F$ et au delà, il est habituel d'utiliser des condensateurs polarisés quand c'est possible, parce que c'est plus courant. Ici, on a mis la borne "+" du côté du $1.5V$, et c'est bien normal : on sait très bien que le $1.5V$ continu est toujours supérieur au $\pm 1V$ alternatif du signal audio, et le condensateur est donc toujours polarisé correctement.

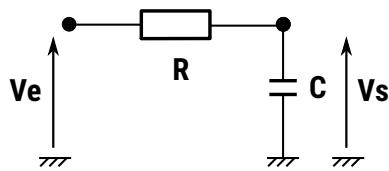
Notez qu'avec des condensateurs de fortes valeur et sous forte tension (au delà de plusieurs dizaines de Volts)), il est possible de faire "exploser" des condensateurs si ils ne sont pas polarisés correctement, et c'est à la fois vraiment spectaculaire et dangereux.

Annexe E – Filtre passe-bas numérique

Important

La démarche que l'on montre ici favorise l'intuition au détriment de la rigueur : elle n'est pas du tout générale et peut conduire à des aberrations. La bonne démarche pour créer des filtres numériques est la **transformée en z**, que vous verrez en traitement du signal ultérieurement. Ici, on utilise une "bidouille" qui fonctionne avec le filtre passe-bas mais pas avec le filtre passe-haut par exemple.

On part du schéma simple d'un circuit RC analogique :



E.1 Etablir l'équation du filtre

On calcule son équation différentielle :

$$i(t) = C \frac{dV_s(t)}{dt} \quad \text{et} \quad V_e(t) - Ri(t) - V_s(t) = 0$$

d'où :

$$V_e(t) - RC \frac{dV_s(t)}{dt} - V_s(t) = 0$$

On travaille en numérique avec un pas d'échantillonnage, donné par l'horloge TIM2 pour nous, cadencée à $f = 44000\text{Hz}$. Entre deux coups d'horloge, on a donc un temps $\Delta t = 1/44000\text{s}$. C'est notre référence de temps.

Comme on est en numérique, on n'a pas accès à tous les instants, mais seulement à ceux qui correspondent aux coups d'horloge du Timer, puisque c'est à ce moment là que l'on fait l'acquisition. On va donc écrire :

$$t = k\Delta t$$

où k est le numéro du coup de l'horloge TIM2 depuis que le microcontrôleur a démarré. Ainsi, une première transformation de l'équation différentielle, prenant cela en compte, serait :

$$V_e(k\Delta t) - RC \frac{dV_s(k\Delta t)}{dt} - V_s(k\Delta t) = 0$$

Maintenant on va s'occuper de la dérivée. La dérivée exacte n'est pas calculable en numérique, et on va la remplacer par le taux de variation. Ainsi on va réécrire les choses ainsi :

$$V_e(k\Delta t) - RC \frac{\Delta V_s(k\Delta t)}{\Delta t} - V_s(k\Delta t) = 0$$

Ecrivons maintenant cette expression comme une expression numérique ou informatique :

$$V_e[k] - RC \frac{V_s[k+1] - V_s[k]}{\Delta t} - V_s[k] = 0$$

A partir de cette expression, on peut calculer une relation de recurrence entre $V_s[k+1]$ et $V_s[k]$, autrement dit on peut calculer la nouvelle sortie du filtre à partir de l'état précédent. Faisons le :

$$V_s[k+1] = V_s[k] + \frac{\Delta t}{RC} (V_e[k] - V_s[k])$$

On sait que la constante de temps du circuit $\tau = RC = 2\pi f_c$. On peut donc réécrire le terme $\Delta t / RC$ comme proportionnel à la fréquence de coupure. On va l'écrire sous la forme d'une fraction entre deux nombres entiers plutôt que sous la forme d'un **float**, parce que les calculs à virgule flottante sont longs sur un microcontrôleur tel que celui que nous avons, dans la mesure où il ne dispose pas de "FPU" (floating-point unit). Ce n'est pas le cas de tous les microcontrôleurs, mais ça reste fréquent. Ainsi :

$$V_s[k+1] = V_s[k] + \frac{\text{freqNorm}}{200} (V_e[k] - V_s[k])$$

Le choix de la valeur 200 pour le dénominateur est un peu arbitraire, mais contient une part de bon sens : cela signifie que l'on peut avoir une précision sur la fréquence de coupure de 1/200.

E.2 Code C du filtre

C'est exactement ce que vous retrouvez dans le code :

```
/***
 *  PasseBas
 *  Entrées :
 *      - input est la dernière valeur récupérée sur le ADC
 *      - freq_norm : fréquence de coupure exprimée en pourcentage de la
 *        bande passante totale.
 *  Sorties :
 *      - prevOutput est l'amplitude précédente. Doit pointer sur une
 *        variable globale qui vaut 0 initialement, ensuite elle évolue sans
 *        besoin d'intervenir dessus
 *  Retour :
 *      - nouvelle amplitude, après filtrage
 */
uint16_t PasseBas(uint16_t input, uint16_t freqNorm, uint16_t* prevOutput)
{
    uint16_t output = *prevOutput + freqNorm*(input - *prevOutput)/200;
    *prevOutput = output;
    return output;
}
```

 [codeTP/codeTP-E-2-18.c]

Commentons le code : La fonction **PasseBas** prend comme paramètres d'entrée la dernière valeur vue sur l'ADC, et la fréquence sous une forme **uint16_t**. Cette valeur exprime la fréquence de coupure sous une forme qui est presque un pourcentage de la bande passante totale, qui vaut environ 22 kHz dans le contexte du code du TP, puisque la fréquence du Timer est 44 kHz environ.

Elle prend un dernier paramètre par adresse : la valeur de sortie du filtre précédente. C'est lié à l'équation bien entendu. Mais elle est passée par pointeur à la fonction parce que, de cette façon, la fonction met à jour cette valeur pour l'appel précédent : quand on rentre dans la fonction, elle vaut normalement l'amplitude de sortie du filtre lors du précédent appel, et quand on en sort elle vaut la même chose que **output**. Cela évite de gérer le stockage de cette valeur dans le code de l'interruption, ce qui simplifie l'appel.



Annexe F – Documentation de l'amplificateur opérationnel MCP601

Extrait du PDF

Pages 50 à 83



MCP601/1R/2/3/4

2.7V to 6.0V Single Supply CMOS Op Amps

Features

- Single-Supply: 2.7V to 6.0V
- Rail-to-Rail Output
- Input Range Includes Ground
- Gain Bandwidth Product: 2.8 MHz (typical)
- Unity-Gain Stable
- Low Quiescent Current: 230 μ A/amplifier (typical)
- Chip Select (CS): **MCP603 only**
- Temperature Ranges:
 - Industrial: -40°C to +85°C
 - Extended: -40°C to +125°C
- Available in Single, Dual, and Quad

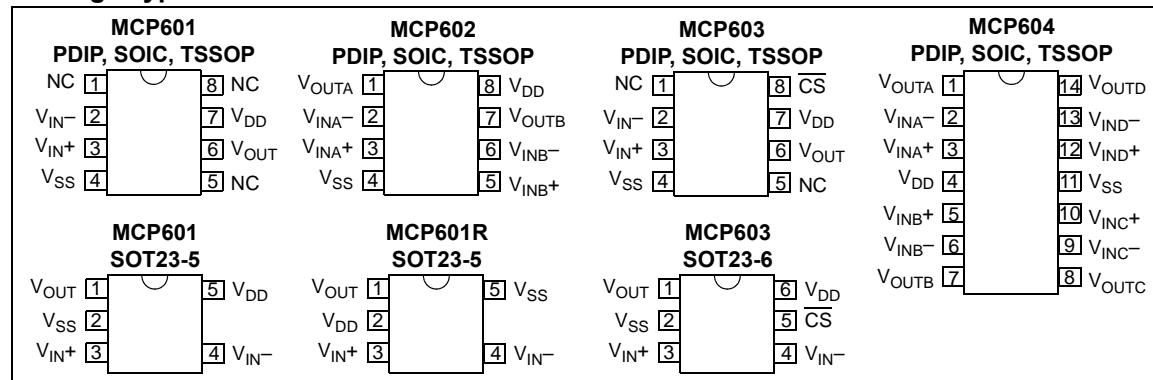
Typical Applications

- Portable Equipment
- A/D Converter Driver
- Photo Diode Pre-amp
- Analog Filters
- Data Acquisition
- Notebooks and PDAs
- Sensor Interface

Available Tools

- SPICE Macro Models
- FilterLab® Software
- Mindi™ Simulation Tool
- MAPS (Microchip Advanced Part Selector)
- Analog Demonstration and Evaluation Boards
- Application Notes

Package Types



Description

The Microchip Technology Inc. MCP601/1R/2/3/4 family of low-power operational amplifiers (op amps) are offered in single (MCP601), single with Chip Select (CS) (MCP603), dual (MCP602), and quad (MCP604) configurations. These op amps utilize an advanced CMOS technology that provides low bias current, high-speed operation, high open-loop gain, and rail-to-rail output swing. This product offering operates with a single supply voltage that can be as low as 2.7V, while drawing 230 μ A (typical) of quiescent current per amplifier. In addition, the common mode input voltage range goes 0.3V below ground, making these amplifiers ideal for single-supply operation.

These devices are appropriate for low power, battery operated circuits due to the low quiescent current, for A/D convert driver amplifiers because of their wide bandwidth or for anti-aliasing filters by virtue of their low input bias current.

The MCP601, MCP602, and MCP603 are available in standard 8-lead PDIP, SOIC, and TSSOP packages. The MCP601 and MCP601R are also available in a standard 5-lead SOT-23 package, while the MCP603 is available in a standard 6-lead SOT-23 package. The MCP604 is offered in standard 14-lead PDIP, SOIC, and TSSOP packages.

The MCP601/1R/2/3/4 family is available in the Industrial and Extended temperature ranges and has a power supply range of 2.7V to 6.0V.

MCP601/1R/2/3/4

1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings †

V _{DD} – V _{SS}	7.0V
Current at Input Pins	±2 mA
Analog Inputs (V _{IN+} , V _{IN-}) ‡	V _{SS} – 1.0V to V _{DD} + 1.0V
All Other Inputs and Outputs	V _{SS} – 0.3V to V _{DD} + 0.3V
Difference Input Voltage	V _{DD} – V _{SS}
Output Short Circuit Current	Continuous
Current at Output and Supply Pins	±30 mA
Storage Temperature.....	–65°C to +150°C
Maximum Junction Temperature (T _J)	+150°C
ESD Protection On All Pins (HBM; MM)	≥ 3 kV; 200V

† Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

‡ See Section 4.1.2 "Input Voltage and Current Limits".

DC CHARACTERISTICS

Electrical Specifications: Unless otherwise specified, T _A = +25°C, V _{DD} = +2.7V to +5.5V, V _{SS} = GND, V _{CM} = V _{DD} /2, V _{OUT} ≈ V _{DD} /2, V _L = V _{DD} /2, and R _L = 100 kΩ to V _L , and CS is tied low. (Refer to Figure 1-2 and Figure 1-3).						
Parameters	Sym	Min	Typ	Max	Units	Conditions
Input Offset						
Input Offset Voltage	V _{OS}	-2	±0.7	+2	mV	
Industrial Temperature	V _{OS}	-3	±1	+3	mV	T _A = -40°C to +85°C (Note 1)
Extended Temperature	V _{OS}	-4.5	±1	+4.5	mV	T _A = -40°C to +125°C (Note 1)
Input Offset Temperature Drift	ΔV _{OS} /ΔT _A	—	±2.5	—	μV/°C	T _A = -40°C to +125°C
Power Supply Rejection	PSRR	80	88	—	dB	V _{DD} = 2.7V to 5.5V
Input Current and Impedance						
Input Bias Current	I _B	—	1	—	pA	
Industrial Temperature	I _B	—	20	60	pA	T _A = +85°C (Note 1)
Extended Temperature	I _B	—	450	5000	pA	T _A = +125°C (Note 1)
Input Offset Current	I _{OS}	—	±1	—	pA	
Common Mode Input Impedance	Z _{CM}	—	10 ¹³ 6	—	Ω pF	
Differential Input Impedance	Z _{DIFF}	—	10 ¹³ 3	—	Ω pF	
Common Mode						
Common Mode Input Range	V _{CMR}	V _{SS} – 0.3	—	V _{DD} – 1.2	V	
Common Mode Rejection Ratio	CMRR	75	90	—	dB	V _{DD} = 5.0V, V _{CM} = -0.3V to 3.8V
Open-loop Gain						
DC Open-loop Gain (large signal)	A _{OL}	100	115	—	dB	R _L = 25 kΩ to V _L , V _{OUT} = 0.1V to V _{DD} – 0.1V
	A _{OL}	95	110	—	dB	R _L = 5 kΩ to V _L , V _{OUT} = 0.1V to V _{DD} – 0.1V
Output						
Maximum Output Voltage Swing	V _{OL} , V _{OH}	V _{SS} + 15	—	V _{DD} – 20	mV	R _L = 25 kΩ to V _L , Output overdrive = 0.5V
	V _{OL} , V _{OH}	V _{SS} + 45	—	V _{DD} – 60	mV	R _L = 5 kΩ to V _L , Output overdrive = 0.5V
Linear Output Voltage Swing	V _{OUT}	V _{SS} + 100	—	V _{DD} – 100	mV	R _L = 25 kΩ to V _L , A _{OL} ≥ 100 dB
	V _{OUT}	V _{SS} + 100	—	V _{DD} – 100	mV	R _L = 5 kΩ to V _L , A _{OL} ≥ 95 dB
Output Short Circuit Current	I _{SC}	—	±22	—	mA	V _{DD} = 5.5V
	I _{SC}	—	±12	—	mA	V _{DD} = 2.7V
Power Supply						
Supply Voltage	V _{DD}	2.7	—	6.0	V	(Note 2)
Quiescent Current per Amplifier	I _Q	—	230	325	μA	I _Q = 0

Note 1: These specifications are not tested in either the SOT-23 or TSSOP packages with date codes older than YYWW = 0408. In these cases, the minimum and maximum values are by design and characterization only.

2: All parts with date codes November 2007 and later have been screened to ensure operation at V_{DD}=6.0V. However, the other minimum and maximum specifications are measured at 1.4V and/or 5.5V.

MCP601/1R/2/3/4

AC CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, and $R_L = 100 \text{k}\Omega$ to V_L , $C_L = 50 \text{ pF}$, and CS is tied low. (Refer to Figure 1-2 and Figure 1-3).						
Parameters	Sym	Min	Typ	Max	Units	Conditions
Frequency Response						
Gain Bandwidth Product	GBWP	—	2.8	—	MHz	
Phase Margin	PM	—	50	—	°	$G = +1 \text{ V/V}$
Step Response						
Slew Rate	SR	—	2.3	—	V/ μ s	$G = +1 \text{ V/V}$
Settling Time (0.01%)	t_{settle}	—	4.5	—	μ s	$G = +1 \text{ V/V}$, 3.8V step
Noise						
Input Noise Voltage	E_{ni}	—	7	—	$\mu V_{P,P}$	$f = 0.1 \text{ Hz}$ to 10 Hz
Input Noise Voltage Density	e_{ni}	—	29	—	nV/ $\sqrt{\text{Hz}}$	$f = 1 \text{ kHz}$
	e_{ni}	—	21	—	nV/ $\sqrt{\text{Hz}}$	$f = 10 \text{ kHz}$
Input Noise Current Density	i_{ni}	—	0.6	—	fA/ $\sqrt{\text{Hz}}$	$f = 1 \text{ kHz}$

MCP603 CHIP SELECT (CS) CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, and $R_L = 100 \text{k}\Omega$ to V_L , $C_L = 50 \text{ pF}$, and CS is tied low. (Refer to Figure 1-2 and Figure 1-3).						
Parameters	Sym	Min	Typ	Max	Units	Conditions
CS Low Specifications						
CS Logic Threshold, Low	V_{IL}	V_{SS}	—	0.2 V_{DD}	V	
CS Input Current, Low	I_{CSL}	-1.0	—	—	μA	$\overline{\text{CS}} = 0.2V_{DD}$
CS High Specifications						
CS Logic Threshold, High	V_{IH}	0.8 V_{DD}	—	V_{DD}	V	
CS Input Current, High	I_{CSH}	—	0.7	2.0	μA	$\overline{\text{CS}} = V_{DD}$
Shutdown V_{SS} current	I_{Q_SHDN}	-2.0	-0.7	—	μA	$\overline{\text{CS}} = V_{DD}$
Amplifier Output Leakage in Shutdown	I_{O_SHDN}	—	1	—	nA	
Timing						
CS Low to Amplifier Output Turn-on Time	t_{ON}	—	3.1	10	μs	$\overline{\text{CS}} \leq 0.2V_{DD}$, $G = +1 \text{ V/V}$
CS High to Amplifier Output High-Z Time	t_{OFF}	—	100	—	ns	$\overline{\text{CS}} \geq 0.8V_{DD}$, $G = +1 \text{ V/V}$, No load.
Hysteresis	V_{HYST}	—	0.4	—	V	$V_{DD} = 5.0\text{V}$

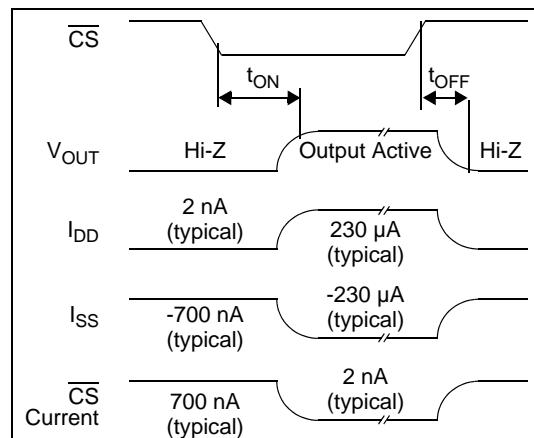


FIGURE 1-1: MCP603 Chip Select (CS) Timing Diagram.

MCP601/1R/2/3/4

TEMPERATURE CHARACTERISTICS

Electrical Specifications: Unless otherwise indicated, $V_{DD} = +2.7V$ to $+5.5V$ and $V_{SS} = GND$.

Parameters	Sym	Min	Typ	Max	Units	Conditions
Temperature Ranges						
Specified Temperature Range	T_A	-40	—	+85	°C	Industrial temperature parts
	T_A	-40	—	+125	°C	Extended temperature parts
Operating Temperature Range	T_A	-40	—	+125	°C	Note
Storage Temperature Range	T_A	-65	—	+150	°C	
Thermal Package Resistances						
Thermal Resistance, 5L-SOT23	θ_{JA}	—	256	—	°C/W	
Thermal Resistance, 6L-SOT23	θ_{JA}	—	230	—	°C/W	
Thermal Resistance, 8L-PDIP	θ_{JA}	—	85	—	°C/W	
Thermal Resistance, 8L-SOIC	θ_{JA}	—	163	—	°C/W	
Thermal Resistance, 8L-TSSOP	θ_{JA}	—	124	—	°C/W	
Thermal Resistance, 14L-PDIP	θ_{JA}	—	70	—	°C/W	
Thermal Resistance, 14L-SOIC	θ_{JA}	—	120	—	°C/W	
Thermal Resistance, 14L-TSSOP	θ_{JA}	—	100	—	°C/W	

Note: The Industrial temperature parts operate over this extended range, but with reduced performance. The Extended temperature specs do not apply to Industrial temperature parts. In any case, the internal Junction temperature (T_J) must not exceed the absolute maximum specification of 150°C.

1.1 Test Circuits

The test circuits used for the DC and AC tests are shown in [Figure 1-2](#) and [Figure 1-2](#). The bypass capacitors are laid out according to the rules discussed in [Section 4.5 “Supply Bypass”](#).

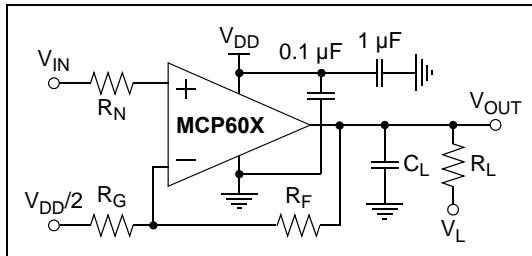


FIGURE 1-2: AC and DC Test Circuit for Most Non-Inverting Gain Conditions.

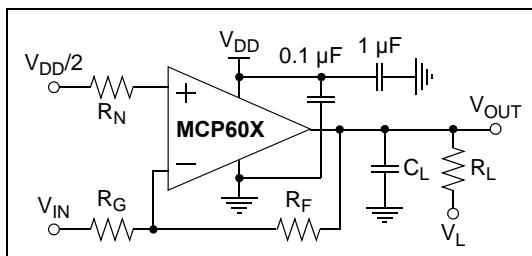


FIGURE 1-3: AC and DC Test Circuit for Most Inverting Gain Conditions.

MCP601/1R/2/3/4

2.0 TYPICAL PERFORMANCE CURVES

Note: The graphs and tables provided following this note are a statistical summary based on a limited number of samples and are provided for informational purposes only. The performance characteristics listed herein are not tested or guaranteed. In some graphs or tables, the data presented may be outside the specified operating range (e.g., outside specified power supply range) and therefore outside the warranted range.

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100 \text{k}\Omega$ to V_L , $C_L = 50 \text{pF}$ and CS is tied low.

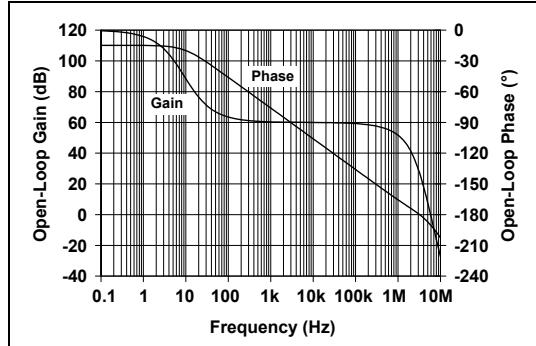


FIGURE 2-1: Open-Loop Gain, Phase vs. Frequency.

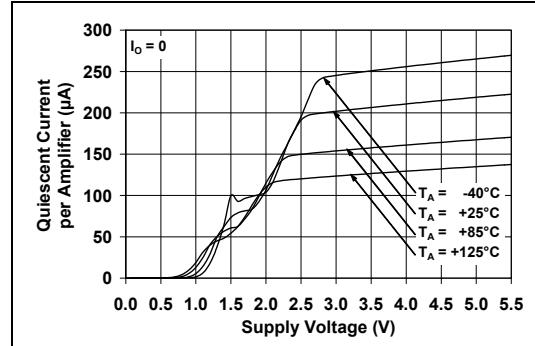


FIGURE 2-4: Quiescent Current vs. Supply Voltage.

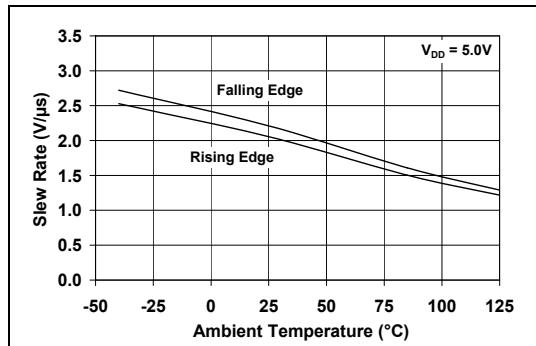


FIGURE 2-2: Slew Rate vs. Temperature.

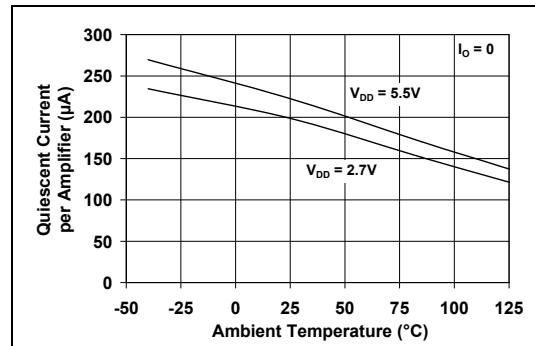


FIGURE 2-5: Quiescent Current vs. Temperature.

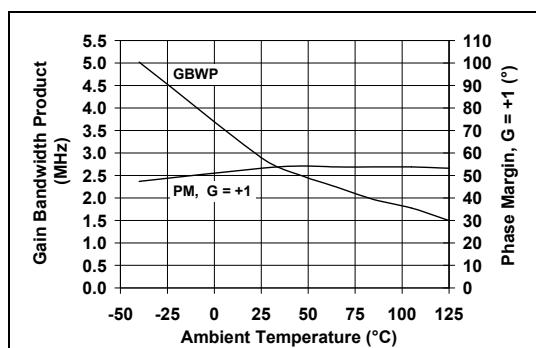


FIGURE 2-3: Gain Bandwidth Product, Phase Margin vs. Temperature.

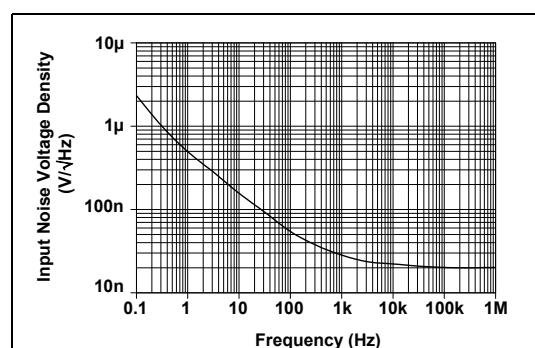


FIGURE 2-6: Input Noise Voltage Density vs. Frequency.

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100 \text{ k}\Omega$ to V_L , $C_L = 50 \text{ pF}$ and CS is tied low.

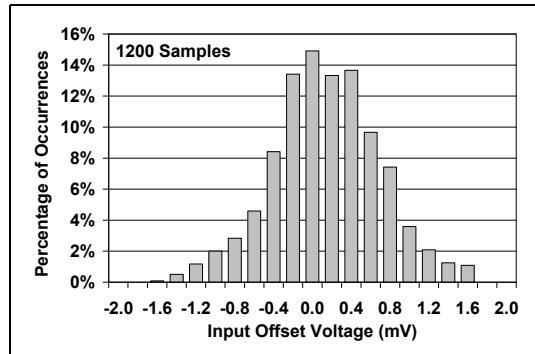


FIGURE 2-7: Input Offset Voltage.

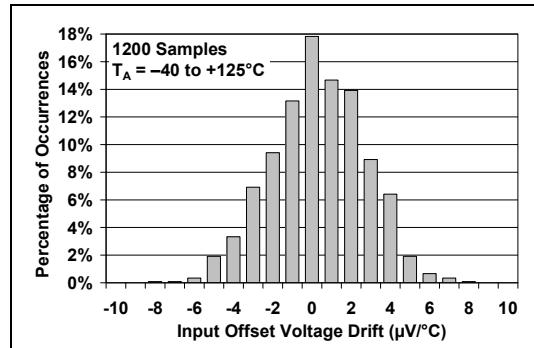


FIGURE 2-10: Input Offset Voltage Drift.

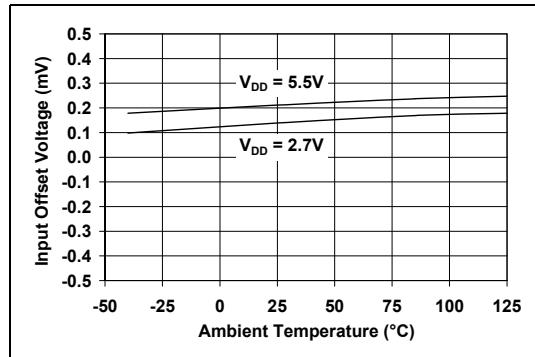


FIGURE 2-8: Input Offset Voltage vs. Temperature.

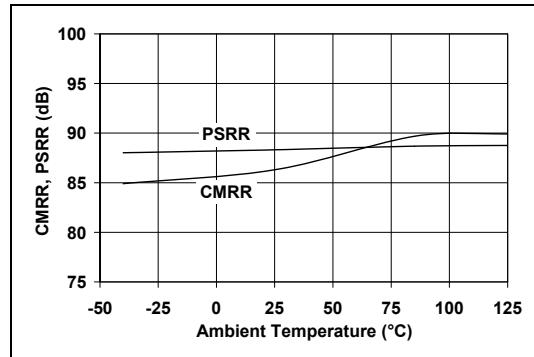


FIGURE 2-11: CMRR, PSRR vs. Temperature.

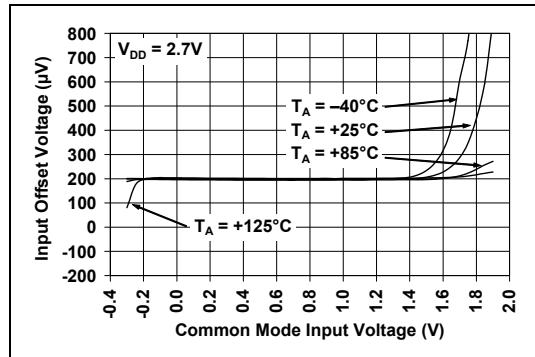


FIGURE 2-9: Input Offset Voltage vs. Common Mode Input Voltage with $V_{DD} = 2.7\text{V}$.

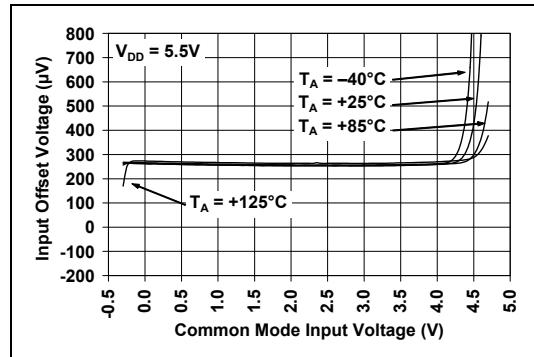


FIGURE 2-12: Input Offset Voltage vs. Common Mode Input Voltage with $V_{DD} = 5.5\text{V}$.

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100\text{ k}\Omega$ to V_L , $C_L = 50\text{ pF}$ and CS is tied low.

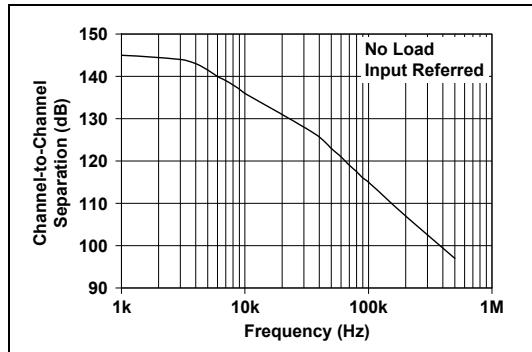


FIGURE 2-13: Channel-to-Channel Separation vs. Frequency.

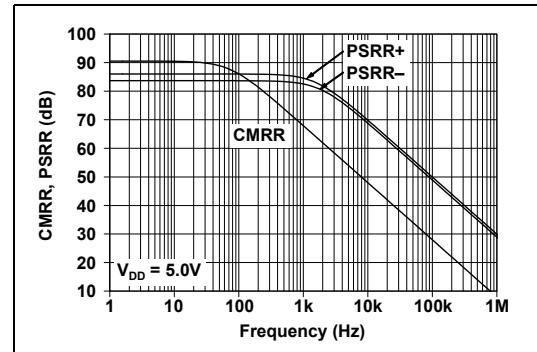


FIGURE 2-16: CMRR, PSRR vs. Frequency.

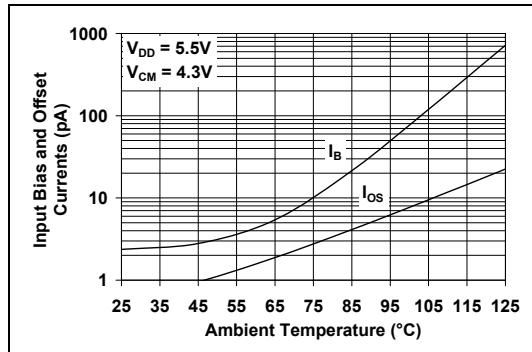


FIGURE 2-14: Input Bias Current, Input Offset Current vs. Ambient Temperature.

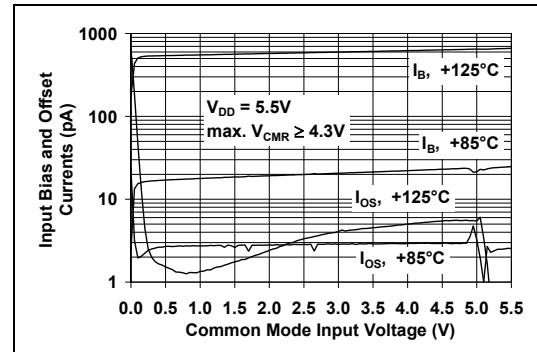


FIGURE 2-17: Input Bias Current, Input Offset Current vs. Common Mode Input Voltage.

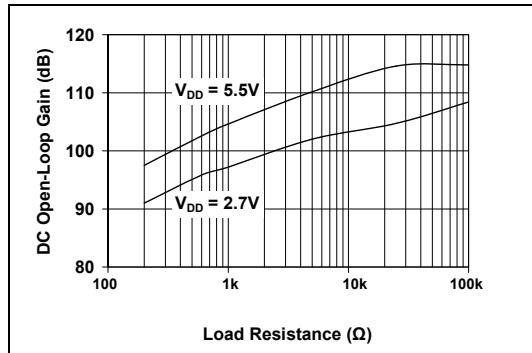


FIGURE 2-15: DC Open-Loop Gain vs. Load Resistance.

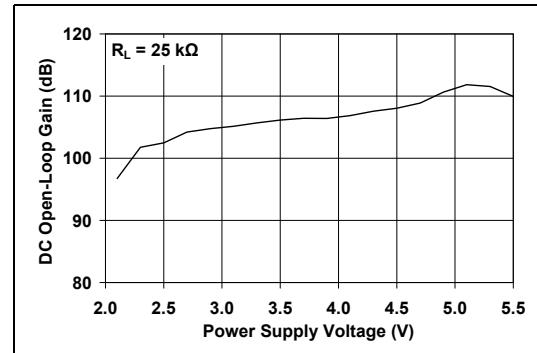


FIGURE 2-18: DC Open-Loop Gain vs. Supply Voltage.

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100\text{ k}\Omega$ to V_L , $C_L = 50\text{ pF}$ and CS is tied low.

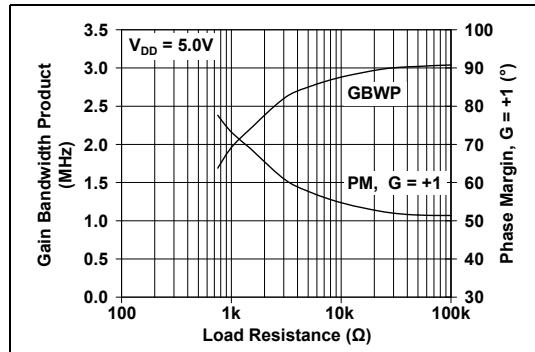


FIGURE 2-19: Gain Bandwidth Product, Phase Margin vs. Load Resistance.

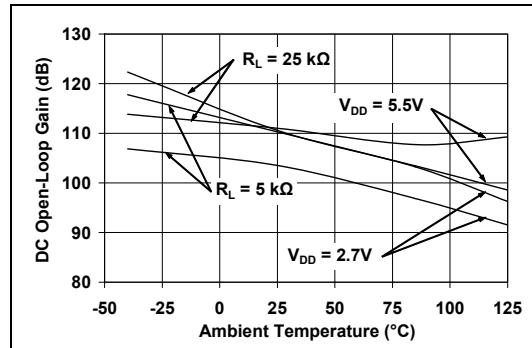


FIGURE 2-22: DC Open-Loop Gain vs. Temperature.

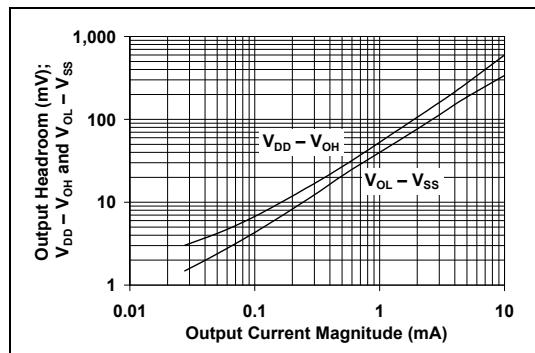


FIGURE 2-20: Output Voltage Headroom vs. Output Current.

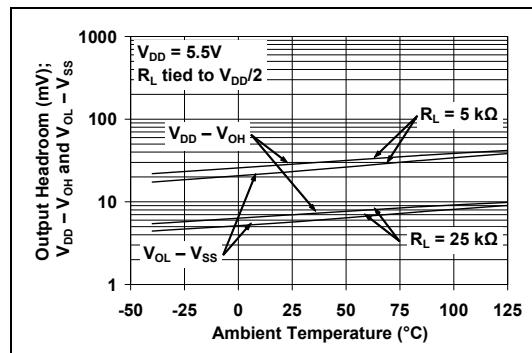


FIGURE 2-23: Output Voltage Headroom vs. Temperature.

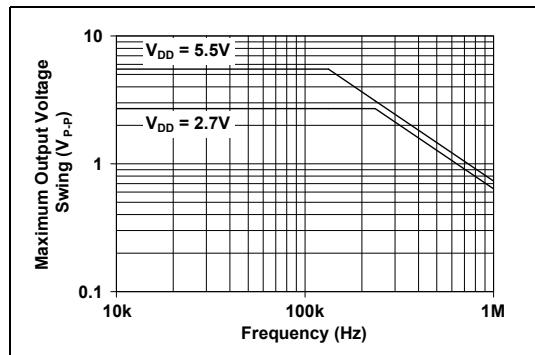


FIGURE 2-21: Maximum Output Voltage Swing vs. Frequency.

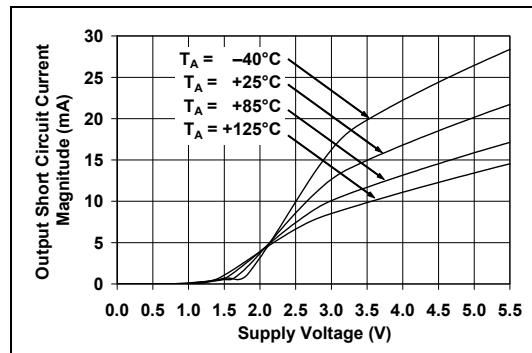


FIGURE 2-24: Output Short-Circuit Current vs. Supply Voltage.

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100 \text{ k}\Omega$ to V_L , $C_L = 50 \text{ pF}$ and CS is tied low.

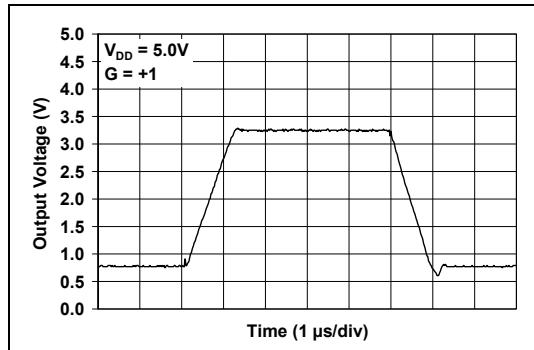


FIGURE 2-25: Large Signal Non-Inverting Pulse Response.

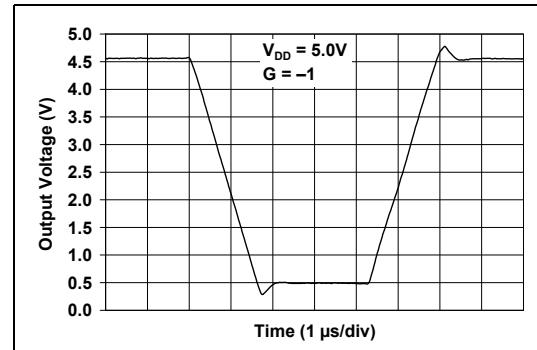


FIGURE 2-28: Large Signal Inverting Pulse Response.

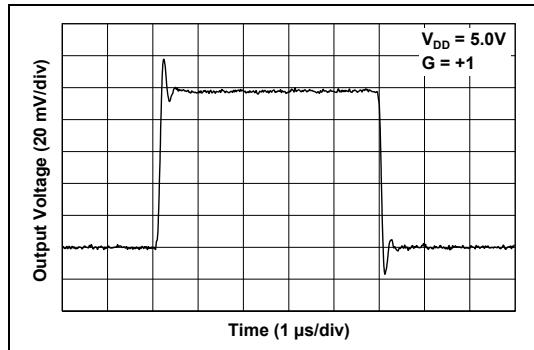


FIGURE 2-26: Small Signal Non-Inverting Pulse Response.

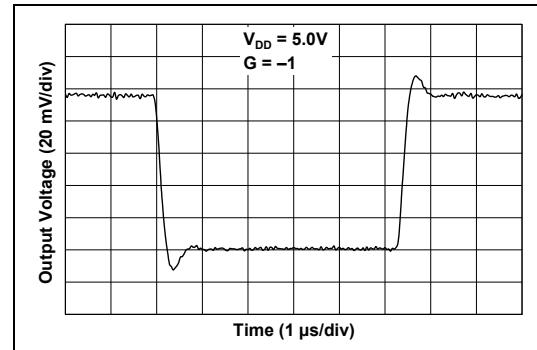


FIGURE 2-29: Small Signal Inverting Pulse Response.

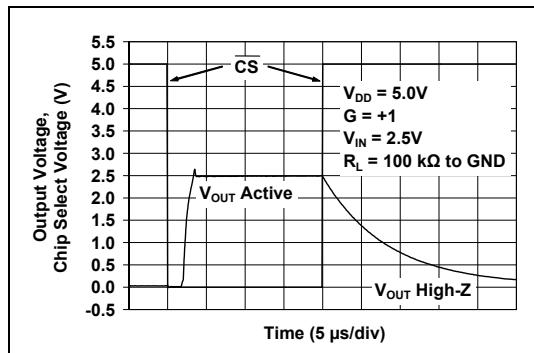


FIGURE 2-27: Chip Select Timing (MCP603).

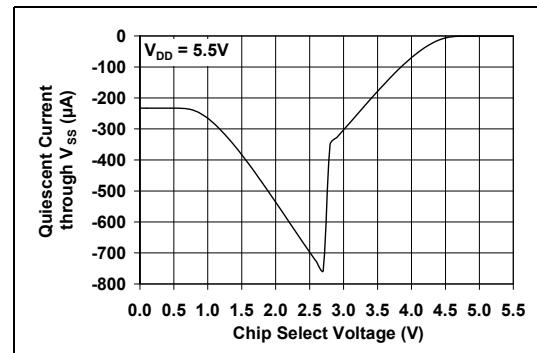


FIGURE 2-30: Quiescent Current Through V_{SS} vs. Chip Select Voltage (MCP603).

MCP601/1R/2/3/4

Note: Unless otherwise indicated, $T_A = +25^\circ\text{C}$, $V_{DD} = +2.7\text{V}$ to $+5.5\text{V}$, $V_{SS} = \text{GND}$, $V_{CM} = V_{DD}/2$, $V_{OUT} \approx V_{DD}/2$, $V_L = V_{DD}/2$, $R_L = 100 \text{ k}\Omega$ to V_L , $C_L = 50 \text{ pF}$ and CS is tied low.

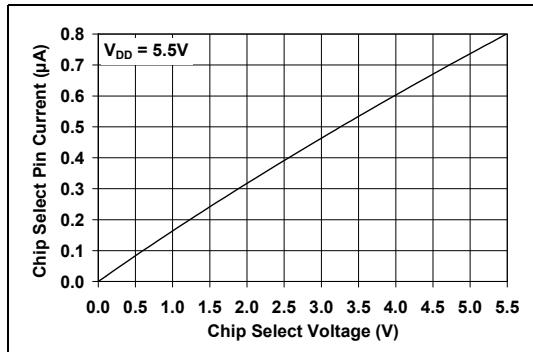


FIGURE 2-31: Chip Select Pin Input Current vs. Chip Select Voltage.

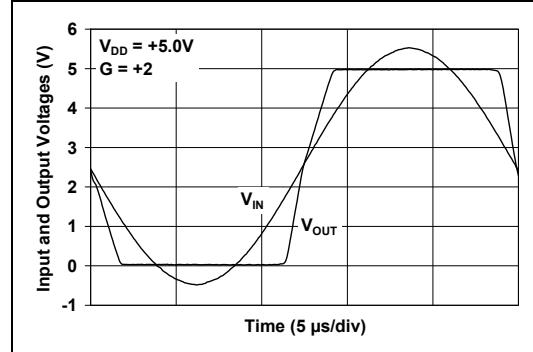


FIGURE 2-33: The MCP601/1R/2/3/4 family of op amps shows no phase reversal under input overdrive.

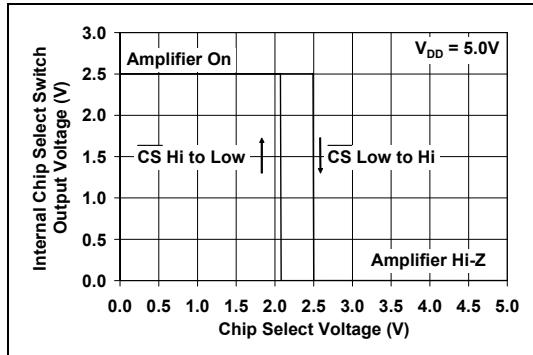


FIGURE 2-32: Hysteresis of Chip Select's Internal Switch.

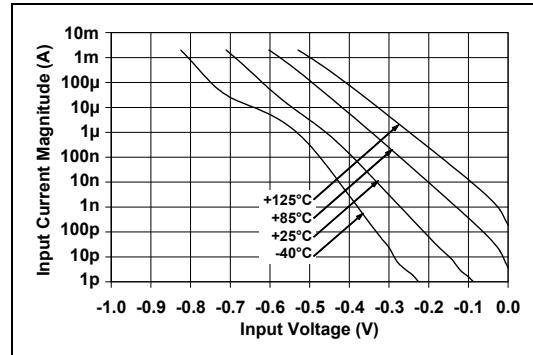


FIGURE 2-34: Measured Input Current vs. Input Voltage (below V_{SS}).

MCP601/1R/2/3/4

3.0 PIN DESCRIPTIONS

Descriptions of the pins are listed in [Table 3-1](#) (single op amps) and [Table 3-2](#) (dual and quad op amps).

TABLE 3-1: PIN FUNCTION TABLE FOR SINGLE OP AMPS

MCP601		MCP601R	MCP603		Symbol	Description
PDIP, SOIC, TSSOP	SOT-23-5	SOT-23-5 (Note 1)	SOT-23-6	PDIP, SOIC, TSSOP		
6	1	1	6	6	V_{OUT}	Analog Output
2	4	4	2	2	V_{IN^-}	Inverting Input
3	3	3	3	3	V_{IN^+}	Non-inverting Input
7	5	2	7	7	V_{DD}	Positive Power Supply
4	2	5	4	4	V_{SS}	Negative Power Supply
—	—	—	8	8	\bar{CS}	Chip Select
1, 5, 8	—	—	1, 5	1	NC	No Internal Connection

Note 1: The MCP601R is only available in the 5-pin SOT-23 package.

TABLE 3-2: PIN FUNCTION TABLE FOR DUAL AND QUAD OP AMPS

MCP602	MCP604	Symbol	Description
PDIP, SOIC, TSSOP	PDIP, SOIC, TSSOP		
1	1	V_{OUTA}	Analog Output (op amp A)
2	2	V_{INA^-}	Inverting Input (op amp A)
3	3	V_{INA^+}	Non-inverting Input (op amp A)
8	4	V_{DD}	Positive Power Supply
5	5	V_{INB^+}	Non-inverting Input (op amp B)
6	6	V_{INB^-}	Inverting Input (op amp B)
7	7	V_{OUTB}	Analog Output (op amp B)
—	8	V_{OUTC}	Analog Output (op amp C)
—	9	V_{INC^-}	Inverting Input (op amp C)
—	10	V_{INC^+}	Non-inverting Input (op amp C)
4	11	V_{SS}	Negative Power Supply
—	12	V_{IND^+}	Non-inverting Input (op amp D)
—	13	V_{IND^-}	Inverting Input (op amp D)
—	14	V_{OUTD}	Analog Output (op amp D)

3.1 Analog Outputs

The op amp output pins are low-impedance voltage sources.

3.2 Analog Inputs

The op amp non-inverting and inverting inputs are high-impedance CMOS inputs with low bias currents.

3.3 Chip Select Digital Input

This is a CMOS, Schmitt-triggered input that places the part into a low power mode of operation.

3.4 Power Supply Pins

The positive power supply pin (V_{DD}) is 2.5V to 6.0V higher than the negative power supply pin (V_{SS}). For normal operation, the other pins are at voltages between V_{SS} and V_{DD} .

Typically, these parts are used in a single (positive) supply configuration. In this case, V_{SS} is connected to ground and V_{DD} is connected to the supply. V_{DD} will need bypass capacitors.

MCP601/1R/2/3/4

4.0 APPLICATIONS INFORMATION

The MCP601/1R/2/3/4 family of op amps are fabricated on Microchip's state-of-the-art CMOS process. They are unity-gain stable and suitable for a wide range of general purpose applications.

4.1 Inputs

4.1.1 PHASE REVERSAL

The MCP601/1R/2/3/4 op amp is designed to prevent phase reversal when the input pins exceed the supply voltages. [Figure 2-34](#) shows the input voltage exceeding the supply voltage without any phase reversal.

4.1.2 INPUT VOLTAGE AND CURRENT LIMITS

The ESD protection on the inputs can be depicted as shown in [Figure 4-1](#). This structure was chosen to protect the input transistors, and to minimize input bias current (I_B). The input ESD diodes clamp the inputs when they try to go more than one diode drop below V_{SS} . They also clamp any voltages that go too far above V_{DD} ; their breakdown voltage is high enough to allow normal operation, and low enough to bypass quick ESD events within the specified limits.

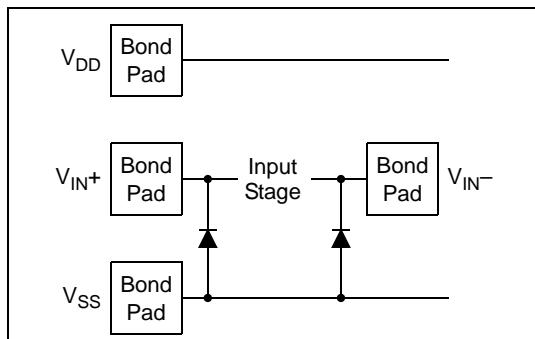


FIGURE 4-1: Simplified Analog Input ESD Structures.

In order to prevent damage and/or improper operation of these op amps, the circuit they are in must limit the currents and voltages at the V_{IN+} and V_{IN-} pins (see **Absolute Maximum Ratings** † at the beginning of [Section 1.0 “Electrical Characteristics”](#)). [Figure 4-2](#) shows the recommended approach to protecting these inputs. The internal ESD diodes prevent the input pins (V_{IN+} and V_{IN-}) from going too far below ground, and the resistors R_1 and R_2 limit the possible current drawn out of the input pins. Diodes D_1 and D_2 prevent the input pins (V_{IN+} and V_{IN-}) from going too far above V_{DD} , and dump any currents onto V_{DD} . When implemented as shown, resistors R_1 and R_2 also limit the current through D_1 and D_2 .

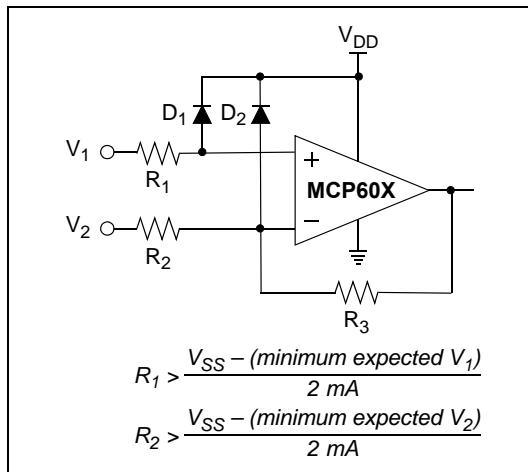


FIGURE 4-2: Protecting the Analog Inputs.

It is also possible to connect the diodes to the left of resistors R_1 and R_2 . In this case, current through the diodes D_1 and D_2 needs to be limited by some other mechanism. The resistors then serve as in-rush current limiters; the DC current into the input pins (V_{IN+} and V_{IN-}) should be very small.

A significant amount of current can flow out of the inputs when the common mode voltage (V_{CM}) is below ground (V_{SS}); see [Figure 2-34](#). Applications that are high impedance may need to limit the useable voltage range.

4.1.3 NORMAL OPERATION

The Common Mode Input Voltage Range (V_{CMR}) includes ground in single-supply systems (V_{SS}), but does not include V_{DD} . This means that the amplifier input behaves linearly as long as the Common Mode Input Voltage (V_{CM}) is kept within the specified V_{CMR} limits ($V_{SS}-0.3\text{V}$ to $V_{DD}-1.2\text{V}$ at $+25^\circ\text{C}$).

[Figure 4-3](#) shows a unity gain buffer. Since V_{OUT} is the same voltage as the inverting input, V_{OUT} must be kept below $V_{DD}-1.2\text{V}$ for correct operation.

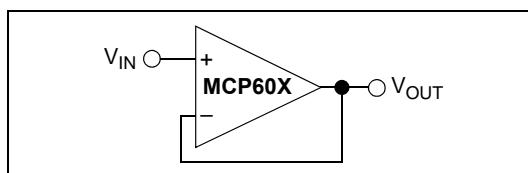


FIGURE 4-3: Unity Gain Buffer has a Limited V_{OUT} Range.

MCP601/1R/2/3/4

4.2 Rail-to-Rail Output

There are two specifications that describe the output swing capability of the MCP601/1R/2/3/4 family of op amps. The first specification (Maximum Output Voltage Swing) defines the absolute maximum swing that can be achieved under the specified load conditions. For instance, the output voltage swings to within 15 mV of the negative rail with a 25 k Ω load to $V_{DD}/2$. [Figure 2-33](#) shows how the output voltage is limited when the input goes beyond the linear region of operation.

The second specification that describes the output swing capability of these amplifiers is the Linear Output Voltage Swing. This specification defines the maximum output swing that can be achieved while the amplifier is still operating in its linear region. To verify linear operation in this range, the large signal (DC Open-Loop Gain (A_{OL})) is measured at points 100 mV inside the supply rails. The measurement must exceed the specified gains in the specification table.

4.3 MCP603 Chip Select

The MCP603 is a single amplifier with Chip Select (CS). When CS is pulled high, the supply current drops to -0.7 μ A (typ.), which is pulled through the CS pin to V_{SS} . When this happens, the amplifier output is put into a high-impedance state. Pulling CS low enables the amplifier.

The \overline{CS} pin has an internal 5 M Ω (typical) pull-down resistor connected to V_{SS} , so it will go low if the \overline{CS} pin is left floating. [Figure 1-1](#) is the Chip Select timing diagram and shows the output voltage, supply currents, and CS current in response to a CS pulse. [Figure 2-27](#) shows the measured output voltage response to a CS pulse.

4.4 Capacitive Loads

Driving large capacitive loads can cause stability problems for voltage feedback op amps. As the load capacitance increases, the feedback loop's phase margin decreases and the closed-loop bandwidth is reduced. This produces gain peaking in the frequency response with overshoot and ringing in the step response.

When driving large capacitive loads with these op amps (e.g., > 40 pF when $G = +1$), a small series resistor at the output (R_{ISO} in [Figure 4-4](#)) improves the feedback loop's phase margin (stability) by making the output load resistive at higher frequencies. The bandwidth will be generally lower than the bandwidth with no capacitive load.

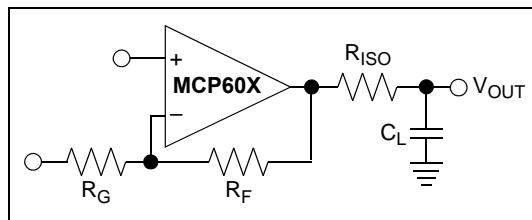


FIGURE 4-4: Output resistor R_{ISO} stabilizes large capacitive loads.

[Figure 4-5](#) gives recommended R_{ISO} values for different capacitive loads and gains. The x-axis is the normalized load capacitance (C_L/G_N) in order to make it easier to interpret the plot for arbitrary gains. G_N is the circuit's noise gain. For non-inverting gains, G_N and the gain are equal. For inverting gains, $G_N = 1 + |Gain|$ (e.g., -1 V/V gives $G_N = +2$ V/V).

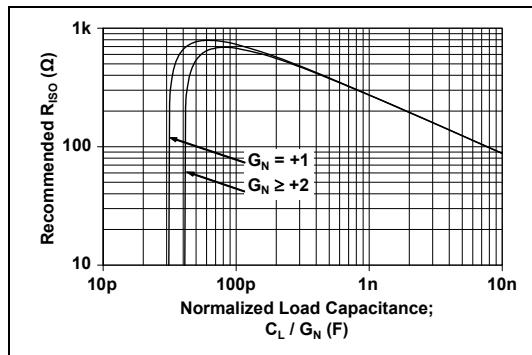


FIGURE 4-5: Recommended R_{ISO} values for capacitive loads.

Once you have selected R_{ISO} for your circuit, double-check the resulting frequency response peaking and step response overshoot in your circuit. Evaluation on the bench and simulations with the MCP601/1R/2/3/4 SPICE macro model are very helpful. Modify R_{ISO} 's value until the response is reasonable.

4.5 Supply Bypass

With this family of op amps, the power supply pin (V_{DD} for single-supply) should have a local bypass capacitor (i.e., 0.01 μ F to 0.1 μ F) within 2 mm for good high-frequency performance. It also needs a bulk capacitor (i.e., 1 μ F or larger) within 100 mm to provide large, slow currents. This bulk capacitor can be shared with nearby analog parts.

MCP601/1R/2/3/4

4.6 Unused Op Amps

An unused op amp in a quad package (MCP604) should be configured as shown in [Figure 4-6](#). These circuits prevent the output from toggling and causing crosstalk. Circuits A sets the op amp at its minimum noise gain. The resistor divider produces any desired reference voltage within the output voltage range of the op amp; the op amp buffers that reference voltage. Circuit B uses the minimum number of components and operates as a comparator, but it may draw more current.

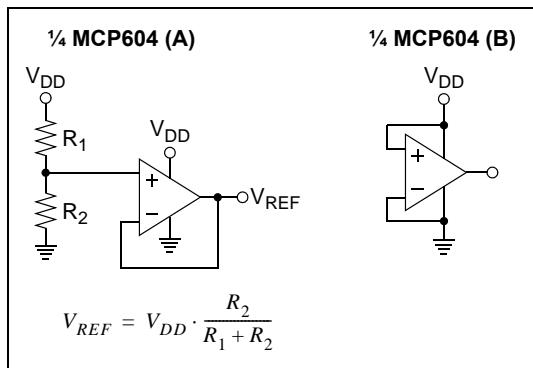


FIGURE 4-6: Unused Op Amps.

4.7 PCB Surface Leakage

In applications where low input bias current is critical, printed circuit board (PCB) surface leakage effects need to be considered. Surface leakage is caused by humidity, dust or other contamination on the board. Under low humidity conditions, a typical resistance between nearby traces is $10^{12}\Omega$. A 5V difference would cause 5 pA of current to flow. This is greater than the MCP601/1R/2/3/4 family's bias current at $+25^\circ\text{C}$ (1 pA, typical).

The easiest way to reduce surface leakage is to use a guard ring around sensitive pins (or traces). The guard ring is biased at the same voltage as the sensitive pin. An example of this type of layout is shown in [Figure 4-7](#).

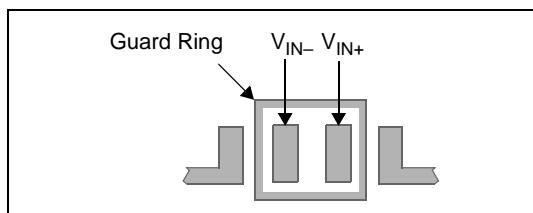


FIGURE 4-7: Example Guard Ring layout.

1. Connect the guard ring to the inverting input pin (V_{IN-}) for non-inverting gain amplifiers, including unity-gain buffers. This biases the guard ring to the common mode input voltage.

2. Connect the guard ring to the non-inverting input pin (V_{IN+}) for inverting gain amplifiers and transimpedance amplifiers (converts current to voltage, such as photo detectors). This biases the guard ring to the same reference voltage as the op amp (e.g., $V_{DD}/2$ or ground).

4.8 Typical Applications

4.8.1 ANALOG FILTERS

[Figure 4-8](#) and [Figure 4-9](#) show low-pass, second-order, Butterworth filters with a cutoff frequency of 10 Hz. The filter in [Figure 4-8](#) has a non-inverting gain of $+1 \text{ V/V}$, and the filter in [Figure 4-9](#) has an inverting gain of -1 V/V .

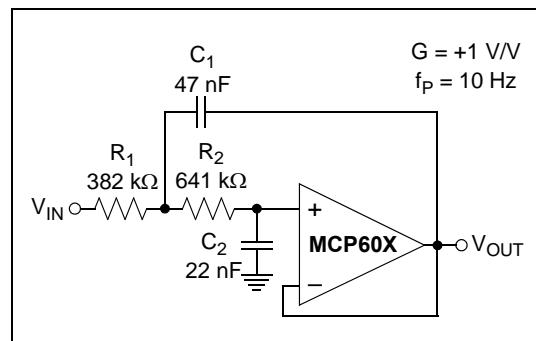


FIGURE 4-8: Second-Order, Low-Pass Sallen-Key Filter.

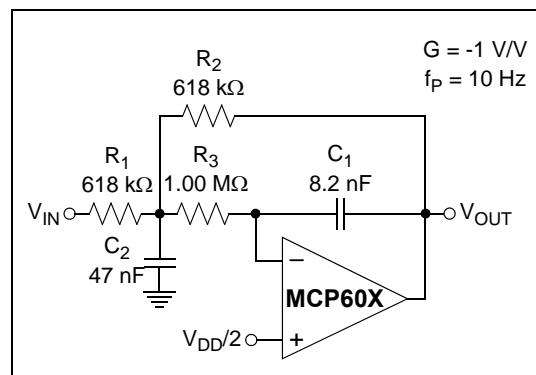


FIGURE 4-9: Second-Order, Low-Pass Multiple-Feedback Filter.

The MCP601/1R/2/3/4 family of op amps have low input bias current, which allows the designer to select larger resistor values and smaller capacitor values for these filters. This helps produce a compact PCB layout. These filters, and others, can be designed using Microchip's Design Aids; see [Section 5.2 "FilterLab® Software"](#) and [Section 5.3 "Mindi™ Simulator Tool"](#).

MCP601/1R/2/3/4

4.8.2 INSTRUMENTATION AMPLIFIER CIRCUITS

Instrumentation amplifiers have a differential input that subtracts one input voltage from another and rejects common mode signals. These amplifiers also provide a single-ended output voltage.

The three-op amp instrumentation amplifier is illustrated in [Figure 4-10](#). One advantage of this approach is unity-gain operation, while one disadvantage is that the common mode input range is reduced as R_2/R_G gets larger.

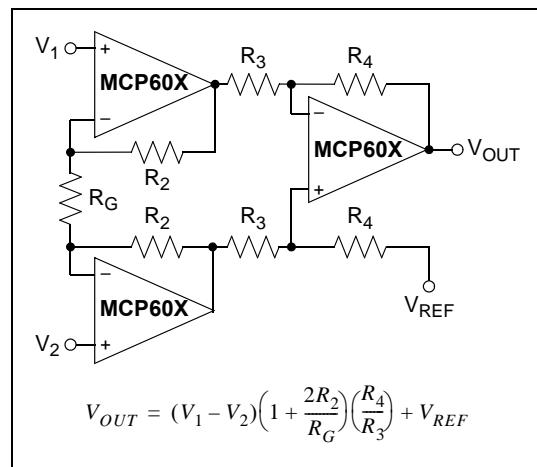


FIGURE 4-10: Three-Op Amp Instrumentation Amplifier.

The two-op amp instrumentation amplifier is shown in [Figure 4-11](#). While its power consumption is lower than the three-op amp version, its main drawbacks are that the common mode range is reduced with higher gains and it must be configured in gains of two or higher.

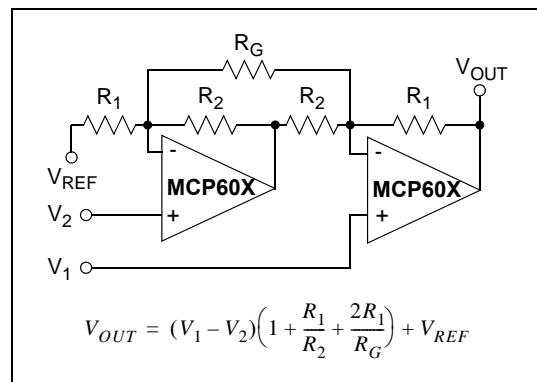


FIGURE 4-11: Two-Op Amp Instrumentation Amplifier.

Both instrumentation amplifiers should use a bulk bypass capacitor of at least 1 μF . The CMRR of these amplifiers will be set by both the op amp CMRR and resistor matching.

4.8.3 PHOTO DETECTION

The MCP601/1R/2/3/4 op amps can be used to easily convert the signal from a sensor that produces an output current (such as a photo diode) into a voltage (a transimpedance amplifier). This is implemented with a single resistor (R_2) in the feedback loop of the amplifiers shown in [Figure 4-12](#) and [Figure 4-13](#). The optional capacitor (C_2) sometimes provides stability for these circuits.

A photodiode configured in the Photovoltaic mode has zero voltage potential placed across it ([Figure 4-12](#)). In this mode, the light sensitivity and linearity is maximized, making it best suited for precision applications. The key amplifier specifications for this application are: low input bias current, low noise, common mode input voltage range (including ground), and rail-to-rail output.

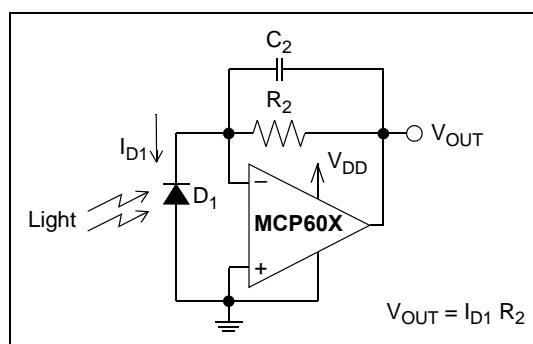


FIGURE 4-12: Photovoltaic Mode Detector.

In contrast, a photodiode that is configured in the Photoconductive mode has a reverse bias voltage across the photo-sensing element ([Figure 4-13](#)). This decreases the diode capacitance, which facilitates high-speed operation (e.g., high-speed digital communications). The design trade-off is increased diode leakage current and linearity errors. The op amp needs to have a wide Gain Bandwidth Product (GBWP).

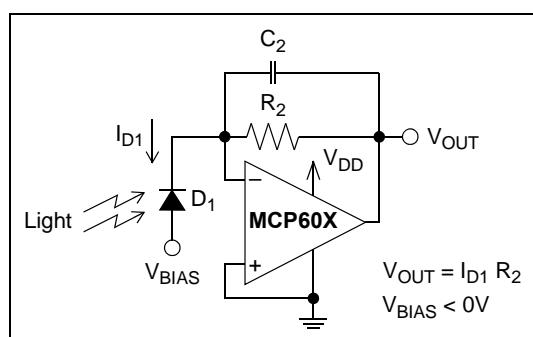


FIGURE 4-13: Photoconductive Mode Detector.

MCP601/1R/2/3/4

5.0 DESIGN AIDS

Microchip provides the basic design tools needed for the MCP601/1R/2/3/4 family of op amps.

5.1 SPICE Macro Model

The latest SPICE macro model for the MCP601/1R/2/3/4 op amps is available on the Microchip web site at www.microchip.com. This model is intended to be an initial design tool that works well in the op amp's linear region of operation over the temperature range. See the model file for information on its capabilities.

Bench testing is a very important part of any design and cannot be replaced with simulations. Also, simulation results using this macro model need to be validated by comparing them to the data sheet specifications and characteristic curves.

5.2 FilterLab® Software

Microchip's FilterLab® software is an innovative software tool that simplifies analog active filter (using op amps) design. Available at no cost from the Microchip web site at www.microchip.com/filterlab, the FilterLab design tool provides full schematic diagrams of the filter circuit with component values. It also outputs the filter circuit in SPICE format, which can be used with the macro model to simulate actual filter performance.

5.3 Mindi™ Simulator Tool

Microchip's Mindi™ simulator tool aids in the design of various circuits useful for active filter, amplifier and power-management applications. It is a free online simulation tool available from the Microchip web site at www.microchip.com/mindi. This interactive simulator enables designers to quickly generate circuit diagrams, simulate circuits. Circuits developed using the Mindi simulation tool can be downloaded to a personal computer or workstation.

5.4 MAPS (Microchip Advanced Part Selector)

MAPS is a software tool that helps semiconductor professionals efficiently identify Microchip devices that fit a particular design requirement. Available at no cost from the Microchip website at www.microchip.com/maps, the MAPS is an overall selection tool for Microchip's product portfolio that includes Analog, Memory, MCUs and DSCs. Using this tool you can define a filter to sort features for a parametric search of devices and export side-by-side technical comparison reports. Helpful links are also provided for Datasheets, Purchase, and Sampling of Microchip parts.

5.5 Analog Demonstration and Evaluation Boards

Microchip offers a broad spectrum of Analog Demonstration and Evaluation Boards that are designed to help you achieve faster time to market. For a complete listing of these boards and their corresponding user's guides and technical information, visit the Microchip web site at www.microchip.com/analogtools.

Two of our boards that are especially useful are:

- **P/N SOIC8EV: 8-Pin SOIC/MSOP/TSSOP/DIP Evaluation Board**
- **P/N SOIC14EV: 14-Pin SOIC/TSSOP/DIP Evaluation Board**

5.6 Application Notes

The following Microchip Application Notes are available on the Microchip web site at www.microchip.com/appnotes and are recommended as supplemental reference resources.

ADN003: "Select the Right Operational Amplifier for your Filtering Circuits", DS21821

AN722: "Operational Amplifier Topologies and DC Specifications", DS00722

AN723: "Operational Amplifier AC Specifications and Applications", DS00723

AN884: "Driving Capacitive Loads With Op Amps", DS00884

AN990: "Analog Sensor Conditioning Circuits – An Overview", DS00990

These application notes and others are listed in the design guide:

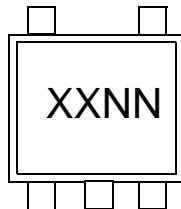
"Signal Chain Design Guide", DS21825

MCP601/1R/2/3/4

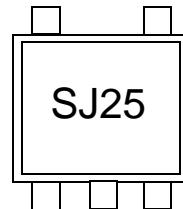
6.0 PACKAGING INFORMATION

6.1 Package Marking Information

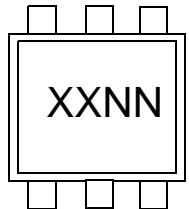
5-Lead SOT-23 (MCP601 and MCP601R only)



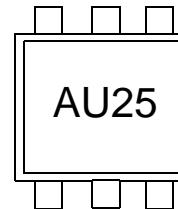
Example:



6-Lead SOT-23 (MCP603 only)



Example:



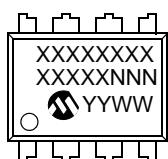
Legend:	XX...X	Customer-specific information
	Y	Year code (last digit of calendar year)
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week '01')
	NNN	Alphanumeric traceability code
	(e3)	Pb-free JEDEC designator for Matte Tin (Sn)
*		This package is Pb-free. The Pb-free JEDEC designator (e3) can be found on the outer packaging for this package.

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line, thus limiting the number of available characters for customer-specific information.

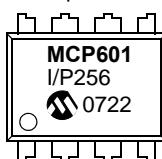
MCP601/1R/2/3/4

Package Marking Information (Continued)

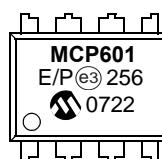
8-Lead PDIP (300 mil)



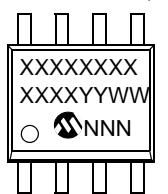
Example:



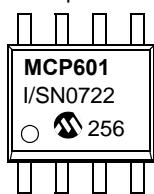
OR



8-Lead SOIC (150 mil)



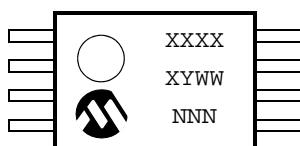
Example:



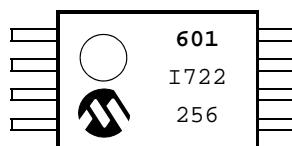
OR

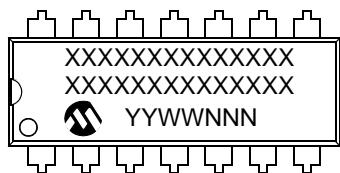


8-Lead TSSOP

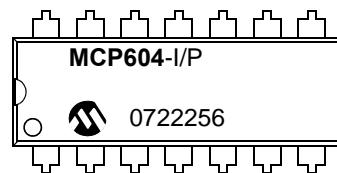


Example:

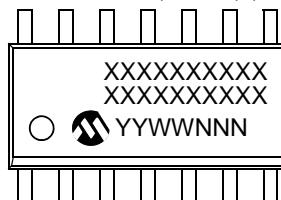


MCP601/1R/2/3/4**Package Marking Information (Continued)**14-Lead PDIP (300 mil) (**MCP604**)

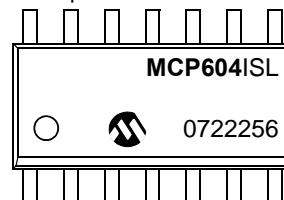
Example:



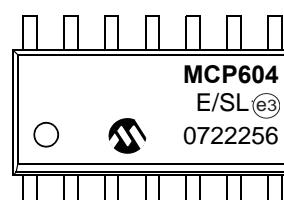
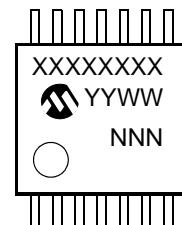
OR

14-Lead SOIC (150 mil) (**MCP604**)

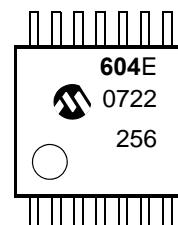
Example:



OR

14-Lead TSSOP (**MCP604**)

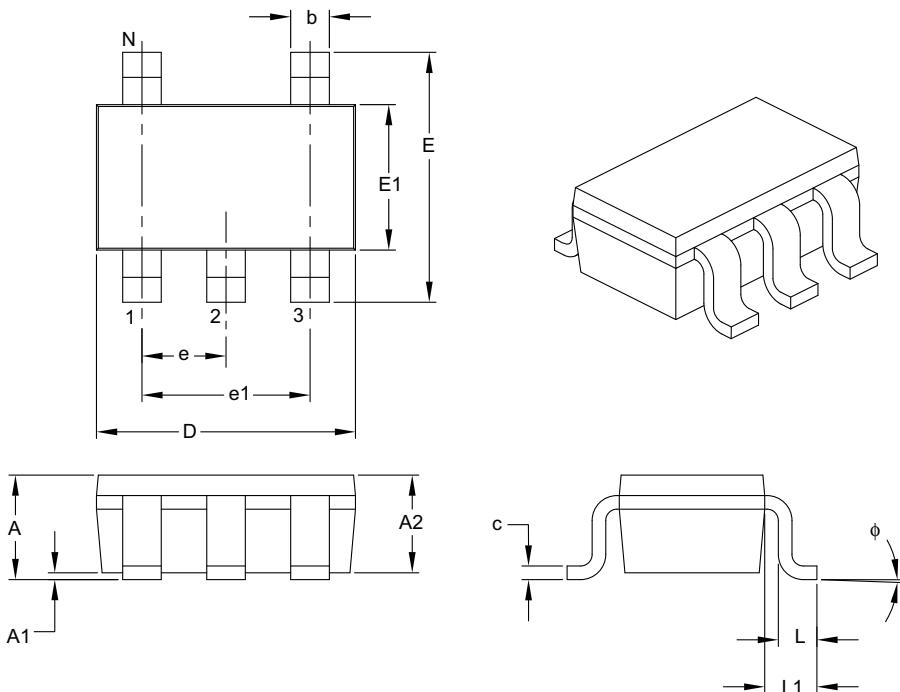
Example:



MCP601/1R/2/3/4

5-Lead Plastic Small Outline Transistor (OT) [SOT-23]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N	5		
Lead Pitch	e	0.95	BSC	
Outside Lead Pitch	e1	1.90	BSC	
Overall Height	A	0.90	—	1.45
Molded Package Thickness	A2	0.89	—	1.30
Standoff	A1	0.00	—	0.15
Overall Width	E	2.20	—	3.20
Molded Package Width	E1	1.30	—	1.80
Overall Length	D	2.70	—	3.10
Foot Length	L	0.10	—	0.60
Footprint	L1	0.35	—	0.80
Foot Angle	ϕ	0°	—	30°
Lead Thickness	c	0.08	—	0.26
Lead Width	b	0.20	—	0.51

Notes:

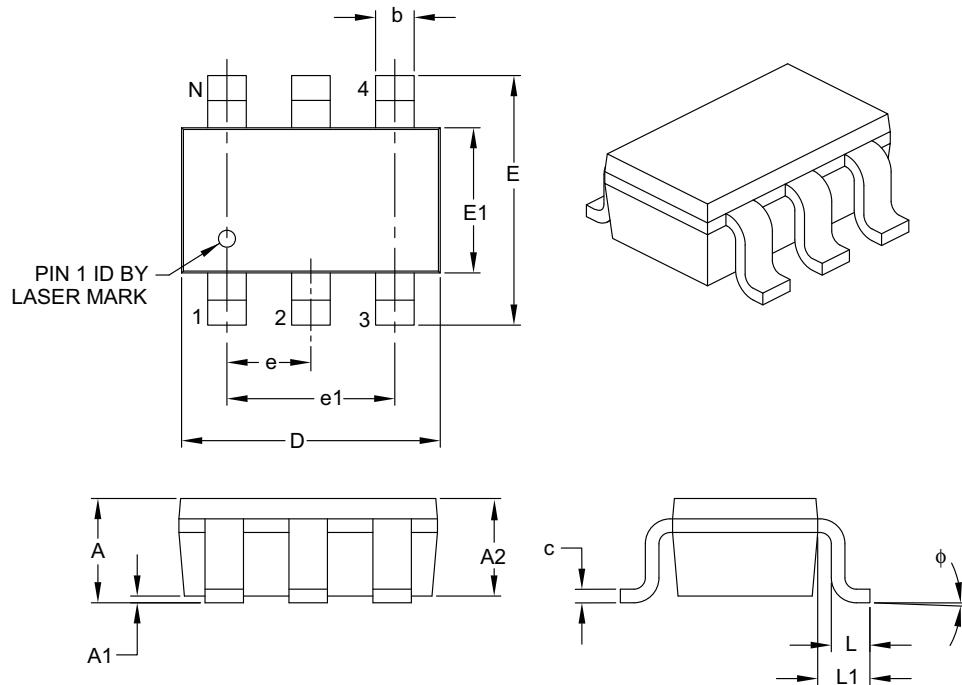
- Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.127 mm per side.
- Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-091B

MCP601/1R/2/3/4**6-Lead Plastic Small Outline Transistor (CH) [SOT-23]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N	6		
Pitch	e	0.95	BSC	
Outside Lead Pitch	e1	1.90	BSC	
Overall Height	A	0.90	—	1.45
Molded Package Thickness	A2	0.89	—	1.30
Standoff	A1	0.00	—	0.15
Overall Width	E	2.20	—	3.20
Molded Package Width	E1	1.30	—	1.80
Overall Length	D	2.70	—	3.10
Foot Length	L	0.10	—	0.60
Footprint	L1	0.35	—	0.80
Foot Angle	ϕ	0°	—	30°
Lead Thickness	c	0.08	—	0.26
Lead Width	b	0.20	—	0.51

Notes:

- Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.127 mm per side.
- Dimensioning and tolerancing per ASME Y14.5M.

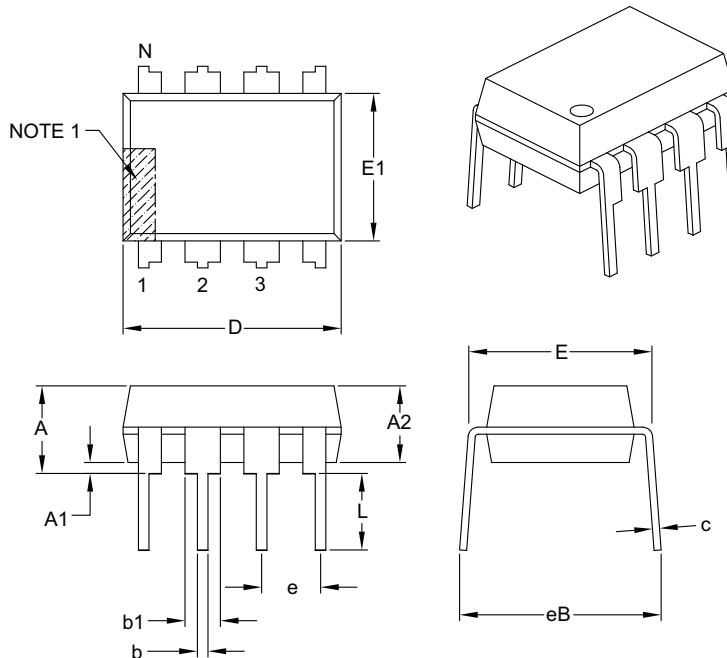
BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-028B

MCP601/1R/2/3/4

8-Lead Plastic Dual In-Line (P) – 300 mil Body [PDIP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Units		INCHES		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		8	
Pitch	e		.100 BSC	
Top to Seating Plane	A	–	–	.210
Molded Package Thickness	A2	.115	.130	.195
Base to Seating Plane	A1	.015	–	–
Shoulder to Shoulder Width	E	.290	.310	.325
Molded Package Width	E1	.240	.250	.280
Overall Length	D	.348	.365	.400
Tip to Seating Plane	L	.115	.130	.150
Lead Thickness	c	.008	.010	.015
Upper Lead Width	b1	.040	.060	.070
Lower Lead Width	b	.014	.018	.022
Overall Row Spacing §	eB	–	–	.430

Notes:

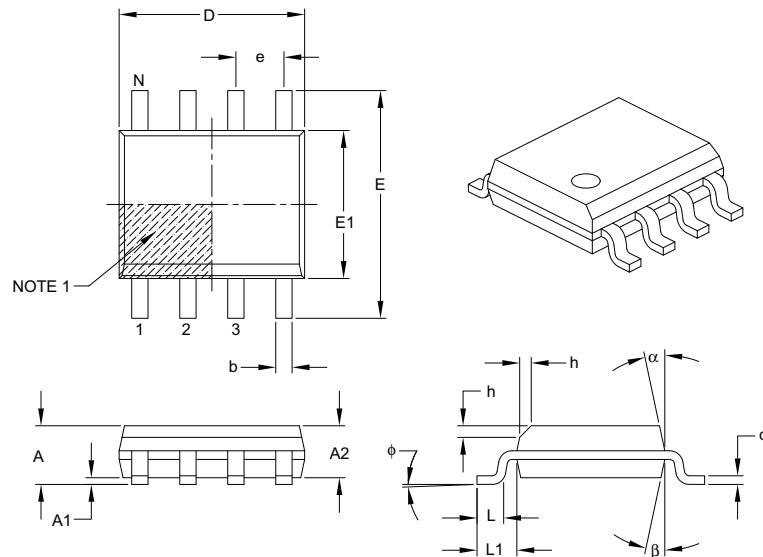
1. Pin 1 visual index feature may vary, but must be located with the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-018B

MCP601/1R/2/3/4**8-Lead Plastic Small Outline (SN) – Narrow, 3.90 mm Body [SOIC]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension Limits	Units	MILLIMETERS		
		MIN	NOM	MAX
Number of Pins	N	8		
Pitch	e	1.27 BSC		
Overall Height	A	—	—	1.75
Molded Package Thickness	A2	1.25	—	—
Standoff §	A1	0.10	—	0.25
Overall Width	E	6.00 BSC		
Molded Package Width	E1	3.90 BSC		
Overall Length	D	4.90 BSC		
Chamfer (optional)	h	0.25	—	0.50
Foot Length	L	0.40	—	1.27
Footprint	L1	1.04 REF		
Foot Angle	ϕ	0°	—	8°
Lead Thickness	c	0.17	—	0.25
Lead Width	b	0.31	—	0.51
Mold Draft Angle Top	α	5°	—	15°
Mold Draft Angle Bottom	β	5°	—	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

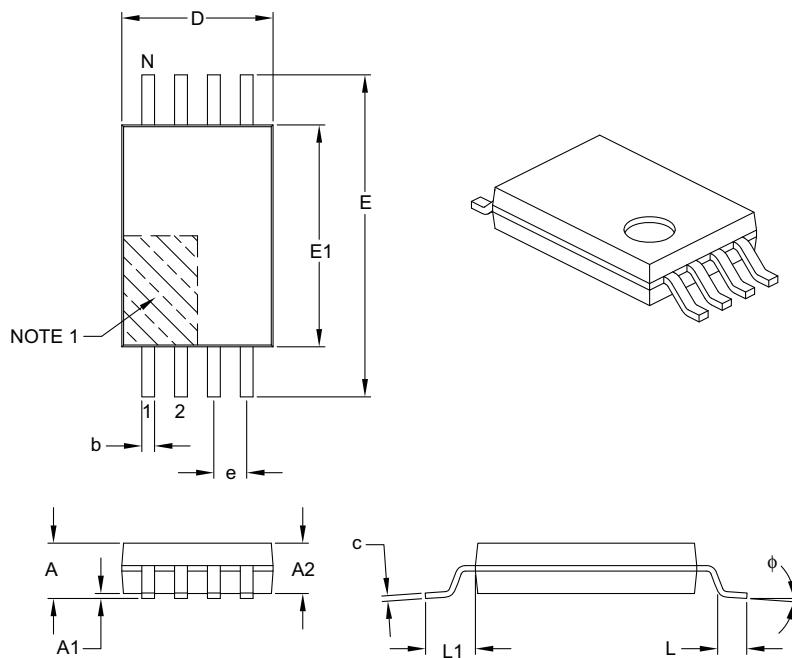
REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-057B

MCP601/1R/2/3/4

8-Lead Plastic Thin Shrink Small Outline (ST) – 4.4 mm Body [TSSOP]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



		Units	MILLIMETERS		
Dimension Limits			MIN	NOM	MAX
Number of Pins	N		8		
Pitch	e		0.65 BSC		
Overall Height	A		–	–	1.20
Molded Package Thickness	A2	0.80	1.00	1.05	
Standoff	A1	0.05	–	0.15	
Overall Width	E	6.40 BSC			
Molded Package Width	E1	4.30	4.40	4.50	
Molded Package Length	D	2.90	3.00	3.10	
Foot Length	L	0.45	0.60	0.75	
Footprint	L1	1.00 REF			
Foot Angle	φ	0°	–	8°	
Lead Thickness	c	0.09	–	0.20	
Lead Width	b	0.19	–	0.30	

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
3. Dimensioning and tolerancing per ASME Y14.5M.

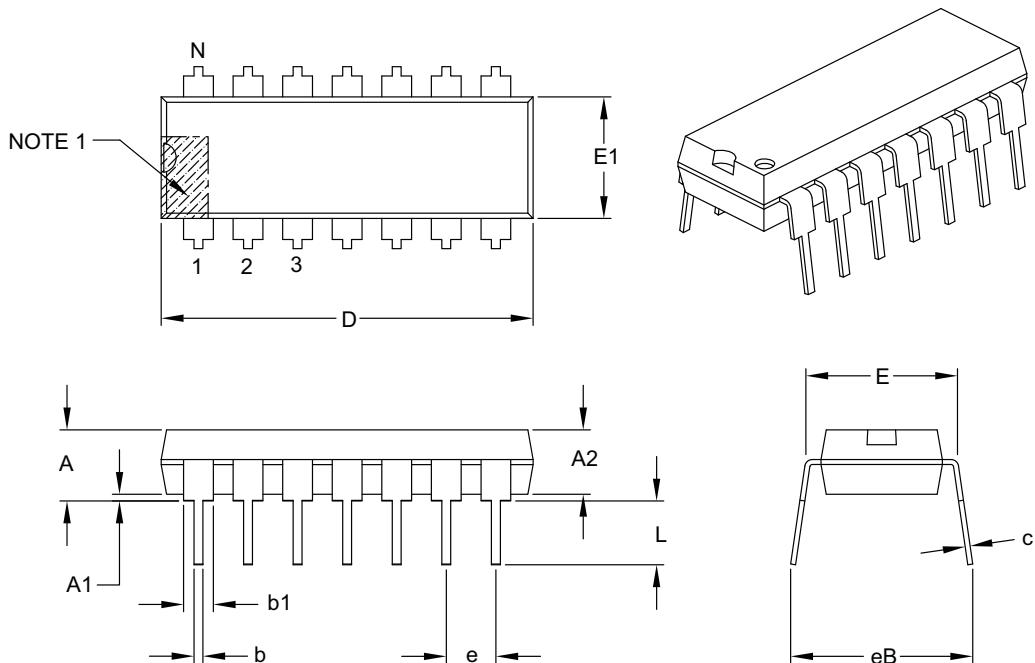
BSC: Basic Dimension. Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-086B

MCP601/1R/2/3/4**14-Lead Plastic Dual In-Line (P) – 300 mil Body [PDIP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Units		INCHES		
Dimension Limits		MIN	NOM	MAX
Number of Pins	N		14	
Pitch	e		.100 BSC	
Top to Seating Plane	A	—	—	.210
Molded Package Thickness	A2	.115	.130	.195
Base to Seating Plane	A1	.015	—	—
Shoulder to Shoulder Width	E	.290	.310	.325
Molded Package Width	E1	.240	.250	.280
Overall Length	D	.735	.750	.775
Tip to Seating Plane	L	.115	.130	.150
Lead Thickness	c	.008	.010	.015
Upper Lead Width	b1	.045	.060	.070
Lower Lead Width	b	.014	.018	.022
Overall Row Spacing §	eB	—	—	.430

Notes:

1. Pin 1 visual index feature may vary, but must be located with the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

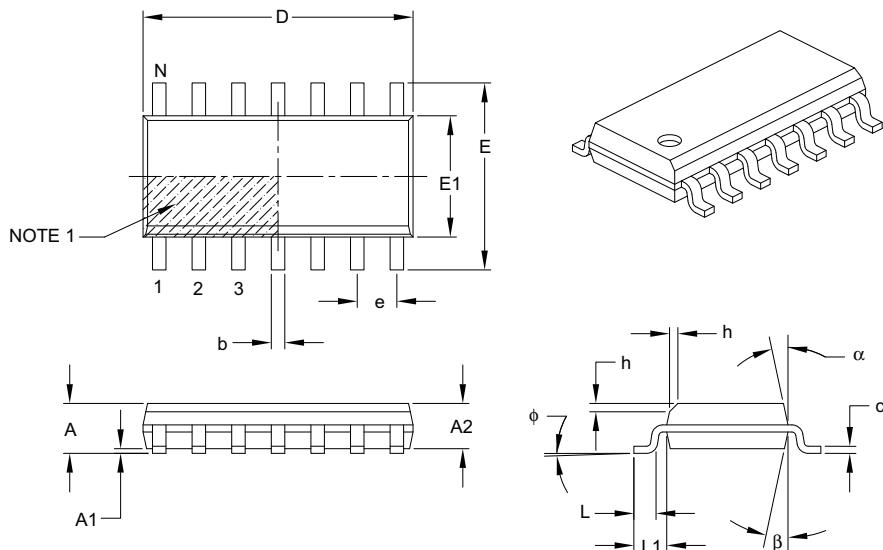
BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing C04-005B

MCP601/1R/2/3/4

14-Lead Plastic Small Outline (SL) – Narrow, 3.90 mm Body [SOIC]

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



	Units	MILLIMETERS		
	Dimension Limits	MIN	NOM	MAX
Number of Pins	N	14		
Pitch	e	1.27 BSC		
Overall Height	A	—	—	1.75
Molded Package Thickness	A2	1.25	—	—
Standoff §	A1	0.10	—	0.25
Overall Width	E	6.00 BSC		
Molded Package Width	E1	3.90 BSC		
Overall Length	D	8.65 BSC		
Chamfer (optional)	h	0.25	—	0.50
Foot Length	L	0.40	—	1.27
Footprint	L1	1.04 REF		
Foot Angle	ϕ	0°	—	8°
Lead Thickness	c	0.17	—	0.25
Lead Width	b	0.31	—	0.51
Mold Draft Angle Top	α	5°	—	15°
Mold Draft Angle Bottom	β	5°	—	15°

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. § Significant Characteristic.
3. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
4. Dimensioning and tolerancing per ASME Y14.5M.

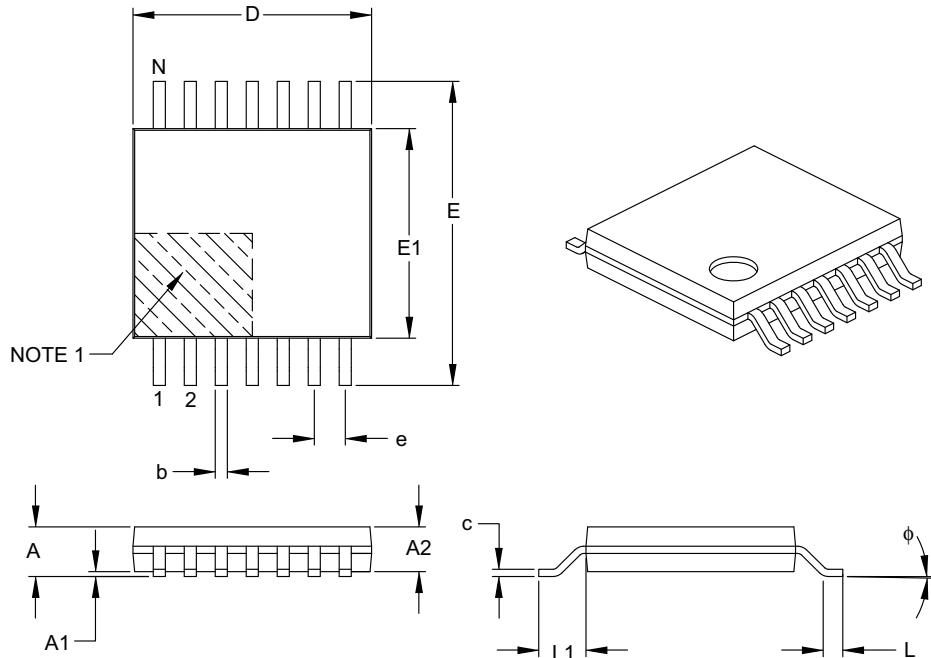
BSC: Basic Dimension. Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-065B

MCP601/1R/2/3/4**14-Lead Plastic Thin Shrink Small Outline (ST) – 4.4 mm Body [TSSOP]**

Note: For the most current package drawings, please see the Microchip Packaging Specification located at <http://www.microchip.com/packaging>



Dimension Limits	Units MILLIMETERS		
	MIN	NOM	MAX
Number of Pins	N	14	
Pitch	e	0.65 BSC	
Overall Height	A	–	1.20
Molded Package Thickness	A2	0.80	1.00
Standoff	A1	0.05	–
Overall Width	E	6.40 BSC	
Molded Package Width	E1	4.30	4.40
Molded Package Length	D	4.90	5.00
Foot Length	L	0.45	0.60
Footprint	L1	1.00 REF	
Foot Angle	ϕ	0°	–
Lead Thickness	c	0.09	–
Lead Width	b	0.19	0.30

Notes:

1. Pin 1 visual index feature may vary, but must be located within the hatched area.
2. Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.15 mm per side.
3. Dimensioning and tolerancing per ASME Y14.5M.

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

REF: Reference Dimension, usually without tolerance, for information purposes only.

Microchip Technology Drawing C04-087B

MCP601/1R/2/3/4

NOTES:

MCP601/1R/2/3/4

APPENDIX A: REVISION HISTORY

Revision G (December 2007)

- Updated Figure 2-15 and Figure 2-19.
- Updated Table 3-1 and Table 3-2.
- Updated notes to **Section 1.0 “Electrical Characteristics”**.
- Expanded Analog Input Absolute Maximum Voltage Range (applies retroactively).
- Expanded operating V_{DD} to a maximum of 6.0V.
- Added Figure 2-34.
- Added **Section 4.1.1 “Phase Reversal”**, **Section 4.1.2 “Input Voltage and Current Limits”**, and **Section 4.1.3 “Normal Operation”**.
- Corrected **Section 6.0 “Packaging Information”**.

Revision F (February 2004)

- Undocumented changes.

Revision E (September 2003)

- Undocumented changes.

Revision D (April 2000)

- Undocumented changes.

Revision C (July 1999)

- Undocumented changes.

Revision B (June 1999)

- Undocumented changes.

Revision A (March 1999)

- Original Release of this Document.

MCP601/1R/2/3/4

NOTES:

MCP601/1R/2/3/4**PRODUCT IDENTIFICATION SYSTEM**

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

<u>PART NO.</u>	<u>-X</u>	<u>/XX</u>	
Device	Temperature Range	Package	
Device			<p>MCP601 Single Op Amp MCP601T Single Op Amp (Tape and Reel for SOT-23, SOIC and TSSOP) MCP601RT Single Op Amp (Tape and Reel for SOT-23-5) MCP602 Dual Op Amp MCP602T Dual Op Amp (Tape and Reel for SOIC and TSSOP) MCP603 Single Op Amp with Chip Select MCP603T Single Op Amp with Chip Select (Tape and Reel for SOT-23, SOIC and TSSOP) MCP604 Quad Op Amp MCP604T Quad Op Amp (Tape and Reel for SOIC and TSSOP)</p>
Temperature Range	I = -40° C to +85° C E = -40° C to +125° C		
Package		OT = Plastic SOT-23, 5-lead (MCP601 only) CH = Plastic SOT-23, 6-lead (MCP603 only) P = Plastic DIP (300 mil body), 8, 14 lead SN = Plastic SOIC (3.90 mm body), 8 lead SL = Plastic SOIC (3.90 mm body), 14 lead ST = Plastic TSSOP (4.4 mm body), 8, 14 lead	
			Examples: a) MCP601-I/P: Single Op Amp, Industrial Temperature, 8 lead PDIP package. b) MCP601-E/SN: Single Op Amp, Extended Temperature, 8 lead SOIC package. c) MCP601T-E/ST: Tape and Reel, Extended Temperature, Single Op Amp, 8 lead TSSOP package d) MCP601RT-I/OT: Tape and Reel, Industrial Temperature, Single Op Amp, Rotated 5 lead SOT-23 package. e) MCP601RT-E/OT: Tape and Reel, Extended Temperature, Single Op Amp, Rotated, 5 lead SOT-23 package. a) MCP602-I/SN: Dual Op Amp, Industrial Temperature, 8 lead SOIC package. b) MCP602-E/P: Dual Op Amp, Extended Temperature, 8 lead PDIP package. c) MCP602T-E/ST: Tape and Reel, Extended Temperature, Dual Op Amp, 8 lead TSSOP package. a) MCP603-I/SN: Industrial Temperature, Single Op Amp with Chip Select, 8 lead SOIC package. b) MCP603-E/P: Extended Temperature, Single Op Amp with Chip Select, 8 lead PDIP package. c) MCP603T-E/ST: Tape and Reel, Extended Temperature, Single Op Amp with Chip Select 8 lead TSSOP package. d) MCP603T-I/SN: Tape and Reel, Industrial Temperature, Single Op Amp with Chip Select, 8 lead SOIC package. a) MCP604-I/P: Industrial Temperature, Quad Op Amp, 14 lead PDIP package. b) MCP604-E/SL: Extended Temperature, Quad Op Amp, 14 lead SOIC package. c) MCP604T-E/ST: Tape and Reel, Extended Temperature, Quad Op Amp, 14 lead TSSOP package.

MCP601/1R/2/3/4

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, rfPIC and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2007, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
= ISO/TS 16949:2002 =**

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
 2355 West Chandler Blvd.
 Chandler, AZ 85224-6199
 Tel: 480-792-7200
 Fax: 480-792-7277
 Technical Support:
<http://support.microchip.com>
 Web Address:
www.microchip.com

Atlanta

Duluth, GA
 Tel: 678-957-9614
 Fax: 678-957-1455

Boston

Westborough, MA
 Tel: 774-760-0087
 Fax: 774-760-0088

Chicago

Itasca, IL
 Tel: 630-285-0071
 Fax: 630-285-0075

Dallas

Addison, TX
 Tel: 972-818-7423
 Fax: 972-818-2924

Detroit

Farmington Hills, MI
 Tel: 248-538-2250
 Fax: 248-538-2260

Kokomo

Kokomo, IN
 Tel: 765-864-8360
 Fax: 765-864-8387

Los Angeles

Mission Viejo, CA
 Tel: 949-462-9523
 Fax: 949-462-9608

Santa Clara

Santa Clara, CA
 Tel: 408-961-6444
 Fax: 408-961-6445

Toronto

Mississauga, Ontario,
 Canada
 Tel: 905-673-0699
 Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
 Suites 3707-14, 37th Floor
 Tower 6, The Gateway
 Harbour City, Kowloon
 Hong Kong
 Tel: 852-2401-1200
 Fax: 852-2401-3431

Australia - Sydney

Tel: 61-2-9868-6733
 Fax: 61-2-9868-6755

China - Beijing

Tel: 86-10-8528-2100
 Fax: 86-10-8528-2104

China - Chengdu

Tel: 86-28-8665-5511

Fax: 86-28-8665-7889

China - Fuzhou

Tel: 86-591-8750-3506
 Fax: 86-591-8750-3521

China - Hong Kong SAR

Tel: 852-2401-1200
 Fax: 852-2401-3431

China - Nanjing

Tel: 86-25-8473-2460
 Fax: 86-25-8473-2470

China - Qingdao

Tel: 86-532-8502-7355
 Fax: 86-532-8502-7205

China - Shanghai

Tel: 86-21-5407-5533
 Fax: 86-21-5407-5066

China - Shenyang

Tel: 86-24-2334-2829
 Fax: 86-24-2334-2393

China - Shenzhen

Tel: 86-755-8203-2660

Fax: 86-755-8203-1760

China - Shunde

Tel: 86-757-2839-5507
 Fax: 86-757-2839-5571

China - Wuhan

Tel: 86-27-5980-5300
 Fax: 86-27-5980-5118

China - Xian

Tel: 86-29-8833-7252

Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
 Tel: 91-80-4182-8400
 Fax: 91-80-4182-8422

India - New Delhi

Tel: 91-11-4160-8631
 Fax: 91-11-4160-8632

India - Pune

Tel: 91-20-2566-1512
 Fax: 91-20-2566-1513

Japan - Yokohama

Tel: 81-45-471-6166
 Fax: 81-45-471-6122

Korea - Daegu

Tel: 82-53-744-4301
 Fax: 82-53-744-4302

Korea - Seoul

Tel: 82-2-554-7200
 Fax: 82-2-558-5932 or
 82-2-558-5934

Malaysia - Kuala Lumpur

Tel: 60-3-6201-9857
 Fax: 60-3-6201-9859

Malaysia - Penang

Tel: 60-4-227-8870
 Fax: 60-4-227-4068

Philippines - Manila

Tel: 63-2-634-9065
 Fax: 63-2-634-9069

Singapore

Tel: 65-6334-8870
 Fax: 65-6334-8850

Taiwan - Hsin Chu

Tel: 886-3-572-9526
 Fax: 886-3-572-6459

Taiwan - Kaohsiung

Tel: 886-7-536-4818
 Fax: 886-7-536-4803

Taiwan - Taipei

Tel: 886-2-2500-6610
 Fax: 886-2-2508-0102

Thailand - Bangkok

Tel: 66-2-694-1351
 Fax: 66-2-694-1350

EUROPE

Austria - Wels
 Tel: 43-7242-2244-39
 Fax: 43-7242-2244-393

Denmark - Copenhagen
 Tel: 45-4450-2828
 Fax: 45-4485-2829

France - Paris
 Tel: 33-1-69-53-63-20
 Fax: 33-1-69-30-90-79

Germany - Munich
 Tel: 49-89-627-144-0
 Fax: 49-89-627-144-44

Italy - Milan
 Tel: 39-0331-742611
 Fax: 39-0331-466781

Netherlands - Drunen
 Tel: 31-416-690399
 Fax: 31-416-690340

Spain - Madrid
 Tel: 34-91-708-08-90
 Fax: 34-91-708-08-91

UK - Wokingham
 Tel: 44-118-921-5869
 Fax: 44-118-921-5820

10/05/07