Abdullah Almarzouq
2/10/20

Design Project #1

Introduction:

The Design Project #1 is a two-part project where each part has a its objectives in communicating the BeagleBone Black with the RC 8660 voice synthesizer. The first part of the project covers the development of a program that sends a basic message to the RC 8660 board on an interrupt basis. When a button (GPIO1_29) is pushed the program will send the message, byte after byte, using a UART. While the second part of the project covers the development of a program that a timer, same button, LEDs, UART and the RC 8660. For the second part, when the button is pushed, the LEDs will turn on and the timer will start counting up and when the button is pushed for the second time the LEDs will turn off and the timer will stop then the program will send the value of the timer to the RC8660, using the UART, and the speaker will announce the value of the timer between the first and second button push.

<u>PART 1:</u>

The first part of the project, as mentioned above, cover the development of a program that will send a message, byte by byte, to the RC 8660, using the UART, on an interrupt basis when the button is pressed.

Main Tasks:

- Obtain necessary addresses to the change the modes/mapping of pin MUX.
- Obtain necessary addresses to control the RC8660 device and UART5.
- Create the MAINLINE to initialize all necessary inputs, outputs and other parameters.
- Create other necessary branches to perform the following:
    - Respond to interrupt and decide if its from button or UART.
    - Check if button then perform necessary actions.
    - Check if UART then perform necessary actions.

High Level Algorithm:

MAINLINE:

Change the mapping of pin MUX from lcd_data to UART5
Enable GPIO1_29 (button) clock and detect falling edge
Initialize INTC
Initialize UART5 clock, set the baud rate, disable FIFOs

ENTER WAIT LOOP
If an interrupt happens then go to INT_DIRECTOR

INT_DIRECTOR:

```
Check if interrupt caused by UART:
    If yes, then check CTS# asserted low:
        If CTS# asserted low, then check if THR bit is 1, if yes then go to TLKR_SVC.
    Else, if CTS# is not asserted low then mask the THR bit and go back to WAIT LOOP.
Else if interrupt not from UART, then check if interrupt from button:
    If yes, then go to BUTTON_SVC, else go back to WAIT LOOP.
```

BUTTON_SVC:

```
Turn off new interrupt bits so that a new interrupt can happen.
Turn on the UART interrupt (enable THR, MODEM and CTS interrupts).
```

TLKR_SVC:

```
Load the message then send a character to RC 8660 and go back to WAIT LOOP.
If all character/bytes have been sent successfully then turn off UART5 interrupt and go back to
WAIT LOOP.
```

Low Level Algorithm:

MAINLINE:

```
Load base address 0x44E10000 of control module.
Change from mode 0 to mode 4:
    Load base address 0x44E10000 of control module.
    Write 0x04 to offset 0x8C0 (change to mode 4 ‘UART5_TxD’ by writing to lcd_data8)
    Write 0x24 to offset 0x8C4 (change to mode 4 ‘UART5_RxD’ by writing to lcd_data9)
Change from mode 0 to mode 6:
    Write 0x26 to offset 0x8D8 (change to mode 6 ‘UART5_CTS’ by writing to lcd_data14)
    Write 0x06 to offset 0x8DC (change to mode 6 ‘UART5_RTS’ by writing to lcd_data15)

Write 0x2 to 0x44E000AC to enable GPIO1 clock.
Load GPIO1 base address 0x4804C000:
    Read register FALLINGDETECT at offset 0x14C and OR it with 0x20000000 then write
    the result back to offset 0x14C to enable falling edge detect for bit 29 of GPIO1.
    Write 0x20000000 to offset 0x34 to enable GPIO1_29 request on POINTRPEND1.
Load base address 0x48200000 of INTC:
    Write 0x2 to INTC config register at offset 0x10 to reset INTC.
    Write 0x00004000 to register INTC_MIRCLEAR1 at offset 0xA8 to unmask INTC INT
    #46, UART interrupt.
    Write 0x04 to INTC_MIRCLEAR3 register at offset 0xE8 to unmask INTC INT #98
    GPIO1 interrupt.

UART5 setup:
    Write 0x2 to 0x44E00038 to enable UART5 clock
    Load UART5 base address 0x481AA000
    Set UART5 in mode A by writing 0x83 to offset 0x0C (UART5_LCR)
```

Set 38.4Kbps baud rate in DLL by writing 0x4E to offset 0x00 (DLL register)
Set 38.4Kbps baud rate in DLH by writing 0x00 at offset 0x04 (DLH register)
Set 16x divisor by writing 0x000 to offset 0x20 (MDR1 register)

Set UART5 in operational mode by writing 0x03 to 0x0C offset (LCR register)
Write 0x0000 to disable THRIT bit and MODDEMSTSIT bit at offset 0x04 (IER_UART5 register) since we are still in the MAINLINE.
Write 0x04 to offset 0x8 (FCR register) to disable FIFOs.

ENTER WAIT LOOP

INT_DIRECTOR:
Save register on STACK
Test bit 14 (0x00004000) with value in INTC-PENDING_IRQ1 register at 0x482000B8.
If bit 14=1 then go to UART5_CHK, else check if interrupt from button.
Test bit 29 (0x20000000) with value in GPIO1_IRQSTATUS_0 register at 0x4804C02C
If bit 29=1 then go to BUTTON_SVC, else go back to PASS_ON.

PASS_ON:
Clear bit 7 0x80 in CPSR
Write 0x1 to 0x48200048 INTC_CONTROL register to clear interrupts.
JUMP to WAIT LOOP

BUTTON_SVC:
Write 0x20000000 to 0x4804C02C to clear GPIO1_29 interrupt.
Write 0x000A to 0x481AA004 IER_UART5 register to enable THR and CTR interrupts.
JUMP to PASS_ON

UART5_CHK:
Test bit 5 (0x20) with value in MSR register at 0x481AA018 to check if CTS# asserted low.
If bit 5=0 then mask THR bit (0x20) at 0x481AA014 LSR register.
Else is bit 5=1 then check if THR bit is 1 by testing 0x20 with value in LSR register at 0x481AA014.
If THR bit is 1 then go to TLKR_SVC.
Else, if THR bit is 0 and CTS# is not asserted low then go to PASS_ON.

TLKR_SVC:
Load byte from message, increment message address and save address to message pointer.
Load value of counter (length of message).
Send byte to 0x481AA000 (UART5_THR) register.
Decrement counter and save value in counter pointer.
If counter’s value is not zero, then go to PASS_ON.

Else, reset the message pointer to the first character/byte of the message and reset the counter value.
Turn off UART5 interrupt by writing 0x0A to 0x481AA004 (IER_UART5 register).
Go to PASS_ON.

LOG:
1/16/20:-

- The MAINLINE of the program was created. The STACK has been set as before (using last term's programs). Also, the button (GPIO1_29) has be initialized and the UART5 inputs and outputs have been mapped successfully.
- INTC has been initialized for both GPIO1_29 and UART5.
- The GPIO1_29 initialization process and commands were taken from last term's programs and design projects to save time and effort since it is almost the same initialization process.
- After mapping UART5 inputs & outputs and turning its clock on, the UART5 has been set to mode A to change the and set the baud rate and the 16x divisor.
- The initialization process for both GPIO1_29 and UART5 mentioned above are what the MAINLINE and program consists of. While below the MAINLINE are branches that needs to be setup and initialized. These branches are : INT_DIRECTOR, PASS_ON, BUTTON_SVC, UART_CHK and TLKR_SVC.

1/17/20:-

- INT_DIRECTOR has been set to check if the interrupt was caused by button push or the UART5. IF interrupt from button then jump to BUTTON_SVC, else if from UART5, then jump to UART5_CHK.
- Branch UART5_CHK checks if CTS# bit was asserted low or not. IF CTS# bit is 1 then the program jump to branch THR_CHK, else if CTS# bit was 0 then jump to THR_MASK. The THR_MASK branch checks if the THR bit is one then set it to zero by masking it, else if THR bit is zero then jump to PASS_ON. While the THR_CHK branch checks if THR bit is one then jumps to TLKR_SVC, else jumps to PASS_ON.
- The BUTTON_SVC branch has also been setup and initialized so that it turn on the UART5 interrupt and then it goes back to the wait loop.
- The TLRK_SVC branch has also been set so that when the program jumps to it, it reads the message string from the literal pool and then it sends each character to the UART5.
- After the first build, the program has some typing errors, however, all of these errors were fixed as the program compiled successfully after that.
- After the first run, it turns out that some of the addresses are not loading correctly into some of the registers, yet, this problem was fixed by changing the register number.
- After that, another problem occurred where the program keeps jumping to INT_DIRECTOR because the UART5 interrupts were turned on in the MAINLINE before the button was pushed. The solution to this problem was to turn off the UART5 interrupt in the MAINLINE.

- Now, after fixing all these problems, the program responds successfully to the interrupts from the button and it turns on the UART5 interrupts which are working because the program jump to INT_DIRECTOR then to UART5_CHK as the interrupt was caused by the UART5. However, the CTS# signal never gets asserted low as bit 4 in MSR register is zero and it never turns to one for some reason.

1/23/20:

- The CTS# still not working, however, when loading the value from MSR register it appears to be 0x20 instead of 0x01. Therefore I changed the testing bit from 0x01 to 0x02 just to let the program go through and act as if CTS# was asserted low. After doing so, the program went through to the TLKR_SVC and it turns out that the program works! The talker program can talk when the characters are being sent through the UART.
- Another problem occurred where the program only talks when an interrupt happens for the first time and when the button is being pushed again it doesn't talk. This problem was being caused because after the first time the program finish talking, the value of CHAR_COUNT (message length) doesn't get loaded back correctly (after the first time). The value that is being loaded back is the address of CHAR_COUNT instead of the value.
- To fix this problem, I added another CHAR_COUNT (CHAR_COUNT2) so that, after the program finish sending all the characters, the address of CHAR_COUNT2 can be loaded into a register then the value of CHAR_COUNT2 can be loaded from that address back into the original CHAR_COUNT so it can be used again when the button is pushed for the second time and third time and so on.
- After confirming that the program can talk by sending characters every time we click the button, I changed the voice of the speaker from default to VADER. The process was simple as I managed to create the command that changes the voice in the literal pool just like the initial test message. Now the program sends the command characters and bytes to change the voice (just like sending a test message) inside a branch called VADERV. Then after executing branch VADERV, the program goes to TLKR_SVC to send speech characters without going through VADERV branch again if the button was pushed for more than one time.

PART 2:

The second part of the design project, as mentioned in the introduction, covers the development of a program that uses the same button, UART as the first part. In addition to a timer (Timer2) and the USR LEDs. The purpose of the program is to use RC 8660 to announce the time between the first button push and the second button push. To be more specific, when the button is pushed for the first time, the LEDs and the timer will turn on and when the button is pressed again, the timer will stop, and the LEDs will turn off. After that the program will send the value of the timer to the RC 8660 to announce it (in HEX).

(Note: The program in the first part will be used to build this program, only the necessary additions will be made to the program from the first part to match the required specifications for the second part's program to be built successfully.)

Main Tasks:

- Obtain the necessary addresses to initialize the timer (Timer2).
- Obtain the necessary addresses to turn on/off the LEDs.
- Create a branch that performs the following:
    - Turn off LEDs.
    - Turn on UART.
    - Convert the Timer value from HEX to ASCII.
- Add the following instructions:
    - If the button was pushed for the first time then turn on LEDs and start the timer.
    - If the button was pushed for the second time, then turn off LEDs, stop the timer and turn the UART.
    - Convert the timer's Hex value to ASCII to be sent to the RC 8660 through the UART.

High Level Algorithm:

(Note: since the program from the first part will be used, only the necessary/required instructions will be mentioned in both High Level Algorithm and Low Level Algorithm.)

```
MAINLINE:
Initialize the timer (turn on clock and initialize the count value).

INT_DIRECTOR:
Check if interrupt from button or UART:
    If from button, then go to BUTTON_SVC.
    Else if from UART5 then go to UART5_CHK.
    Else, go to PASS_ON.

BUTTON_SVC:
If button was pushed for first time:
    Start Timer2 to count up and auto reload.
```

Turn on USR LEDs.
Go to PASS_ON.

Else, if button was pushed for second time:

Go to OFF branch.

OFF:

Stop timer and obtain the value.

Convert each byte of the timer's value from HEX to ASCII and save it in the MESSAGE string.

Reset timer to zero.

Turn on UART5 interrupts to start sending timer value to RC 8660.

Go to PASS_ON.

Low Level Algorithm:

MAINLINE:

Write 0x2 to 0x44E00080 to turn on clock of Timer2.

Write 0x2 to 0x44E00508 (PRCMCLKSEL_TIMER2 register) to select 32 Khz CLK.

Load base address of Timer2 register 0x48040000:

Write 0x1 to offset 0x10 (Timer2 CFG register) to reset the timer.
Write 0x00000000 to offset 0x40 (Timer2 TLDR register) to store the reload value.
Write 0x00000000 to offset 0x3C (Timer2 TCRR register) to store the count value.

INT_DIRECTOR:

Same as the INT_DIRECTOR branch in the first part

BUTTON_SVC:

If button pressed for first time:

Write 0x03 to 0x48040038 (Timer TCLR register) to turn on the timer (Start counting up).
Write 0x01E00000 to 0x4804C194 (GPIO1_SETDATAOUT register) to turn on the LEDs.
Write 0x0 to 0x481AA004 (IER_UART5 register) to disable UART5 interrupts.
Go back to wait LOOP

Else if button pressed for second time:

Go to OFF branch.

OFF:

Write 0x00 to 0x48040038 (Timer2 TCLR register) to stop Timer2.

Read value in TCRR register (0x4804003C) and save it in a register.

Load the address of the message from the literal pool into a register and point the register at the end of the message.

Load 0x0000000F into a register to mask the timer value to be able to convert a byte from HEX to ASCII.

Initialize a register (i.e R5) with 0x0 to be used to shift the masked value.

REPEAT
AND timer value with mask.
Shift the masking value for bits to the left.
Shift the masked value with the number of bits stored in R5 to the left.
Increment R5 by 4 to be used to shift the next value.

If the masked value is larger than or equal to 0xA, then add 0x37 to it to convert it to ASCII.
Else, if the value is smaller than 0xA, then add 0x30 to convert it to ASCII.

Store the masked value/byte in the message address stored previously in a register.
Decrement the message address pointer by one byte to point to the next location to save the next converter byte.
Decrement counter
REPEAT until the counter value is zero.

Write 0x01E00000 to 0x4804C190 (GPIO1_CLEARDATAOUT register) to turn off the LEDs.
Write 0xA to 0x481AA004 (IER_UART5 register) to enable UART5 interrupts.
Write 0x00000000 to 0x4804003C (Timer2 TCRR register) to reset the count value.
Go to PASS_ON.

LOG:
1/31/20:

- The algorithm for the second program has been written.
- The LEDs bits (21-24) has been set as outputs on GPIO1. Now all the LEDs on once the button is pressed.
- The timer (Timer2) has been set and initialized with the value 0x00000000 which has been stored in the TCRR register.
- Now the LEDs and the timer are switched on once the button is pressed.
- A branch called OFF has been added so that, if the button is pushed for the second time, the program jumps to OFF branch to stop the timer and turn off the LEDs and the timer.
- The value of the timer in TCRR register is now being stored in register R6 inside the OFF branch. Once the timer stops, the count value is stored in R6 and the value in TCRR is restored to 0x00000000.
- Now, the tasks that need to be done and worked on are:
    - Turning on the speaker when the button is pushed for the second time. The speaker will also need to receive the timer value (stored in R6) as an ASCII message.
    - The speaker needs to say the value letter by letter or number by number.

2/3/20:

- The speaker is now turned on when the button is pushed for the second time, while the LEDs turn off when the button is pushed for the second time.

- To let the speaker say the Hex value, I had to add a branch that retrieve a byte from the value of the timer (saved in R6) by ANDing R6 with 0x0000000F then shift the mask 4 bits to the left so that it can be used to retrieve the next value.
- Once a timer value (byte) has been retrieved by masking, it goes into a branch called HtoA to convert the Hex byte to ASCII character. If the Hex byte is larger than or equal to 0xA then add 0x37 to convert it to ASCII. Else if the retrieved Hex byte is less than 0xA then add 0x30 to convert to ASCII.
- Once a value has been converted into ASCII, it gets stored in the ASCII message address stored in the literal pool.
- After converting and storing all the values, the UART5 turns on, then the timer value gets restored to zero. After that, the program sends the desired characters to the RC 8660 then the speaker announces the timer value in Hex, letter by letter.
- There's also a command that has been added to make the RC 8660 pronounce the message letter by letter. The command is being sent to the RC 8660 along with the other command that changes the voice under the VADERV branch.

I development and wrote this program by myself with no help from anyone except the instructor and/or the TA. I did not give any help to anyone else and I did not copy anything from online sources.

Signature: Abdullah Almarzouq