Ecole d'ingénieur Denis Diderot/ M2 Informatique

Compte rendu TP 4 Évalué
Programmation embarquée

Langages, techniques et outils

Vienne Maxime/ Duriez Antoine
23/10/2020

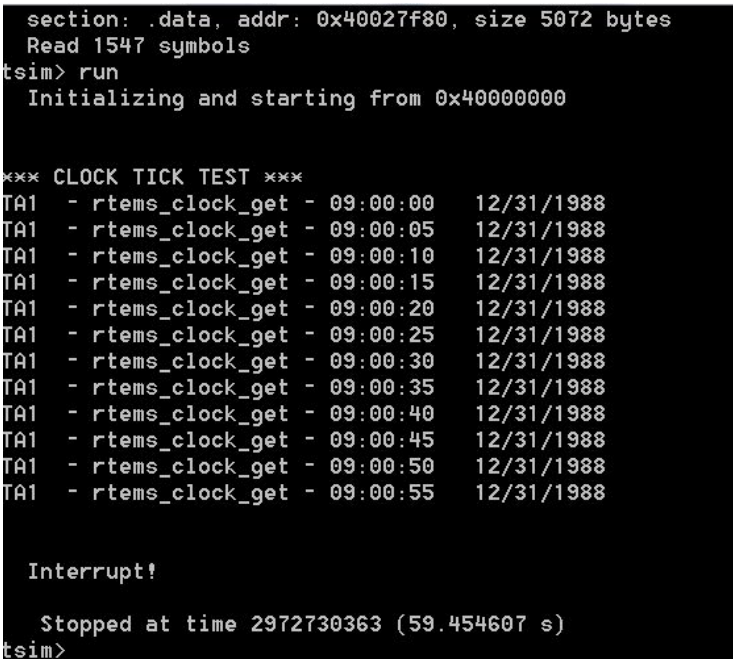
Sommaire:

I) Création du projet RTEMS

Nous avons récupéré le toolchain comme demandé dans l'énoncé. Ensuite nous l'avons ajouté au PATH. Enfin nous avons ajouté les options indiquées dans notre formule de compilation.

II) Développer une tâche d'acquisition

Dans cette partie on a créé la tâche demandé pour jouer le rôle de driver afin d'acquérir une image par étoile toutes les 500ms. Pour cela nous avons utilisé la bibliothèque windows-producer fournie sur le Drive. On obtient en sortie l'image ci-dessous.



>> Quel est le temps entre deux acquisitions?
Il y a 0.5 secondes de temps entre deux acquisitions pour la tâche 1.Comme on peut le voir sur l'image précédente.

III) Utiliser un timer et une queue de messages

-Temps entre deux acquisitions et comparaison avec les valeurs précédentes

Dans cette partie, dans la tâche Init, on initialise les messages queue pour la tâche d'acquisition, et la processing task.

```
// ajout de queue message, pas d'assert dans le cours ici
status = rtems_message_queue_create(rtems_build_name('M','S','Q', '1'),
MESSAGE_QUEUE_COUNT, MESSAGE_QUEUE_SIZE,
RTEMS_LOCAL | RTEMS_PRIORITY, &message_queue_id_1);

// 2 ème queue message pour processing task
status = rtems_message_queue_create(rtems_build_name('M','S','Q', '2'),
  MESSAGE_QUEUE_COUNT, MESSAGE_QUEUE_SIZE,
  RTEMS_LOCAL | RTEMS_PRIORITY, &message_queue_id_2);

// le semaphore se créer
status = rtems_semaphore_create(rtems_build_name('S','E','M', '1'),1, // le 1 permet de récupérer une donnée
  RTEMS_PRIORITY | RTEMS_BINARY_SEMAPHORE | RTEMS_INHERIT_PRIORITY, 0,
  &semaphore_id_1);
  assert(status == RTEMS_SUCCESSFUL);

// ajout du timer, il n'y a pas d'assert dans le cours ici
status = rtems_timer_create(rtems_build_name('T','I','M', '1'), &timer_id_1);
assert(status == RTEMS_SUCCESSFUL);

status = rtems_timer_fire_after(timer_id_1, 50, timer_1_entry, 0); // arme le timer sur 0.5 secondes
assert(status == RTEMS_SUCCESSFUL);
```

On utilise un script gdb pour afficher nos résultats, comme la fonction rtems_clock_get_ticks_since_boot était déclarée plusieurs fois dans la bibliothèque rtems. On passe donc par une déclaration de fonction vide pour faire nos break point.

```
//permet d'afficher dans gdb le clock_get_ticks
rtems_interval f_affichage(rtems_interval a){
    return a;
}
```

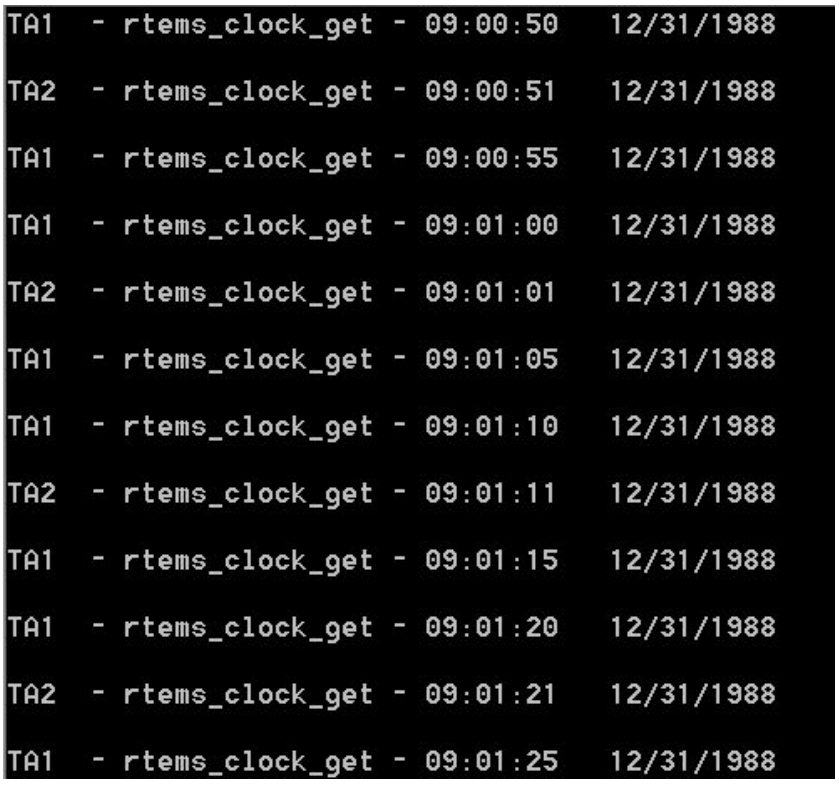
Pour la tâche d'acquisition, elle va avoir un send et un receive, lui permettant de recevoir le timer, et d'envoyer les valeurs du compteur d'acquisition à la tâche Processing.

```
rtems_task Acquisition_task(rtems_task_argument unused)
{
    uint32_t buffer[BUFFER_SIZE];
    size_t size;
  rtems_id          tid;
  rtems_time_of_day time;
  uint32_t  task_index;
  rtems_status_code status;
    /*
    status = rtems_clock_get_tod( &time );
    assert(status == RTEMS_SUCCESSFUL);*/

    init(&wp, img, NB_IMG);
    while(1){
    //appel à produce_images
    produce_images(&wp);
    /*
    break_simu(0,"reset");//reset
    f_affichage(rtems_clock_get_ticks_since_boot());// temps écoulé, on l'appel dans gdb script Gets the current ticks counter value.
    break_simu(1,"f_affichage");// affiche dans gdb
    */
    //receive queue et send
    status = rtems_message_queue_receive(message_queue_id_1,buffer, &size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);// recoit le timer
    assert(status == RTEMS_SUCCESSFUL);
    //break_simu(1,"receive");
    status = rtems_message_queue_send(message_queue_id_2,buffer, size);// envoie la valeur du compteur d'acquisition
    assert(status == RTEMS_SUCCESSFUL);
    //rtems_task_wake_after(50); // acquiert une image toutes les 500 ms

  }
}
```

>> Quel est le temps entre deux acquisitions? Comparez le résultat avec celui obtenu avant l'utilisation du timer.
Il y a 0.5 secondes de temps entre deux acquisitions pour la tâche 1 avec le timer
On ne prend pas en compte les valeurs de la tâche 2 présente qui n'était pas nécessaire pour cet affichage.



IV) Ajouter une tâche de traitement

Nous avons donc créé une nouvelle tâche pour le traitement, ainsi qu'une queue de messages pour indiquer à la tâche quand le buffer est à mis à jour. Enfin on a calculé la photométrie de 10 acquisitions successives. Pour cette partie nous avons réutilisé la fonction flux pondéré du TP2.

V)Transmettre des données entre tâches

Pour transmettre les données calculées précédemment, on crée un buffer circulaire, pour stocker les données avant de les afficher en (FIFO). Notre buffer circulaire est capable de contenir 100 struct flux. Enfin dans Inti() on a initialisé le buffer circulaire. Puis on l'a alimenté avec les struct flux. Le code se trouve dans la partie suivante.

VI)Ajouter une tâche de gestion de télémétrie

- Création de la tâche, affichage de TM et mutex

```
rtems_task Processing_task(rtems_task_argument unused)
{
    uint32_t buffer[BUFFER_SIZE];
    size_t size;
    int cpt=0;
    rtems_status_code status;
    int var_return;
      // flux pour les imagettes
      flux f[100];

      while(cpt< 100){

        // la tache2 recoit la valeur du compteur d'acquisition
        // indique que le buffer d'imagette est à jour
        status = rtems_message_queue_receive(message_queue_id_2,buffer, &size, RTEMS_WAIT, RTEMS_NO_TIMEOUT);
        assert(status==RTEMS_SUCCESSFUL);

        f[cpt].id_window = &wp;
        f[cpt].id_first_acquisition = buffer[0];


        // utiliser tsim avec -nofpu dans le load
        // problème dans le get_mask
        for(uint32_t i = 0 ; i< FLUX_LENGTH ; i++){
            /*printf("image buffer : %d\n", img[buffer[0]]);
            printf("get mask : %d\n", get_mask(&wp, i));*/
            f[cpt].measures[i] = calculFluxPondere(img[buffer[0]], get_mask(&wp, i));
            printf("measures[%d] : %lf\n", i, f[cpt].measures[i]);
        }
        //alimente le buffer circulaire partage en structure de flux
        printf("avant le push cpt : %d \n ",cpt);
        var_return = push_partagee(&buffercp, (uint8_t*)(&f[cpt]), sizeof(flux)); // la taille doit être supérieur
        printf("var_return du push partagee: %d\n", var_return);
        printf("cpt : %d \n ",cpt);
        cpt++;
      }
}
```

Dans cette partie, pour le processing_task, on donne comme argument à la fonction calculFluxPondere, le tableau img, et l'appel à la fonction get_mask. Les valeurs seront générées aléatoirement par la bibliothèque windows producer.

On envoie ensuite les valeurs dans le buffer circulaire partagé.

>> Quelle est la sortie de ce script au cours des 12 premières secondes ?
Données acquises à la sortie du script appelant la fonction send_flux.

	Données1	Données2	Données3	Données4	Données5	Données6	Données7	Données8	Données9	Données10
60	14827.66015	24041.23632	20094.29101	17508.47070	11586.45996	23865.27734	28498.20312	13267.62500	18983.07617	19925.345703
110	9779.040039	13243.91113	13581.32324	10131.57812	8328.549805	13951.08789	17355.45312	7999.962891	10727.82421	11774.933594
160	28787.23437	43001.56640	37754.57031	34801.47265	22251.58398	43062.63281	53772.70312	25921.81835	37474.29296	34145.402344
210	369.698975	490.502930	534.954285	383.544464	369.375092	554.305542	699.480835	288.164398	398.199646	497.304230
260	24390.85156	38453.80468	34106.51562	28251.23046	21097.51171	40444.60156	47413.41796	21183.80078	30488.56445	34348.945312
310	3035.507812	4399.955566	4065.586182	3380.310791	2551.031250	4543.172363	5881.634766	2627.610596	3608.572021	3892.840088
360	16464.95117	22899.96093	20950.09179	18216.75390	11748.73339	22691.99023	29374.58398	15138.99707	19682.45703	17023.945312
410	11763.21777	17952.92578	15393.47656	14429.99023	8895.669922	17895.04492	22104.11718	10784.65820	15933.10058	13783.042969
460	1593.342285	2583.168457	2316.123779	1802.366211	1522.923828	2763.677734	3268.756836	1278.690674	1889.397461	2632.758789
510	12468.75683	16494.04687	16504.68554	13038.22656	9714.311523	16838.70703	21870.32226	10611.42089	13836.05078	13671.521484
560	14742.61523	23903.34570	19979.03906	17408.05078	11520.00488	23728.39648	28334.75000	13191.52734	18874.19726	19811.062500
610	56974.46093	77161.42968	79127.25000	59028.41015	48523.64062	81281.57031	101116.0156	46609.23437	62502.25000	68602.898438
660	3931.416748	5872.640625	5156.068848	4752.769531	3038.855957	5880.980469	7343.634277	3540.092773	5117.791504	4663.171387
710	7471.344238	9912.703125	10811.03125	7751.151367	7464.798828	11202.10742	14135.99218	5823.590332	8047.321289	10050.152344
760	13749.65039	21677.24218	19226.58007	15925.82910	11893.12304	22799.49609	26727.96875	11941.76660	17187.06250	19363.242188
810	3017.803467	4374.293457	4041.874268	3360.595703	2536.152832	4516.674805	5847.330566	2612.285400	3587.525391	3870.135742
860	3662.230225	5093.542480	4659.841309	4051.876221	2613.221436	5047.284180	6533.665527	3367.303711	4377.886230	3786.564697
909	288.112518	439.714935	377.027222	353.428864	217.878647	438.297272	541.388672	264.144989	390.244049	337.583405
960	9857.219727	15980.78418	14328.70996	11150.34668	9421.575195	17097.50585	20222.17968	7910.625977	11688.76660	16287.575195
1010	857.682678	1134.568481	1135.300293	896.854492	668.213928	1158.276489	1504.383789	729.922974	951.734070	940.416687
1060	2467.733398	4001.127686	3344.246826	2913.894775	1928.307983	3971.843262	4742.890625	2208.100342	3159.309814	3316.129639
1110	15633.26757	21172.38671	21711.78906	16196.85253	13314.44140	22302.91406	27745.30468	12789.14550	17150.04296	18824.003906
1160	14991.46777	22393.83593	19661.36914	18123.49023	11587.91113	22425.63671	28003.09960	13499.25195	19515.40820	17781.830078
1210	8433.196289	11188.85351	12202.83105	8749.025391	8425.808594	12644.25390	15955.84375	6573.312988	9083.324219	11343.997070

>> Quel est le code du buffer_circulaire_partagee ?

```
void init_buffer_circulaire_partagee (buffer_circulaire_partagee* buffer_circulaire, uint8_t* buffer, uint16_t taille_donnee, uint32_t nombre_donnees){
    init_buffer_circulaire(&buffer_circulaire->buffer_circulaire, buffer, taille_donnee, nombre_donnees);
}

int push_partagee(buffer_circulaire_partagee* fifo, uint8_t* source, uint16_t taille) {
    int var_return;
    rtems_semaphore_obtain(semaphore_id_1, RTEMS_WAIT, RTEMS_NO_TIMEOUT);// verrouille la ressource partagée
    var_return = push(&fifo->buffer_circulaire, source, taille);
    rtems_semaphore_release(semaphore_id_1);// libère la ressources partagée
    return var_return;
}

int pop_partagee(buffer_circulaire_partagee* fifo, uint8_t* destination, uint16_t taille_max) {
    int var_return;
    rtems_semaphore_obtain(semaphore_id_1, RTEMS_WAIT, RTEMS_NO_TIMEOUT);
    var_return = pop(&fifo->buffer_circulaire, destination, taille_max);
    rtems_semaphore_release(semaphore_id_1);
    return var_return;
}
```

Pour le buffer circulaire partagé on utilise des semaphore pour verrouiller la ressources partagée utilisée par la tâche courante, et ensuite libérer la ressource une fois l'action terminée.

- Régulation de débit

>> Quel est le code de la tâche de régulation de télémétrie ?

```
rtems_task TelemetryManager_task(rtems_task_argument unused)
{
    const uint32_t interval_de_regulation = 10;
    const uint32_t nombre_maximal_de_flux = 30;
    uint32_t debut = rtems_clock_get_ticks_since_boot(); //On conserve l'heure de début du slot (moment d'émission)
    uint32_t static cpt_flux = 0; // nombre de flux déjà émis depuis le début
    flux pop_flux;      //Prend le flux du buffer circulaire partagee
    int vide;// si le pop_partagee ne rend rien on affiche rien

    while(1){
        if(cpt_flux > nombre_maximal_de_flux){  //Si 30 flux ont été émis depuis de début du slot.
            rtems_task_wake_after(interval_de_regulation - (rtems_clock_get_ticks_since_boot() - debut));   //la tâche se met en pause jusqu'à la fin du slot
        }else if(rtems_clock_get_ticks_since_boot() > debut + interval_de_regulation){  //Quand un flux est à émettre et que la fin de slot est passé,
            cpt_flux = 0; // remise à 0 du compteur
            debut = rtems_clock_get_ticks_since_boot(); // on redonne l'heure
        }else{
            vide = pop_partagee(&buffercp, (uint8_t*)(&pop_flux), sizeof(flux));
            if(vide > 0){// on affiche seulement si il y a quelque chose dans le buffer circulaire
                break_simu(0,"reset");//reset
                send_flux(rtems_clock_get_ticks_since_boot(), pop_flux);
                break_simu(1,"send_flux");// affiche dans gdb
                cpt_flux++;
             }
        }
    }
}
```

Pour la TelemetryManager_task, on déclare les variables qui vont nous permettre de compter les flux jusqu'à 30. De mettre en pause la tâche jusqu'à la fin du slot, de réinitialiser le compteur, lorsqu'un flux apparaît mais que le slot est terminé. Enfin une variable d'entier pour lire l'état du pop_partagee, et le donné à send_flux lorsqu'il y a une valeur, et que le buffer n'est pas vide. Si le buffer est vide et qu'il est lu, cela provoque des erreurs.

>> Quelles sont les données émises au cours des 12 premières secondes ?

60	14827.66015	24041.23632	20094.29101	17508.47070	11586.45996	23865.27734	28498.20312	13267.62500	18983.07617	19925.345703
110	9779.040039	13243.91113	13581.32324	10131.57812	8328.549805	13951.08789	17355.45312	7999.962891	10727.82421	11774.933594
160	28787.23437	43001.56640	37754.57031	34801.47265	22251.58398	43062.63281	53772.70312	25921.81835	37474.29296	34145.402344
210	369.698975	490.502930	534.954285	383.544464	369.375092	554.305542	699.480835	288.164398	398.199646	497.304230
260	24390.85156	38453.80468	34106.51562	28251.23046	21097.51171	40444.60156	47413.41796	21183.80078	30488.56445	34348.945312
310	3035.507812	4399.955566	4065.586182	3380.310791	2551.031250	4543.172363	5881.634766	2627.610596	3608.572021	3892.840088
360	16464.95117	22899.96093	20950.09179	18216.75390	11748.73339	22691.99023	29374.58398	15138.99707	19682.45703	17023.945312
410	11763.21777	17952.92578	15393.47656	14429.99023	8895.669922	17895.04492	22104.11718	10784.65820	15933.10058	13783.042969
460	1593.342285	2583.168457	2316.123779	1802.366211	1522.923828	2763.677734	3268.756836	1278.690674	1889.397461	2632.758789
510	12468.75683	16494.04687	16504.68554	13038.22656	9714.311523	16838.70703	21870.32226	10611.42089	13836.05078	13671.521484
560	14742.61523	23903.34570	19979.03906	17408.05078	11520.00488	23728.39648	28334.75000	13191.52734	18874.19726	19811.062500
610	56974.46093	77161.42968	79127.25000	59028.41015	48523.64062	81281.57031	101116.0156	46609.23437	62502.25000	68602.898438
660	3931.416748	5872.640625	5156.068848	4752.769531	3038.855957	5880.980469	7343.634277	3540.092773	5117.791504	4663.171387
710	7471.344238	9912.703125	10811.03125	7751.151367	7464.798828	11202.10742	14135.99218	5823.590332	8047.321289	10050.152344
760	13749.65039	21677.24218	19226.58007	15925.82910	11893.12304	22799.49609	26727.96875	11941.76660	17187.06250	19363.242188
810	3017.803467	4374.293457	4041.874268	3360.595703	2536.152832	4516.674805	5847.330566	2612.285400	3587.525391	3870.135742
860	3662.230225	5093.542480	4659.841309	4051.876221	2613.221436	5047.284180	6533.665527	3367.303711	4377.886230	3786.564697
909	288.112518	439.714935	377.027222	353.428864	217.878647	438.297272	541.388672	264.144989	390.244049	337.583405
960	9857.219727	15980.78418	14328.70996	11150.34668	9421.575195	17097.50585	20222.17968	7910.625977	11688.76660	16287.575195
1010	857.682678	1134.568481	1135.300293	896.854492	668.213928	1158.276489	1504.383789	729.922974	951.734070	940.416687
1060	2467.733398	4001.127686	3344.246826	2913.894775	1928.307983	3971.843262	4742.890625	2208.100342	3159.309814	3316.129639
1110	15633.26757	21172.38671	21711.78906	16196.85253	13314.44140	22302.91406	27745.30468	12789.14550	17150.04296	18824.003906
1160	14991.46777	22393.83593	19661.36914	18123.49023	11587.91113	22425.63671	28003.09960	13499.25195	19515.40820	17781.830078
1210	8433.196289	11188.85351	12202.83105	8749.025391	8425.808594	12644.25390	15955.84375	6573.312988	9083.324219	11343.997070

Les ticks s'exécutent tous les 500ms, dont on prend les 24 premières valeurs correspondantes aux 12 premières secondes. Cela équivaut à prendre les valeurs du timer de 60ms à 1210ms, ici représenté par la première colonne.

>> Quel est le comportement du système si la tâche d'émission de TM est de plus haute priorité que la tâche de traitements ? Pourquoi ?

La priorité va faire que certaines tâches vont se bloquer, car elles n’auront pas la priorité sur la ressource partagée. Si on met la tâche télémétrie Manager à une priorité plus faible, elle ne pourra pas lire les données, et certaines seront perdues, pendant que la tâche d’acquisition écrira une nouvelle valeur.