

Présentation de la BeagleBone Black (BBB) et son utilisation dans les applications temps réel.

fabien.lementec@gmail.com

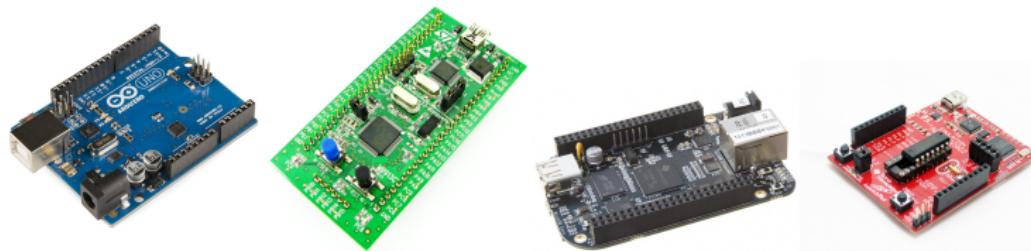
<https://www.logre.eu>



Plan

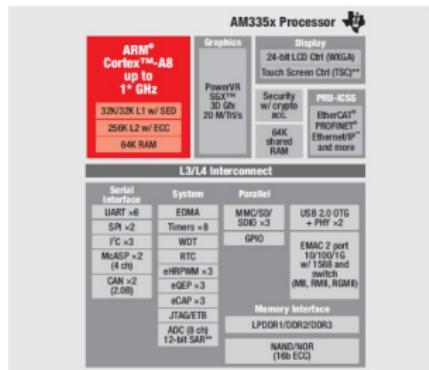
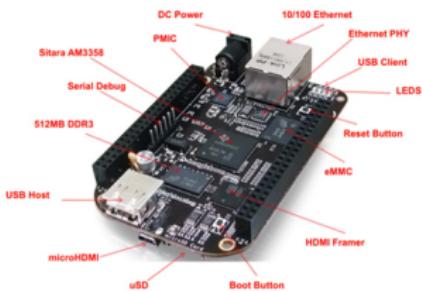
- Introduction
- BeagleBone Black
- Temps réel
- Programmable Realtime Unit
- Démonstration
- Questions

Introduction



- Multiplication des solutions pour l'embarqué
- Compromis: puissance, périphériques, coût, complexité, support ...
 - ✓ La BeagleBone Black s'impose dans sa catégorie

BBB - Caractéristiques matérielles



- Processeur: AM335x 1GHz ARM Cortex-A8
- Mémoire RAM: 512MB DDR3
- Flash intégrée: 4GB eMMC
- Connectivité: Ethernet, USB (client et hôte)
- Périphérie: UART, I2C, SPI, CAN, PWM, ADC ...
- Extension: 2x 46 pin headers

BBB - Cartes d'extension, les *capes*

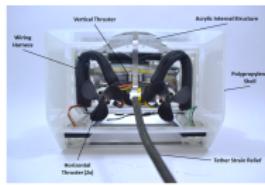
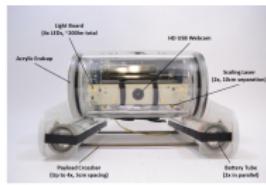


- Capteurs et acquisition
- Logique (FPGA ...)
- Affichage (LCD ...)
- Communication (CAN, radio ...)
- Prototypage
- http://elinux.org/Beagleboard:BeagleBone_Capes

BBB - Support

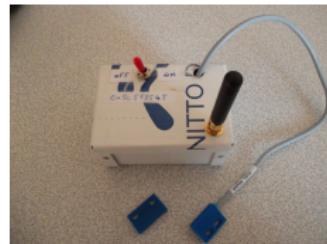
- Texas Instrument
 - processeur ARM3358
 - <http://processors.wiki.ti.com/index.php/Sitara>
- Farnell Element14
 - Communauté (forums ...)
 - <http://www.element14.com/BeagleBone-Black>

BBB - Exemple d'application: robotique



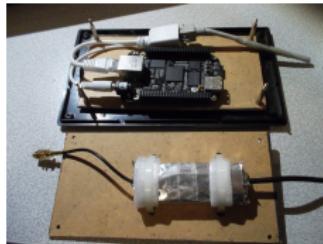
- OpenROV
- Webcam, propulsion, éclairage, serveur http (NodeJS)
- <http://openrov.com>

BBB - Exemple d'application: domotique



- BANO
- Dongle NRF, webcam, serveur http (Mongoose)
- <https://github.com/texane/bano>
- https://github.com/texane/bano_reed

BBB - Exemple d'application: SDR



- Dongle USB Realtek, librtlsdr

BBB - Références

- <http://beagleboard.org/black>
- <http://elinux.org/Beagleboard:BeagleBoneBlack>

Temps réel - Définition

Certaines applications nécessitent qu'une partie du code s'exécute précisément dans le temps. On parle d'applications *temps réel*.

L'ordre de grandeur varie selon l'application, et dépend largement des processus physiques impliqués.

Exemples:

- Pilotage de moteurs CNC: microsecondes
- Contrôle alimentation secteur: millisecondes
- Asservissement en température: secondes

Temps réel - Problématique

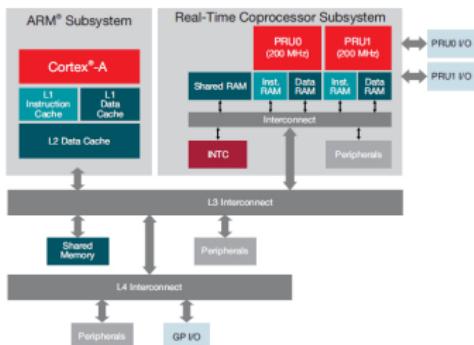
Il est difficile de garantir le respect des contraintes temporelles dès lors que le code de l'application partage les ressources (CPU, mémoire, IOs ...) avec un autre code plus prioritaire et non prévu pour cela (Linux).

Temps réel - Solutions pour Linux

2 types de solutions

- Modification du noyau
 - Realtime Linux
 - Xenomai
- Coprocesseurs dédiés
 - Auxiliaires: microcontrôleur, FPGA
 - Intégrés: **PRU**, Zynq, cœur dédié

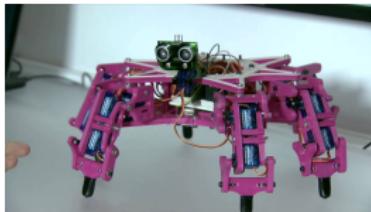
PRU - Introduction



Le processeur de la BBB intègre 2 microcontrôleurs dédiés à l'application: les **PRUs**

- Programmable Realtime Unit
- Pas de système d'exploitation
- 200MHz 32 bits RISC, 5 ns par instruction
- Mémoire locales (instructions: 8KB, données: 8KB), partagée (12KB)
- Périphériques dédiés (5ns latency Enhanced GPIOs)
- Accès à la totalité des périphériques via un bus d'interconnection
- Communication processeur (mémoire partagée, interruptions)

PRU - Applications



- Contrôle et asservissement: CNC, robots ...
- Mesure et instrumentation
- http://processors.wiki.ti.com/index.php/PRU_Projects

PRU - Workflow

Préparation

- Installation toolchain GNU et PRU
- Compilation noyau (activation driver PRUSS)
- Création d'une image disque

Développement

- Programmes hôte et PRU
- Script device tree

Lancement de l'application

- Chargement driver et overlay device tree
- Lancement programme hôte

PRU - Installation toolchain GNU pour ARM

Nécessaire pour compiler les programmes hôtes et le noyau.

Différentes méthodes:

- crosstool-ng
 - <https://github.com/texane/lfs/blob/master/board/bbb/crosstool-ng-1.18.0.config>
- gestionnaire de paquets
 - `aptitude install gcc-4.7-arm-linux-gnueabi`
- http://elinux.org/Building_BBB_Kernel

PRU - Installation toolchain PRU

Nécessaire pour compiler les programmes PRU

- http://github.com/texane/pru_sdk
- <http://www.embeddedrelated.com/showarticle/586.php>

PRU - Compilation noyau Linux

Nécessaire pour activer le driver PRU

- version 3.8.13
- CONFIG_UIO_PRUSS=m
- modprobe uio_pruss
- <https://github.com/texane/lfs/blob/master/board/bbb/linux-3.8.13.config>

PRU - Création d'une image disque

Utilisation de l'outil LFS

- <https://github.com/texane/lfs>
- Compile aussi une toolchain GNU et un noyau
- L'image produite se copie via dd sur une carte SD

PRU - Overlay Device Tree (1)

Device Tree est un sous système de Linux qui décrit le matériel au noyau sous la forme d'une arborescence de périphériques:

- http://elinux.org/Device_Tree
- [petazzoni-device-tree-dummies.pdf](#)
- <http://hipstercircuits.com/beaglebone-black-gpio-mux-for-pru-with-device-tree-overlay/>

Un script Device Tree (xxx.dts) est nécessaire pour modifier, étendre ou surcharger cette arborescence. On parle aussi d'*overlay*.
Dans notre cas:

- Activer le PRU
- Configurer le multiplexeur des IOs

PRU - Overlay Device Tree (2)

L'overlay se compile en un binaire (xxx.dtbo) puis se copie sur la cible dans dans un répertoire particulier:

- `dtc -@ -O dtb -o xxx.dtbo xxx.dts`
- `cp xxx.dtbo /lib/firmware/`

Avant l'exécution du programme, l'overlay doit être ajouté à l'arborescence des périphériques du noyau:

- `echo part_number > /sys/devices/bone_capemgr.9/slots`
- *part_number* est un identifiant défini dans l'overlay

PRU - Overlay Device Tree (3)

Vérification du chargement via dmesg

```
[ 996.191234] bone-capemgr bone_capemgr.9: part_number 'gpio_p8_11',
version 'N/A'
[ 996.191350] bone-capemgr bone_capemgr.9: slot #8: generic override
[ 996.191377] bone-capemgr bone_capemgr.9: bone: Using override eeprom
data at slot 8
[ 996.191407] bone-capemgr bone_capemgr.9: slot #8: 'OverrideBoard
Name,00A0,OverrideManuf,gpio_p8_11'
[ 996.191512] bone-capemgr bone_capemgr.9: slot #8: Requesting part
number/version based 'gpio_p8_11-00A0.dtbo
[ 996.191546] bone-capemgr bone_capemgr.9: slot #8: Requestingfirmware
'gpio_p8_11-00A0.dtbo' for board-name'Override Board Name',version'00A0',
[ 996.191793] bone-capemgr bone_capemgr.9: slot #8: dtbo 'gpio_p8_11-00A0.dtbo'
loaded; converting to live tree
[ 996.192981] bone-capemgr bone_capemgr.9: slot #8: #3overlays
[ 996.195171] bone-capemgr bone_capemgr.9: slot #8: Applied #3overlays
uu
```

PRU - Overlay Device Tree (4)

Overlay d'activation du PRU

```
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "pruss_enable";
    version = "00A0";

    fragment@0 {
        target = <&pruss>;
        __overlay__ {
            status = "okay";
        };
    };
};

};
```

PRU - Overlay Device Tree (5)

Overlay de multiplexage la pin 8.15 en mode PWM

```
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "my_pwm_P8_15";
    version = "00A0";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {
            pwm_P8_15: pinmux_pwm_P8_15_pins {
                pinctrl-single,pins = <0x3c 0x5>;
            };
        };
    };

    fragment@1 {
        target = <&ocp>;
        __overlay__ {
            pwm_test_P8_15 {
                compatible      = "pwm-test";
                pwm-names      = "PWM_P8_15";
                pinctrl-names   = "default";
                pinctrl-0       = <&pwm_P8_15>;
                enabled         = <1>;
                status          = "okay";
            };
        };
    };
};
```

PRU - Overlay Device Tree (6)

En pratique, on copie colle un script existant en changeant les offsets et les modes de multiplexage des pins. 2 documentations utiles:

- https://github.com/texane/pru_sdk/blob/master/doc/BeagleboneBlackP8HeaderTable.pdf
- https://github.com/texane/pru_sdk/blob/master/doc/BeagleboneBlackP9HeaderTable.pdf

PRU - Programme PRU en assembleur (1)

Converti un programme écrit dans le langage assembleur en un binaire qui sera chargé sur le PRU par le programme hôte:

- `pasm -V2 -I$(PRU_SDK_DIR)/include -b xxx.p`
- Produit `xxx.bin`

PRU - Programme PRU en assembleur (2)

Jeu d'instructions généraliste

- Branchement, arithmétique, load store ...
- Unité MAC avec saturation
- am335xPruReferenceGuide.pdf

PRU - Programme PRU en assembleur (3)

```
// blink.pasm

#define GPIO1 0x4804c000
#define GPIO_CLEARDATAOUT 0x190
#define GPIO_SETDATAOUT 0x194

INIT_OCP:
    LBC0 r0, C4, 4, 4
    CLR r0, r0, 4
    SBC0 r0, C4, 4, 4

    MOV r1, 10

BLINK:
    MOV r2, 7<<22
    MOV r3, GPIO1 | GPIO_SETDATAOUT
    SBB0 r2, r3, 0, 4
    MOV r0, 0x00a00000

DELAY:
    SUB r0, r0, 1
    QBNE DELAY, r0, 0
    MOV r2, 7<<22
    MOV r3, GPIO1 | GPIO_CLEARDATAOUT
    SBB0 r2, r3, 0, 4
    MOV r0, 0x00a00000

DELAY2:
    SUB r0, r0, 1
    QBNE DELAY2, r0, 0
    SUB r1, r1, 1
    QBNE BLINK, r1, 0
```

PRU - Programme PRU en C (1)

Texas Instrument CSS (celui utilisé)

- http://processors.wiki.ti.com/index.php/Download_CCS
- www.embeddedrelated.com/showarticle/603.php
- Celui utilisé dans cette conférence

GNU GCC, portage non officiel

- <https://github.com/dinuxbg/gnupru>
- difficile à installer et fonctionnalités absentes (05/2015)

PRU - Programme PRU en C (2)

```
/* blink.c */

#include <pru/io.h>

static void delay_cycles(unsigned int n)
{
    unsigned int i;
    for (i = 0; i < n/2; i++) asm volatile ("nop" : : );
}

static void delay_us(unsigned int us)
{
    /* assume cpu frequency is 200MHz */
    delay_cycles(us * (1000 / 5));
}

const unsigned int period_us = 250 * 1000;

int main(void)
{
    unsigned int c;

    for (c = 0; ; c++)
    {
        write_r30(c & 1 ? 0xffff : 0x0000);
        delay_us(period_us);
    }

    return 0;
}
```

PRU - Programme PRU en C (3)

Assembleur inline possible

```
uint32_t shm_read(register uint32_t i)
{
    /* i is the absolute offset relative from shared memory start */
    /* read x at shm + i */

    __asm__ __volatile__
    (
        "    LDI32 r0, 0x000000120\n"
        "    LDI32 r1, 0x22028\n"
        "    SBB0 r0, r1, 0, 4\n"

        "    LDI32 r0, 0x00100000\n"
        "    LDI32 r1, 0x2202c\n"
        "    SBB0 r0, r1, 0, 4\n"

        "    LBC0 r14, C31, 0, 4\n"
        "    JMP R3.w2\n"
    );
    /* unreached */
    return 0;
}
```

PRU - Programme hôte (1)

Le programme hôte s'exécute sur le processeur hôte. Il initialise le PRU et y charge le binaire précédemment compilé. En cours d'exécution, l'hôte communique avec le PRU via de la mémoire partagée et des interruptions.

- Communication de consignes
- Acquisition de données
- Attente d'événements

PRU - Programme hôte (2)

```
/* main.c */

#include "prussdrv.h"
#include "pruss_intc_mapping.h"

int main(int ac, char** av)
{
    /* initialize the PRU */
    pruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;
    prussdrv_init();
    if (prussdrv_open(PRU_EVTOUT_0)) return -1;
    prussdrv_pruinitc_init(&pruss_intc_initdata);

    /* ... */

    /* execute code on pru0 */
#define PRU_NUM 0
    prussdrv_exec_program(PRU_NUM, "./xxx.bin");

    /* ... */

    /* wait until pru0 finishes */
    prussdrv_pru_wait_event(PRU_EVTOUT_0);
    prussdrv_pru_clear_event(PRU_EVTOUT_0, PRUO_ARM_INTERRUPT);

    /* disable pru and close memory mapping */
    prussdrv_pru_disable(PRU_NUM);
    prussdrv_exit();

    return 0;
}
```

PRU - Environements et outils

Programmation Python

- <http://hipstercircuits.com/pypruss-a-simple-pru-python-binding-for-beaglebone>

Déboggage

- http://processors.wiki.ti.com/index.php/PRU_Debugging
- <http://sourceforge.net/projects/prudebug>

PRU - Références

- <http://beagleboard.org/pru>
- http://elinux.org/Ti_AM33XX_PRUSSv2
- <http://processors.wiki.ti.com/index.php/PRU-ICSS>

Démonstration - pru_sdk/example/pruss_iep

Le CPU lit des valeurs écrites par le PRU à intervalles de temps régulier dans une zone de mémoire partagée. Le module de timers IEP est utilisé pour cadencer l'écriture du PRU.

Démonstration - pru_sdk/example/pruss_spwm

Le PRU génère un signal carré. La fréquence est donnée par le programme hôte via une valeur écrite en RAM.

Questions

**Questions ?
Remarques pour un futur atelier ?**