On Using Sitara AM335x Starter Kit to achieve basic applications based on Linux operating system

SEPTIMIU MISCHIE, ROBERT PAZSITKA
Faculty of Electronics and Telecommunications
Politehnica University of Timisoara
Vasile Parvan 2
ROMANIA
septimiu.mischie@etc.upt.ro, robert.pazsitka@etc.upt.ro, http://www.etc.upt.ro

Abstract: - Recently, Texas Instruments has been presented the new Sitara AM335x Starter Kit (short, AM335x SK) system. This device is based on an AM3358 ARM processor and it can be used by either a Linux operating system or an Android operating system. Also the system contains a 4.3 inch LCD touchscreen and other hardware resources. In the literature there are very few documents which report applications with this new system. This paper presents Linux based applications with this system like: accessing the GPIO pins, accessing the user LED's and keys, connecting an I2C device to the appropriate connector, reading the values from the accelerometer. All of these applications can be started from a PC which is connected to the AM335x SK by Ethernet and also by using the touchscreen of the AM335x SK.

Key-Words: - Sitara AM335x Starter Kit, Linux, Code Composer, Matrix application, I2C interface.

1 Introduction

The applications based on ARM processors and Linux operating systems are very popular today [1-3], [6]. Also, development systems like Beagle Board [4], AM335x Evaluation Module and Sitara AM335x Starter Kit [5] were launched recently on the market. In the literature, especially on the Internet, there are documents about these systems, but very few present feasible applications. So, this paper presents all the necessary steps to achieve applications using Sitara AM335x Starter Kit.

The paper is organized as follows. Chapter 2 presents the hardware and software structure of this system. Chapter 3 presents the software requirements for the host PC which is used to develop applications on AM335x SK. The applications which are started by the host PC are presented in chapter 4, while chapter 5 presents how these applications can be started by the touchscreen of the AM335x SK. Finally, chapter 6 presents the conclusions.

2 The Structure of the AM335x SK

The hardware structure

The AM335x SK is based on an AM3358 ARM Cortex A8 processor and contains a LCD with touchscreen. Other resources are: one micro SD card connector, 256 MB DDR3 SDRAM memory, two Ethernet connectors, one USB connector, one micro USB connector, one WiFi/BlueTooth module, one stereo audio output (stereo jack), four user LED's and four keys. Also, the system contains an accelerometer and two expansion connectors. The first has 6 pins and corresponds to SPI interface; the other has 4 pins for I2C interface, see Appendix 1. The system is powered by a 5V adapter.

The software structure

The AM335x SK comes with two micro SD cards, one with Linux and the other with Android. Thus, the system can boot up with one of these operating systems, depending which of them is introduced into the micro SD card connector.

In this paper we are concerning with applications achieved when the AM335x SK uses the Linux operating system. So, when the system is boot up, the application called Matrix Application Launcher is started. As effect, a 4x3 matrix of icons are displayed. These icons can be operated by touch.

3 The Software Requirements of the Host PC

In order to develop applications for Sitara AM335x system when it runs Linux, it is necessary a host Personal Computer (PC) running Linux (Linux PC). The simplest solution to have a Linux PC is to install a virtual machine like VMWare on a Windows PC. Then, the Linux distribution called Ubuntu 10.04 LTS can be installed under VMWare.

In this way, the content of the Linux SD card of the AM335x SK can be accessed under the Ubuntu.

ISBN: 978-960-474-349-0

In addition to Linux Operating System, this card contains the necessary tools to develop applications like Sitara Linux SDK and Code Composer Studio v5 (CCS). By connecting the Linux SD card to the Linux PC by using an USB card reader, three partitions are mounted under Ubuntu. One of them, START_HERE, can be accessed to install the Sitara Linux SDK and the CCS. The Sitara Linux SDK contains drivers and libraries which are used by CCS to develop applications. The Linux SD card can also be accessed direct on the AM335x SK, by using an USB cable between the Linux PC and the AM335x SK (micro USB connector).

It is known that CCS is used to write applications in C language and download them into the flash memory of the processors to execute or debug them. Furthermore, in this case, CCS allows the Linux PC to see the resources (directories and files) of the AM335x SK (called *Target*), in addition to its own resources, and also to send Linux commands to the Target. For this purpose, the AM335x SK and the Linux PC must be connected in a local network. This can be done by connect the AM335x SK to an output of a (wireless) router using one Ethernet connector while the Linux PC is (wireless) connected to the same router. The IP which the router has been allocated to AM335x SK can be read by accessing Settings/Network Settings – eth0, within Matrix Application Launcher. This IP is used in the process of configuring the Remote Interface option in CCS. Figure 1 presents a capture form the CCS. It can be seen Local (Linux PC) and Target (its name is My Target EVM). Thus, the applications which were created on the Linux PC can be moved on the Target using the classical copy/paste procedure. These applications can be achieved using either CCS or any other method (as in the future will be presented).

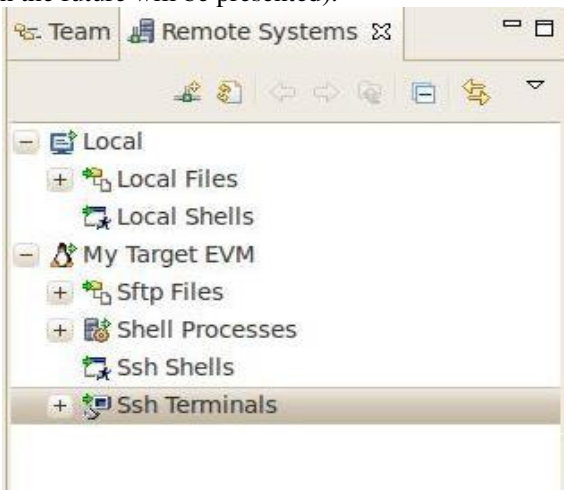

Fig. 1 Resources of the Local and Target, in CCS

Also the IP of AM335x SK can be used to run the Matrix Application Launcher from a web browser located on a Linux PC (remote mode).

The other facility of CCS is to allow executing the Linux commands designed to AM335x SK as we presented. This can be done by the Terminals tab of CCS (fig.2). Examples:

1. General Linux commands. Example:

```
~# pwd       //print work directory
/home/root     //this is the answer
```

2. Commands to configure the peripherals of the ARM processor. Example:

```
~#  echo 3 >/sys/class/gpio/export
```

3. Command to execute the application which have been created in CCS and then moved on the Target Example (the application is named application_1):

```
~# chmod +x application_1
~# ./application_1
```

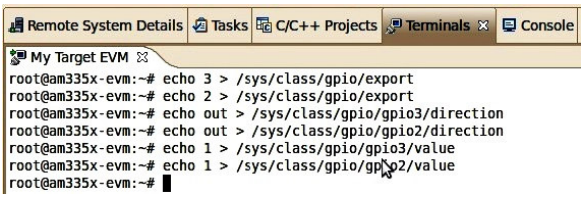

Fig. 2 Linux commands in Terminals tab of CCS

4 Example Applications

In order to create applications which uses the peripherals of the ARM processor but also the external resources which are on the system (accelerometer, touch screen, …), the Linux operating system of the Target has directories, subdirectories and files which allows configuring the peripherals and writing or reading data to/from them. In this way is not necessary to know the registers of the processor and their bits to create an application. Appendix 2 presents the paths to the files which are used in this paper, and some further information about them. These files can be accessed either using Linux commands or applications created in C language, using CCS.

Using the Linux commands represents a very simple possibility to test the peripherals of the processor. Thus, the command *echo* can be used for write to files and command *cat* for read the files. The functions fopen(), fclose(), fread() and fwrite() can be used to make similar things in C language.

In the following, a few applications achieved using Linux commands but also C language are presented.

1. *Writing and reading the logic level of the general purpose input-output pins.*

To use this facility it is necessary to know the number of the pin, denoted by *x*. For AM3358, *x* can

ISBN: 978-960-474-349-0

be from 1 to 128, see Appendix 1 for details. Throughout in this example, *x*=3, because this pin is available in connector J11. The first step is to execute the following command.

```
~# echo 3 >/sys/class/gpio/export
```

The effect of this command is to create the directory gpio3, which contains files for handling the pin 3 (direction, value, etc.).

Then, the command that configure the direction of the pin 3 as an output (*out*) or input (*in*) must be executed. In this example, the pin 3 is configured as output.

```
~# echo out > /sys/class/gpio/gpio3/direction
```

Finally, the command that set the logic level, 1 or 0 (1 in this example) of the pin can be executed.

```
~# echo 1 >/sys/class/gpio/gpio3/value
```

If a pin is configured as an input, the following command read its logic level, while the result of reading is displayed on the next row, 1 in this example.

```
~# cat /sys/class/gpio/gpio3/value
1
```

If a pin configured as output or input will no longer be used, it can be disabled by the command

```
~# echo 3 >/sys/class/gpio/unexport
```

We also tested this application on the pins having the numbers 2, 5 and 6, because these pins are available in the expansion connector J11 (see Appendix 2).

2. *Accessing the user LED's and the user keys.* The four user LED's are connected to the pins gpio1_4,…,gpio1_7, whereas the four keys are connected to the pins gpio0_30, gpio2_3, gpio2_2 and gpio2_5. Appendix 1 presents the method to convert these GPIO numbers of the pins in the numbers within the range 1-128. These pins can not be configured using the statements from the previous application. Thus, the command `echo x >/sys/class/gpio/export` has no effect for these pins, that is the directory gpio*x* is not created. Instead, there are special directories which allow configuring of these devices.

In Appendix 1 is presented the path where the directories for the four LED's are: `/sys/devices/platform/leds-gpio/leds`. Here there are four directories, one for each led:

```
am335x:EVM_SK:usr0
am335x:EVM_SK:usr1
am335x:EVM_SK:mmc0
am335x:EVM_SK:heartbeat.
```

Each of these directories contains more files. One of them is *trigger*. The content of this file specifies how (when) that led is controlled. By default, the content of this file is *none* for the first two LED's. That is, there is no a device which controls these LED's, so they can be controlled by the user, so it will be seen.

The content of the *trigger* file is *mmc0* for the third led, that means that it is controlled when the SD card (or multi media card) is accessed.

The content of the *trigger* file is *heartbeat* for the fourth led, that means it blinks depending of the CPU (processor) activity.

Thus, the first two LED's can be set "on" by the command:

```
~# echo 1 >/sys/devices/platform/leds-gpio/leds /am335x:EVM_SK:usrx/brightness,
```

where *x* can be 0 or 1. If the argument 0 replaces 1 after the echo, the LED will be "off ". This command can also be applied for the third LED (mmc0), if the SD card is not accessed.

The fourth led, which blinks all the time, can be disabled by the command.

```
~# echo none >/sys/devices/platform/leds-gpio/leds/am335x:EVM_SK:heartbeat/trigger
```

This command can be applied to the third LED too.

Then, these LED's can be set "on" or "off " similar to the first two LED's.

To read the status of the four user keys, the next command can be used:

```
~# cat /dev/input/event2 |hexdump
```

After the execution of this command, the system is waiting for pressing a key. Only when a key is pressed (the logic level applied to the pin is changed) an answer message will be displayed in the terminal. However, the content of the message is difficult to understand due to hex codes. To obtain a better display mode, the next command can be executed

```
~# evtest /dev/input/event2
```

In this case, the answer message can be like in Figure 3.

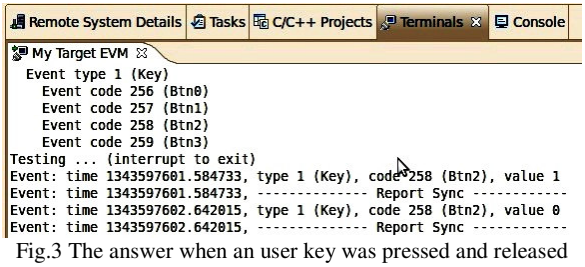

Fig.3 The answer when an user key was pressed and released

By inspecting this message it follows:

-the time moment (*time*) when the event was happened; by subtracting two such values, the time interval in seconds, between two actions can be determined;

-the key which was pressed (*code*); this field can have one of the values: Btn0 (code 256),…, Btn3 (code 259).

-the level (*value*) applied to the appropriate pin (in this example: 1, 0); or, in other words, what action

was happened: key *pressed* or *released* (by pressing the key, the level 1 is applied to the pin)

3. *Configuring the accelerometer and reading the 3-axis values.* There are two possibilities to do this function.

The first possibility allows returning the 3-axis value of the acceleration at the moment when the next Linux command is executed.

```
~#  cat  /sys/devices/platform/lis3lv02d/
position
```

The answer message can be like
(4, 12, 1096).
Also the *range* and the *rate* of the accelerometer can be read or set. The following Linux commands allow reading these values:

```
~#cat /sys/devices/platform/lis3lv02d/range
 2
~# cat /sys/devices/platform/lis3lv02d/rate
 50
```

The other possibility for accessing the accelerometer is the Linux command

```
~# evtest  /dev/input/event0
```

In this case, the answer message is displayed at any time when at least the value corresponding to one axis of the accelerometer is changed. The format is very similar with that of the user keys, as in Figure 4.

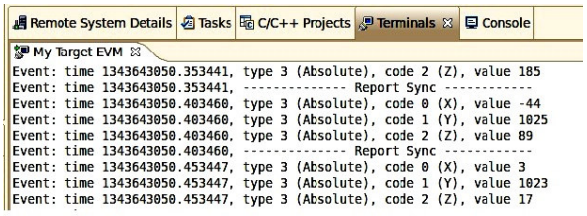

Fig. 4 The answer of the accelerometer

It this case, the field *code* represents the axis (0 for x, 1 for y and 2 for z) and *value* represents the acceleration of that axis.

Also, by looking to the format of the keys and that of the accelerometer, it can be seen that the field *type* has the value 1 (Key), and respectively 3 (Absolute). Another possible value of this field is 0, for synchronization (---Report Sync---). This field is sent between two successive results, and is visible only in C language.

In the following, some details about using C language to access these data are presented.
The program should contain 2 important steps:
-opening the file `/dev/input/event0` for read:

```
int  file  =  open("/dev/input/event0",
O_RDONLY);
```

-reading data from this file and send it in variable `ev`:

```
num_bytes=read(file,&ev,sizeof(struct
input_event));
```

The variable `ev` is of `struct input_event` type and has 4 components (the library linux/input.h):

```
struct input_event {
      struct timeval time;
      __u16 type;
      __u16 code;
      __s32 value;
};
struct timeval  {
    time_t         tv_sec;   //seconds
    susecond_t     tv_usec;  //microseconds
};
```

The components of this structure are in connection with the data returned by the accelerometer and presented above.
These components can be displayed, for instance by

```
print("%ld sec ",ev.time.tv_sec);
print("%ld usec  \t ",ev.time.tv_usec);
print("%u  %u  %d    \n  ",ev.type,ev.code,
ev.value);
```

4. *Reading the touchscreen values*
By executing the Linux command

```
~# evtest /dev/input/touchscreen0
```

the system is waiting as the touchscreen to be touched and as effect it answers by the coordinates (x,y) of the point which was touched. The coordinates have values in the range (0,0) - for the lower right corner - to (4095, 4095) - for the upper left corner. Also, in this case the display format is very similar to that of the user keys or accelerometer.

5. *Connect a device to I2C interface.*
We want to connect an I2C device to AM335x SK, using the connector J8. This device is an integrated circuit, AD5241, which contains three digital potentiometers having the next 7 bit addresses: 2Ch, 2Eh and 2Fh [7].

On the other hand it is known that ARM processor contains three I2C modules (I2C0, I2C1 and I2C2) and the system AM335x SK contains some devices, like accelerometer, audio codec, etc., connected to I2C0 module. The addresses of these devices are: 12h, 18h, 1Bh, 2Dh, 38h, 50h. It is important to verify that the addresses of our device are not overlapped with these addresses.

To do the desired connection, it is necessary to identify a suitable I2C adapter. This can be done by executing the command

```
~# i2cdetect -l
```

The answer is i2c-1, so the file having this name, which is within the directory *dev*, must be used to access the devices connected to the I2C interface.

ISBN: 978-960-474-349-0

Then, the two I2C (SDA and SCL) terminals of AD5241, as well as the two power terminals, must be connected to the connector J8 (see Appendix 1). Also two pull-up resistors must be connected between SDA and V3_3D, and, respectively, SCL and V3_3D.

Having these statements in mind, a C program must be created to control the digital potentiometers. The program should contain 3 important steps:

-opening the file /dev/i2c-1 for read and write. This can be done by the C sequence:

```
if ((file = open("/dev/i2c-1",O_RDWR)) < 0)
{printf("Failed to open the bus");
exit(1);
}
```

-establish the I2C address (addr1, 2Ch for instance) like in the next C sequence:

```
if (ioctl(file, I2C_SLAVE, addr1) < 0)
{   printf("Failed   to   establish   the
address");
exit(1);
}
```

This command has no effect after its execution, but the I2C address will be used in the next step, as it will be seen.

-then, a number of bytes (2 in our example, the first two elements of the array *buf*) can be written as in the following C sequence.

```
if (write(file,buf,2) !=2)
{ printf("Failed to write to I2C bus");
exit(1);
}
```

Actually this command sends to the I2C interface a first byte containing the I2C address (2Ch) appended with a bit having the level 0 (write) and then the two bytes of the array *buf*. The first of these bytes configures the potentiometer, and the last one establishes its dividing factor [7].

5 Using the Touch Screen to start the applications

To launch the Target applications, there is a most advanced option, further to that of using the Linux command on the Linux PC. Thus, it can be added new icons within the Matrix Application Launcher. These icons can launch applications by simple touch the screen. For this, two steps must be performed, as following.

1. *Create a new Matrix empty directory*. This directory has an icon that will be displayed among the Matrix Applications icons. To do this, a directory having two files must be created on the Linux PC. The first file is called *.desktop and its content must contain: the name of this icon, the path for the image file of the icon, the position in the Matrix Application where the icon will be placed. The second file is called *.img and it contains the icon. This directory must be moved on the Target (the path: /usr/share/matrix-gui-2.0/apps) using the facility of the CCS. After the executing of Refresh command (Settings/Refresh Matrix, within Matrix Application Launcher), the icon will be displayed on the LCD.

2. *Create a new application within the previous created Matrix directory (icon).* For this purpose, another directory containing 4 files must be created on the Linux PC. The first two files are very similar with those of the previous created directory. The third is called *.sh and contains the Linux commands that will be executed. These Linux commands can be similar to those presented in chapter 4 (for peripheral accessing or for executing C applications). The last is called *.html and its content will be displayed on the LCD before the execution of the application and therefore can contain a description of the application. Also in this case, the created directory must be moved within the same path of the Target and the Refresh command must be executed.

Then, when the icon of the directory is touched, the icon of the application can be seen. By touching this icon, the RUN icon and the content of the *.html file will be displayed. Finally, the application can be executed by touching the RUN icon. In this case, the execution results are displayed on the touchscreen of AM335x SK.

Fig. 5 presents the Matrix Application Launcher (upper panel), where the new created icon First_App can be seen, and three applications created within this icon (lower panel). Both images were obtained using the remote mode of the Matrix Application. In the first image, it can be seen the IP of the target, 192.168.1.132.

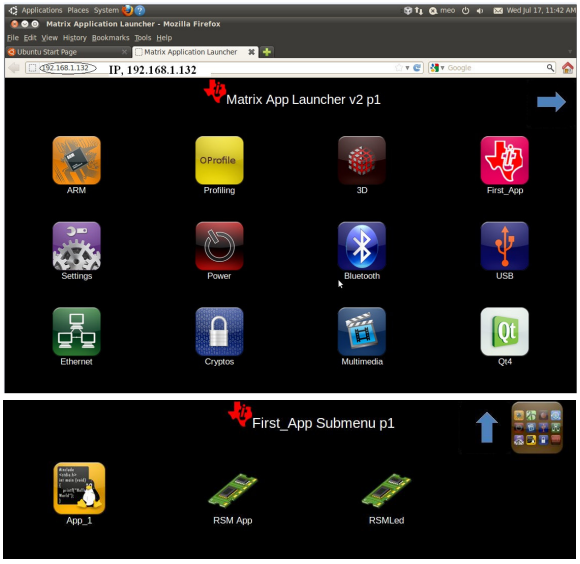

Fig.5 The new application, First_App

6 Conclusion

The paper presents the basics issues to be known to achieve applications for AM335x SK. For this purpose, it is necessary to use CCS on a Linux PC. Applications can be run using the Linux commands executed in Terminals tab of CCS but also by using the touchscreen of the AM335x SK.

In the future, these applications can be improved. Thus, the accelerometer will be used to display the orientation of the system, and the digital potentiometer connected by the I2C interface will be used to control a helicopter.

References:

[1] Ping Li ,Jian Ping Li, Embedded Intelligent Home Control System Based on Arm Linux, ICWAMTIP, 2012, pp. 429-431.

[2] Ting Zhang, Xu Chunxiu et all, Implementation of Ad Hoc Network management system based on embedded ARM-Linux platform, ICNDS 2012, pp.167-170.

[3] Xiang-hai, Jian-hui Liu, Design of embedded data acquisition and remote control system based on Linux, ICCASM 2012, vol. 3, pp. 299-302.

[4] B. Betea, L. Tomesc, P. Dobra, M. Trusca, Spectrum Analyzer Instrument based on OMAP3530, Proceedings of 5th European DSP Conference, 2012, pp.257-260.

[5] http://www.ti.com/tool/tmdssk3358.

[6] Shi Jun-Yong, Design and Implementation of Embedded GPS System, CSAE 2012 IEEE International Conference, pp. 311-314.

[7] http://www.analog.com.

Appendix 1

The Sitara AM335x system has two expansion connectors, J11 (for SPI0 interface) and J8 (for I2C interface).

The structure of the two connectors is presented in tables A1 and A2.

It is known that pins of an ARM-type processor which have input and output capabilities are grouped in 4 modules, each having 32 pins. Thus, the GPIO number presented in tables A1 and A2, represents the module and, respectively, the pin within that module. Example: gpio0_4 means pin 4 within the module 0. Using this number, the index that corresponds to that pin for using in Linux commands can be computed. For example, gpio3_5 has the index 3*32+5=101.

Table A1

Pin number	Function	Physical package pin number	GPIO number
1	V3_3D	-	-
2	SPI0_CS_0	A16	gpio0_5
3	SPI0_D1	B16	gpio0_4
4	GND	-	-
5	SPI0_CLK	A17	gpio0_2
6	SPI0_D0	B17	gpio0_3

Table A2

Pin number	Function	Physical package pin number	GPIO
1	GND	-	-
2	I2C0_SDA	C16	gpio3_5
3	I2C0_SCL	C17	gpio3_5
4	V3_3D	-	-

Appendix 2

Function	**The path of the file**	**Values that can be written**
GPIO pins	/sys/class/gpio/export /sys/class/gpio/unexport /sys/class/gpio/gpiox/direction /sys/class/gpio/gpiox/value	1,…,128 1,…,128 out or in 1 or 0
user leds	/sys/devices/platform/leds-gpio/leds/ am335x:EVM_SK:usr0/trigger … am335xEVM_SK:heartbeat/brightness	 none, mmc0, hearbeat,… 0 or 1
accelerometer	/sys/devices/platform/lis3lv02d/range /sys/devices/platform/lis3lv02d/rate /sys/devices/platform/lis3lv02d/position	2, 4, 8 50,100, 400, 1000 can only be read
accelerometer user keys touchscreen	/dev/input/event0 /dev/input/event2 /dev/input/touchscreen0	return automatically after an event
New Matrix applications	/usr/share/matrix-gui2.0/apps	-

ISBN: 978-960-474-349-0