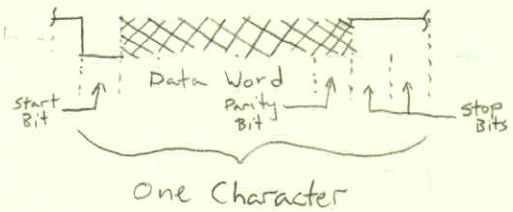
7/9 * Button is on GPIO1_30

- Chapter 5 Notes (ECE 371 textbook, pages 251-255) on RS-232C

1. a. - Asynchronous serial data is sent on the line bit-by-bit starting with the LSB.
- When no data is being sent, the line is in a constant high state (MARKING STATE)
- The start of a data character is indicated by one low bit (START BIT)
- The start bit is followed by the DATA WORD, which is 5, 6, 7, or 8 bits depending on convention
- Data word can be followed by an optional PARITY BIT (used to detect single-bit errors in data words)
- 1 or 2 always high STOP BITS follow data and parity bit (if present)



Bit format for sending 5-bit asynchronous serial data

b. - BAUD RATE is the rate data bits are being sent serially, and is defined as $\frac{1}{\text{time between each bit (time in seconds)}}$.

- Baud rate doesn't correspond to bits/sec because of the start and stop bits sent with the data bits added to the data bits.

Baud Rate Clock: see Part 2 on page 3

On 255, Baud R clock is a division of UART function clock, which is a division of system c

c. - Data Communication Equipment (DCE): Devices like a MODEM used to convert the serial data and send it over lines (bit-by-bit).
- Data Terminal Equipment (DTE): The computer/terminal that produces the serial data.
- standard RS-232C was created by the Electronics Industries Association (EIA) and describes the function of 25 "handshake" signals for serial data transfer, as well as data line properties.

d. 9-Pin Connector between DTE and DCE (Computer and Modem respectively):

	Pin#	Signal Name (Common Name/RS-232C Name)	DTE Direction	DCE Direction	Function
	1	Carrier Detect ($\overline{CD}$/CF)	IN	OUT	Received Line Signal Detector (from remote Mode
★	2	Received Data (RXD/BB)	IN	OUT	Modem telling Terminal data was received from it.
★	3	Transmitted Data (TXD/BA)	OUT	IN	Terminal telling Modem data was sent to it.
	4	Data Terminal Ready ($\overline{DTR}$/CD)	OUT	IN	Terminal power on & ready
	5	Ground (GND/AB)	—	—	
	6	Data Set Ready ($\overline{DSR}$/CC)	IN	OUT	Modem power on & ready
★	7	Request To Send ($\overline{RTS}$/CA)	OUT	IN	Terminal ready to send data
★	8	Clear To Send ($\overline{CTS}$/CB)	IN	OUT	Modem ready to send data
	9	Ring Indicator	IN	OUT	

★ = Signals used in project

7/10 • Chapter 5 notes continued

e. - When the terminal is ready to send data to the modem (DTE to DCE), DTE asserts Request To Send ($\overline{RTS}$) to the modem.
- When the Modem is ready to receive data from the terminal to send to a remote modem (DCE to DTE), DCE asserts Clear To Send ($\overline{CTS}$).
- If the modem is no longer ready at any time to send data to a remote modem, it will send a logic high signal to $\overline{CTS}$.
- The above "handshaking" between DTE and DCE is the method of data flow control on a serial data link.

f. Straight Though connection for DTE to DCE
- TXD output from DTE connects to TXD input of DCE, RXD input of DTE connects to RXD output of DCE, etc.
- Connections are done with 9-pin or 25-pin cables ⟹ don't worry about connections, just plug cable into DTE and DCE

g. Null MODEM connection for DTE to DTE or DCE to DCE
- Cross the following signals when connecting terminal-to-terminal or modem-to-modem, or signal 1 of device 1 to signal 2 of device 2 and vice versa. Ground connects to ground between devices
  $\overline{TXD}$ & $\overline{RXD}$
  $\overline{RTS}$ & $\overline{CTS}$
  $\overline{DTS}$ & $\overline{DSR}$
- Null modem cable allows user to not worry about connections and simply connect both devices, similar to straight through cable.

h. Interact with UART
- Included in Part 3
- UART5 system steps (pg 256-261)
  1. Set up stacks (see ECE 371 project #2, Example code on pg 237 in ECE 371 text and supporting text on pg 2)
  2. Set up GPIO1 for falling edge of button signal on GPIO1_30 (same reference as step 1.)
  3. Initialize INTC for GPIO1_30 interrupt
  4. Initialize INTC for UART2 interrupt. Use appendix E in ECE 371 text for int #, then map to appropriate INTC_MIR_CLEAR register (same procedure as GPIO1_30)

* Pg 256-260 (ECE 571 textbook, chapter 5): Steps to initialize UART, int direction section, button service section, and talker service section

7/2 (2.) Auto-Baud Rate:

- BRD (page 7/55): Band Rate Detection signal, used to determine baud rate by taking sample of seri stre
- BRS_0-BRS_3 (page 7): Baud Rate Select signals, used to set baud rate automatically for Receive data (RXD) and transmit data (TXD) pins.
  Tie pins logic high if not used. Tie high to auto-detect baud rate.
  always tie high

From RX pin tie high if not us

- Page 10: The serial ports baud rate can be programmed using auto-detect, which enables the serial port to automatically detect the baud rate to automatically detect the baud rate of the incoming data.

Auto baud rate detection is enabled when BRS_0-BRS_3 pins are all set high and the BRD pin is connected to the RXD pin. Baud rate is determined by shortest high-to-low period detected in input stream.

The CR character (0x0D) is recommended for locking the baud rate

Measurement cycle ends when there have been no high-to-low or low-to-high transitions on the BRD pin for at least 75 ms.

All pg# from AM335X tech ref

(3.) UART (button interrupt) (registers on page 4374 in AM335X technical reference)

AM335X Tech Ref: UART: pg 45 UART Reg: pg 45

UART = Universal Asynchronous Receiver/Transmitter

- Enable interrupt in interrupt enable register (IER). Must be in UART mode (pg. 4325)
- Read interrupt identification register (IIR) For interrupt source. (pg. 4325)
- UART2 base address (pg 181, AM335X ref): 0x48024000 (start address)
- UART2 clk enable (pg 1274): CM_PER_UART2_CLKCTRL
  - enable with 0x2
- UART2 int # (pg. 545): UART2INT = 74, signal name = nig
- UART2 clk enable address (addr offset, pg. 1250): 0x48024000 + 0x70 = 0x48024070
- CM_PER base address: 0x44E00000 (pg. 179)

(4.) ★ To enable UART2,

Write 0x2 to CM_PER_UART2_CLKCTRL register at ~~0x48024070~~ 0x44E00070

- UART2 IER address: 0x48024000 + 0x4 offset (pg 4374)
- UART2 IIR address: 0x48024000 + 0x8 offset (pg 4374)
- UART2 MSR address: 0x48024000 + 0x18 (pg 4374)
- UART2 LSR address: 0x48024000 + 0x14 (pg 4374)

7/11 4. continued

- Turn on UART2 clk: Write 0x2 to CM_PER_UART2_CLKCTRL @ 0x44E00070

- MIR0: 0-31
  MIR1: 32-63
  MIR2: 64-95

→ INTC_MIR_CLEAR2: 0x48200000 + 0xC8 (pg 581) (pg. 183)

★ Write 0x400 to INTC_MIR_CLEAR2 @ 0x482000C8 to unmask int #74

bit #	95 92	91 88	87 84	83 80	79 76	74 72	71 68	67 64
bit val	0 ←					0 1 0 ←		→ 0
Hex	0	0	0	0	0	4	0	0

7/12 5. Initialize UART2

- To change Modes: Write to Line Control Register (LCR) at address 0x4802400C (0xC offset, pg 283 371 textbook)(UART2 base addr)

For Operational mode, write 0x00 to 0x4802400C

Value	x xxx	x x x x	x x x x	x x x x	x x x x	x x x x	0 xx x	0 0 0 0
bit #	31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
Hex	F	F	F	F	F	F	0	0

x = don't ca

For Configuration mode A, write 0x83 to 0x4802400C. 0x83 because bit 7 must b, and data format selection is 0x83, so 0x83 meets both requirements (pg 258 in 371 textbook)

★ To select operational mode, write 0x00 to UART2 LCR @ 0x4802400C
To select Configuration Mode A (to select Band rate), write 0x83 to UART2 LCR

- To set band rate: write 0x00 to DLH (Divisor Latches High) @ 0x48024004 (offset 0x4, pg 283 in 371 textbook) in Config mode A.
  Then write 0x4E to Divisor Latchs Low (DLL) register (same as DLH) @ 0x48024000 in config mode A.
  (Above steps do first two clock speed divisions of functional clock 48MHz

- To set 16x divisor: write 0x00 to Mode Definition Register 1 (MDR1) @ 0x48024020 (0x20 offset, pg 283 in 371 text)

- To switch back to operational mode: write 0x03 to the LCR @ 0x4802400C

- To enable UART2 interrupts: Write 0xA to IER_UART register @ 0x48024004 (ints for THR & Modem status) (offset 0x4, pg 283 in 371 text) (pg 4381, BBBtech ref)

To disable, write 0x0

7/12 Initializing UART2 continued

- Disable FIFO: Write 0x6 to FIFO Control Register (FCR) @ 0x48024008
  clear & disable TX_FIFO & RX_FIFO
  Table 19-40, pg 4387 in AM335X tech ref: bits 0-3 were to clear & disable FIFO

- In main, enable IRQ interrupt in CPS (like in 371, project 2) using following code: (pg 237 in 371 text)

```
MRS Rx, CPSR
BIC Rx, #0x80      Rx = some register selected during coding
MSR CPSR_c, Rx
 ↑ upper case      ↑ lower case
```

- Set up supervisor and interrupt stacks (pg 237, 371 text)
  - Write 0x12 to CPS for IRQ mode, 0x13 for SVC (SVC by default)
  - Set up stacks in RIB, don't forget to allocate space in memory at end of code
  - Copy from pg 237

- Set up falling edge detect for button on GPIO1_30
  - Turn on GPIO1 module clock: Write 0x2 to CM_PER_GPIO1_CLKCTRL @ 0x44E000AC
  - Enable falling detect

Bit map falling detect word for GPIO1_30

bit#	31	30	27 … 24	23 … 20	19 … 16	15 … 12	11 … 8	7 … 4	3 … 0
bit val	0	1	0	0	0	0	0	0	0
hex	4		0	0	0	0	0	0	0

GPIO1_FALLINGDETECT address: 0x4804C000 + 0x14C = 0x4804C14C (pg 237, 371 text)

Read-Modify-Write (RMW) above bitmap to GPIO1_FALLINGDETECT address to enable falling detect on bit 30 only. This is done by "ORing" 0x40000000 with 0x4804C14C.

7/13

- Initialize INTC for GPIO1_30 interrupt & enable GPIO1_30 as interrupt source

  - Write 0x40000000 (see bitmap on pg 5 for bit 30) to GPIO1_IRQSTATUS_SET_0 at address 0x4804C000 + 0x34 = 0x4804C034 to produce IRQ on POINTRPEND1
    No RMW → writing 0 has no effect.

  - bitmap for INT #98 (pg 280, 371 text) to enable button int on POINTRPEND1

INT#	127 ... 124	123 ... 120	119 ... 116	115 ... 112	111 ... 108	107 ... 104	103 ... 100	98	96
bit val								0 1 0	0
hex	0	0	0	0	0	0	0	4	

MIR0: 0-31
MIR1: 32-63
MIR2: 64-95
MIR3: 96-127

  - Write 0x4 to INTC_MIR_CLEAR3 @ 0x48200000 + 0xE8 = 0x482000E8 to unmask (set to 1) int #98 and allow INTC to generate IRQ from button (pg 233 & 237 in 371 text)

7/14

- UART2 Mapping (TXD, RXD, CTS, RTS)

  - Completed by changing the mode of the MUX that selects signals (HDMI, UART, etc.) that go to the pins on the P8 and P9 pads.

- P9: uart_txd ⟹ MODE1, Pin 21 (Page 86, BBB sys ref man) Mode 0: (spi0_d0)
  uart_rxd ⟹ MODE1, Pin 22 " (spi0_sclk)

- P8: uart2_ctsn ⟹ MODE6, Pin 37 (Page 84, BBB sys ref man) (lcd_data8)
  uart2_rtsn ⟹ MODE6, Pin 38 " (lcd_data9)

  - Control module base address: 0x44E10000 (pg 273, 371 text)

  - Control module register offsets:

    CTS: conf_lcd_data8, 0x8C0 (pg 1457, BBB tech ref)
    RTS: conf_lcd_data9, 0x8C4 "
    TXD: conf_spi0_d0, 0x954 (pg 1458, BBB tech ref)
    RXD: conf_spi0_sclk, 0x950 "

  - To map control registers, (pg 1512, BBB Tech ref) register format

    Bit 5: write 1 for input, 0 for output
    Bits 0-2: mode select pins ⟶ mode 1, 001
    mode 6, 110

    For bit 5, assuming control module DTE.

signal name	bit 5 val	
TXD	1	(input)
RXD	0	(output)
CTS	0	(output)
RTS	1	(input)

7/14 - Bitmaps for UART to connector mapping (MUX)

TXD

bit #	7	6	5	4	3	2	1	0
bit val	0	0	1	0	0	0	0	1
hex	2				1			

RXD

bit #	7	6	5	4	3	2	1	0
bit val	0	0	0	0	0	0	0	1
hex	0				1			

CTS

bit #	7	6	5	4	3	2	1	0
bit val	0	0	0	0	0	1	1	0
hex	0				6			

RTS

bit #	7	6	5	4	3	2	1	0
bit val	0	0	1	0	0	1	1	0
hex	2				6			

⇒ RMW (OR, set bits) by,

Write 0x6 to 0x44E108C0 for CTS signal
Write 0x26 to 0x44E108C4 for RTS signal
Write 0x1 to 0x44E10950 for RXD signal
Write 0x21 to 0x44E10954 for TXD signal

- UART2 interrupt status & interrupt set (IIR_UART2)

bit 0: 0x0, IRQ pending (pg. 4385, BBB tech ref)
0x1, IRQ NOT pending

bit 1: 0x0, modem int (priority 4)
0x1, THR int (priority 3)

Bits 1-5 are for if any of the multiple UART ints are set. Only checking bit 1 since only modem and THR ints are enabled. See pg 4385.

7/18 - Checking Interrupt source

Button: INT #98 → Test 0x4 (bit #2 (see bitmap on pg 6 of this log)) in INTC_PENDING_IRQ3 at 0x48200000 (INTC base, 276 311 text) + 0xF8 (pg 281, 37 tex)

UART2: INT #74 → Test 0x400 (bit #10 (see bitmap on pg 4 of this log)) in INTC_PENDING_IRQ2 at 0x48200000 + 0xD8

- Confirming button was pressed

Test 0x40000000 (bit #30) in GPIO1_IRQSTATUS_0 at 0x4804C000 + 0x2C (pg 4998, BBB tech ref)

-Confirm if interrupt from Transmit Holding Register (THR) or Modem Status Reg (MSR) of UART2

7/19 -Confirm if interrupt from THR or MSR in UART2
Read UART2_IIR at 0x48024008 (see pg. 7 of this log under "UART2 int status...")

UART2 base addr: 0x48024000

- Read bit 4 (0x8) of MSR (0x18 offset) if $\overline{CTS}$ has been asserted (pg 4399, BBB tech ref)
  (=1 if $\overline{CTS}$ asserted) ← (pg 4374, BBB tech ref)
- Read TXFIFOE bit (bit 5 = 1 if THR empty) of LSR (0x14 offset) if THR empty (pg 4396, BBB tech re

- To send message to talker board (after $\overline{CTS}$ asserted and THR empty)
  - Place following code at end of program to set up the message (in ASCII), next character pointer, and message character count.

```
.data
MESSAGE:  .byte 0x0D
.ascii "Type message here, between quotes"
.byte 0x0D
.align 4
CHAR_PTR:  .word MESSAGE
CHAR_COUNT:  .word        (# of chars in message. ex: 21)
.end
```

7/20 • Turn off button IRQ (button_svc)

Write 0x40000000 to GPIO1_IRQSTATUS_0 at 0x4804C02C (pg 4993, BBB tech ref)

• Enable new IRQs in INTC (last step before returning from button_svc)

Write 0x1 to INTC_CONTROL at 0x48200048

7/15 High Level Algorithm (Work in Progress, program 1)

Mainline
1. Set up stacks for SVC and IRQ modes
2. Enable GPIO1 module clock
4. Set up falling edge detection on GPIO1_30 (push button)
5. Enable GPIO1_30 as interrupt source
6. Enable UART2 as interrupt source
3. Enable UART2 module clock
7. Initialize INTC and unmask button interrupt on INTC #98, UART2 interrupt on INTC #74
8. Set UART2 baud rate
9. Set UART2 16× divisor
10. Disable UART2 FIFO
11. Change pin MUX mappings on BBB P8/P9 connectors from HDMI framer to UART2 signals
12. Enable IRQ input by clearing bit 7 in CPSR
13. Wait for interrupt

user hook IRQ procedure in BBB

INT_DIRECTOR (pg 259, 371 text)
1. Save used registers, including Link register (R14)
2. Check if interrupt from UART2 signal in INTC
3. IF YES, check if interrupt from UART2 THR and MSR
   IF YES, go to TALKER_SVC
   ELSE enable new interrupt and go back to wait loop
4. ELSE check interrupt from button signal in INTC
5. IF YES, check if interrupt from button pin
   IF YES, go to BUTTON_SVC
   ELSE enable new interrupt and go back to wait loop

BUTTON_SVC
1. Turn off button interrupt (bit #30, GPIO1_30)
2. Enable new IRQ in INTC.
3. Enable THR and MSR interrupts in IER_UART2 (bits 1 & 3 respectively)
4. Return to wait loop.

TALKER_SVC
1. Check if $\overline{CTS}$ is asserted
2. Check if THR is empty
   IF $\overline{CTS}$ asserted and THR empty, write character to UART Transmit Buffer
   IF $\overline{CTS}$ NOT asserted and THR empty, disable THR interrupt
   IF $\overline{CTS}$ NOT asserted and THR NOT empty, enable new IRQ and return to wait loop
3.

7/20

Low Level Algorithm (draft)

(Modify startup.s file using "b INT_DIRECTOR") (See pg 235, 371 text)

Mainline

1. Set up stacks for svc and irq modes
   - Load stack pointers
   - Point to top of stacks (full descending)
2. Enable GPIO1 module clock
   - Write 0x2 to CM_PER_GPIO1_CLKCTRL at 0x44E000AC
3. Enable UART2 clock
   - Write 0x2 CM_PER_UART2_CLKCTRL at 0x44E00070
4. Set up falling edge detection on GPIO1_30 (push button)
   - Load word from GPIO1_FALLINGDETECT at 0x4804C14C
   - Set bit 30 by ORing word with 0x40000000
   - Write result back to GPIO1_FALLINGDETECT
5. Enable GPIO1_30 as interrupt source
   - Write 0x40000000 to GPIO1_IRQSTATUS_SET_0 at 0x4804C034
6. Enable INTC to respond to an IRQ on POINTRPEND1
   - Write 0x4 to INTC_MIR_CLEAR3 at 0x482000E8 to unmask INT#98
7. Enable INTC to respond to an IRQ on UART2
   - Write 0x400 to INTC_MIR_CLEAR2 at 0x482000C8 to unmask INT #74
8. Set UART band rate
   - Change to mode A by write 0x83 to UART2_LCR at 0x4802400C
   - Write 0x00 to UART2_DLH at 0x48024004
   - Write 0x4E to UART2_DLL at 0x48024000
9. Set UART 16x divisor
   - Write 0x00 to UART2_MDR1 at 0x48024020
   - Write 0x03 to UART2_LCR at 0x4802400C to switch back to operational mode
10. Enable UART2 interrupts (THR and Modem Status) — Move step to BUTTON_SVC, next page
    - Write 0xA to IER_UART2 at 0x48024004 (bit 1=THR, bit 3=Modem → 1010)
11. Disable UART2 FIFO
    - Write 0x6 to FCR_UART2 at 0x48024008
12. Enable IRQ input in CPSR (Rx ⇐⇒ register # TBD)
    - MRS Rx, CPSR
    - BIC Rx, #0x80
    - MRS CPSR_c, Rx
13. Map UART2 TXD, RXD, $\overline{CTS}$, and $\overline{RTS}$ to BBB P8 and P9 MUX
    - RMW (ORing to set bits),
    - Write 0x6 to UART2_CTSN at 0x44E108C0 for CTS signal
    - Write 0x26 to UART2_RTSN at 0x44E108C4 for RTS signal
    - Write 0x1 to UART2_RXD at 0x44E10950 for RXD signal
    - Write 0x21 to UART2_TXD at 0x44E10954 for TXD signal
14. Wait for interrupt (Dead loop, wait for button press)

```
LOOP:
    NOP
    b LOOP
```

7/22 Low Level Algorithm continued

INT_DIRECTOR:

1. Push R0-R10, R14 to stack
2. Check if IRQ is from UART2
   Test 0x400 (bit 10, INT #74) in INTC_PENDING_IRQ2 at 0x482000D8
3. IF YES, check IT_PENDING bit (bit 0) in IIR_UART2 at 0x48024008 (0 → IRQ Pending)
   IF bit=0, go to TALKER_SVC
   ELSE check if IRQ is from button
4. Check if IRQ is from button
   Test 0x4 (bit 2, INT#98) in INTC_PENDING_IRQ3 at 0x482000F8
5. IF YES, check GPIO1 pin
   Test 0x40000000 (bit 30) in GPIO1_IRQSTATUS_0 at 0x4804C02C
   IF pin, go to BUTTON_SVC

   ELSE enable new IRQs and return to wait loop
   Write 0x1 to INTC_CONTROL at 0x48200048
   Pop registers R0-R10, link register (LDMFD SP!, {R0-R10, LR})
   Go back to last instruction (SUBS PC, LR, #4) in svc mode

BUTTON_SVC:

1. Turn off button interrupt
   Write 0x40000000 to GPIO1_IRQSTATUS_0 at 0x4804C02C
2. Enable new IRQ in INTC
   Write 0x1 to INTC_CONTROL at 0x48200048
3. Enable UART2 interrupts (THR and MSR)
   Write 0xA to IER_UART2 at 0x48024004
4. Return to wait loop
   Pop registers R0-R10, LR
   Go back to last instruction in SVC mode (dead loop)

Moved from Mainline, previous pg

7/22 Low level algorithm continued (TALKER_SVC)

TALKER_SVC (see fig 5-24 in 371 text, flowchart) (See "SEND"

1. Check if $\overline{CTS}$ is low
   Test 0x10 (bit 4) in MSR_UART2 at 0x48024018 (bit 4=1 ⟶ $\overline{CTS}$ asserted)
   If bit 4=1; check if THR is empty
   ELSE bit 4=0, BEQ NOCTS, check if THR is empty
2. Check if THR is empty
   Test 0x20 (bit 5) in LSR_UART2 at 0x48024014 (bit 5=1 ⟶ THR is empty)
   If bit 5=1, send character (to SEND)
   ELSE return to wait loop and enable new IRQ
3. NOCTS: branch
   Test 0x20 (bit 5) in LSR_UART2 at 0x48024014
   IF bit 5=5 (THR is empty, $\overline{CTS}$ not asserted), Write 0x8 to (disable) mask THR IRQ in IER_UART at 0x48024004 and return to wait l
   ELSE return to wait loop and enable new IRQ
4. SEND ($\overline{CTS}$ asserted, THR empty)
   Create a message character pointer CHAR_PTR and message character count CHAR_COUNT in .data at end of code (allocate space in memory)
   Load CHAR_PTR that points at next character to send.
   Load CHAR_COUNT to get # of characters left to send.
   Load current character to send from CHAR_PTR, and post increment to next character
   Point to next character in CHAR_PTR
   Decrement CHAR_COUNT and store in memory (.data)
   Store current character to send in THR_UART2 at 0x48024000
   IF no more characters,
      Load message into CHAR_PTR (Point at first character again)
      Load character count into CHAR_COUNT (message length)
      Disable UART2 interrupts
         Write 0x0 to IER_UART2 at 0x48024004
      Enable new IRQ and return to wait loop
   ELSE enable new IRQ and return to wait loop

7/23 Timer 2 Notes

TIMER control Register TCLR (pg 4457-4458) } BBB Tech Ref
pg 4448

- TLDR and TCRR
  CLK = 32768 KHz = 0x8000 (for 1 second)
  10 sec ⟹ 32768 × 10 = 327680 = 0x50000
  (pg 4461, BB tech ref) TTGR (offset 0x44
  Write 0xFFFB0000 (timer value

  = 0x100000000 − 0x50000 = 0xFFFB0000

  Write 0xFFFB0000 at 0x48040040 (Timer2 TLDR) and 0x4804003C (Timer2 TCRR)

- Timer 2 registers (name & address) and utilization steps (initialization, etc) taken from ECE 371 book, pg

RC8660 Talker

number 0-10 "oh"

- To change voice; use syntax #0 at the beginning of message inside quotes?

  Darth Vader voice: CTRL+A "10" "message" (pg 20 RC8660 ref) for cmd char (pg 22 RC8660 ref) for voice

8/3: Per discussion with Tyler,

Updated UART2 mapping: (signals from BBB's perspective)

	bit 5 value	Write Value
TXD	0 (output)	0x1
RXD	1 (input)	0x21
CTS	1 (input)	0x26
RTS	0 (output)	0x6

⟹ RMW (OR, set bit),

Write 0x26 to 0x44E108C0 (CTS)
0x6 to 0x44E108C4 (RTS)
0x21 to 0x44E10950 (RXD)
0x1 to 0x44E10954 (TXD)

7/23 Part 2 High Level Algorithm

Mainline
1. Same as Part 1
2. Initialize Timer2 interrupt (INTC #68)
3. Turn on Timer2 clock
4. Set Timer2 clock frequency for 32.768 KHz
5. Initialize timer2 registers for desired count, overflow interrupt generation

INT_DIRECTOR
1. Same as Part 1
1a. IF int from button, update flag, then go to button_svc
2. Check if interrupt from Timer2
   IF NOT Timer2, return to wait loop and enable IRQ
3. ELSE check if Timer2 overflow interrupt
   IF NOT overflow, enable IRQ and return to wait loop
   ELSE go to BUTTON_SVC

BUTTON_SVC
1. Same as Part 1
2. Check flag
   IF ButtonFlag = 1, enable UART2 interrupts
   ELSE disable UART2 interrupts
3. Enable new IRQ and return to wait loop

TALKER_SVC
1. Same as Part 1
2. In SEND, After last character sent, start Timer2 and set for auto reload
3. Enable new IRQs and return to wait loop

8/5 Part 2 Low Level Algorithm (from ch.5 pg 241-247)

Mainline

1. Before CPSR and wait loop,
   Write 0x2 to INTC_SYSCONFIG at 0x48200010 to reset INTC
2. Initialize TIMER2 interrupt (INTC #68)
   Write 0x10 to INTC_MIR_CLEAR2 at 0x482000C8
3. Turn on TIMER2 clock
   Write 0x2 to CM_PER_TIMER2_CLKCTRL at 0x44E00080
4. Set TIMER2 clock frequency for 32.768 KHz
   Write 0x2 to PRCMCLKSEL_TIMER2 at 0x44E00508
5. Initialize TIMER2 registers for desired count and overflow interrupt generation
   Write 0x1 to CFG register at 0x48040010 to reset TIMER2
   Write 0x2 to IRQENABLE_SET at 0x4804002C
   Write 0xFFFB0000 (count value) to TCRR at 0x4804003C to get 10 seconds
   Write 0xFFFB0000 (count value) to TLDR at 0x48040040 to get 10 seconds

INT_DIRECTOR

1. After UART check,
2. Check if interrupt from Timer
   BEQ to TIMER_SVC
   Test 0x10 in INTC_PENDING_IRQ2 at 0x482000D8
   BEQ BUTTON_CHECK (IF NO, check if interrupt from button)
3. IF YES, check if timer overflow (BNE)
   Test 0x2 in TIMER2_IRQSTATUS at 0x48040028
   Turn off overflow IRQ
   Write 0x2 to IRQSTATUS at 0x480400
   IF YES (BNE), go to BUTTON_SVC
   IF NO, enable new interrupt and return to wait loop
4. BUTTON_CHECK:
   IF button pressed to turn off (BUTTON_FLAG=0), BNE RESET_TURN_OFF (end of SEND)
   IF no button, enable new interrupts and return to wait loop

TALKER_SVC

* Add tag TURN_OFF (no more character section) RESET
2. AFTER decrementing counter,
   Start TIMER2 and set for auto reload
   Write 0x3 to TIMER2_TCLR at 0x48040038

8/7 Per RC8660 Tech Ref (pg 22),

English: "Your blood pressure is 120 over 70. Your pulse is 54."

German: "Ihr Blutdruck ist 120 über 70. Ihr Puls ist 54."

Pheneme mode,

"/Ee\'uh !/B/oo\f'trrutski/i\zt! ain'hoon-derts'vahn'tsik! /oo\'buh!z/ee\b'tsich!.

/Ee\'uh! P/oo\tz! /i\zt! feer'oond'Foonf'tsich!.

Sources:

- FluentU, "How to Pronounce German Words with a Surprisingly Simple Method" https://fluentu.com/blog/german/how-to-pronounce-german-words/
- https://en.wikibooks.org/wiki/German/Grammar/Alphabet_and_Pronunciation
- Google Translate
- Page 22 in RC8660 technical Reference

8/11 TIMER3: Switching from TIMER2 may cause project to run properly currently starting TIMER2 causes button to produce no sound, even though timer can't currently affect program (commented out)

- TIMER3 base address on pg 274 of 371 text (register offsets same as TIMER2)
- TIMER clock select (CLKSEL_TIMER3_CLK) offset on pg 1373 of BBB Tech Ref
- INTC_MIR_SET2: Alternate way to disable timer (instead of CLDR, TA said NOT to use CLDR)
  offset is 0xCC on pg 582 BBB Tech Ref (Same for both TIMER2 and TIMER3)

⟹ TIMER3 didn't solve problem, message still sends but doesn't speak

☆ Solved by Simplified branching, copied code to sections that ended it so code didn't jump around as much. ☆

8/12 More info on making RC8660 speak German:

- The apostrophes in the phonetic spelling on previous page are placed inbetween syllables to make the words easier to hear. Without the apostrophes, it was hard to pick out the German word or that it was even German.
- The "/" and "\" are pitch up and pitch down respectively. These were placed to match the rules of the German language as outlined in the listed sources
- The letter "ü" in the word "über" has a sound that is not possible to write in English phonetically. The closest sound is "oo" like in "boot".

Shaun Crippen
ECE 372 Project 1

RS-232C DRIVER ALGORITHMS, HIGH LEVEL

Hook interrupt procedure INT_DIRECTOR in BeagleBone Black (BBB) startup file

MAINLINE
1. Set up stacks for supervisor and interrupt modes
2. Enable GPIO1 module clock
3. Enable UART2 module clock
4. Set up falling edge detection on GPIO1_30 (button)
5. Enable button as interrupt source
6. Initialize interrupt controller (INTC) and unmask button interrupt (INTC #98), UART2 interrupt (INTC #74)
7. Set UART2 baud rate
8. Set UART2 16x divisor
9. Disable UART2 FIFO
10. Change pin MUX mappings on BBB P8/P9 connectors from HDMI framer to UART2 signals
11. Initialize TIMER2 interrupt (INTC #68)
12. Turn on TIMER2 clock
13. Set TIMER2 clock frequency for 32.768 KHz
14. Initialize TIMER2 registers for desired count, overflow interrupt generation
15. Enable interrupt requests by clearing bit 7 in CPSR
16. Wait for interrupt

INT_DIRECTOR
1. Save registers, including link register
2. Check if interrupt from UART2 signal in INTC
3. IF YES, check if interrupt from UART2 Transmit Hold Register (THR) and Modem Status Register (MSR)
    a. IF YES, go to TALKER_SVC
    b. ELSE enable new interrupt and go back to wait loop
4. Check if interrupt from TIMER2
    a. IF NOT TIMER2, return to wait loop and enable new IRQ
5. IF YES, check if TIMER2 overflow interrupt
    a. IF NOT overflow, enable new IRQ and return to wait loop
    b. ELSE go to BUTTON_SVC
6. ELSE IF check if interrupt from button signal in INTC
7. IF YES, check if interrupt from button pin
    a. IF YES, update flag BUTTON_FLAG, then go to BUTTON_SVC
    b. ELSE enable new interrupt and go back to wait loop

BUTTON_SVC
1. Turn off button interrupt
2. Enable new interrupt in INTC
3. Update BUTTON_FLAG
4. Check BUTTON_FLAG (1 = ON, 0 = OFF)
    a. IF BUTTON_FLAG = 1, enable THR and MSR interrupts in UART2_IER
    b. ELSE disable UART2 interrupts
5. Return to wait loop

TALKER_SVC (See page 260 in ECE 371 text for flow diagram)
1. Check if clear to send (CTS#) is asserted
2. Check if THR is empty

a. IF CTS# asserted and THR empty, write character to THR (In SEND procedure, start TIMER2 and set for auto reload)
b. ELSE IF CTS# asserted and THR not empty, return to wait loop
c. ELSE IF CTS# not asserted and THR empty, disable THR interrupt
d. ELSE return to wait loop

Shaun Crippen
ECE 372 Project 1

RS-232C DRIVER ALGORITHMS, LOW LEVEL

Hook interrupt procedure INT_DIRECTOR in BeagleBone Black (BBB) startup file
    Modify startup.s file using “b INT_DIRECTOR” instruction

MAINLINE
1. Set up stacks for supervisor and interrupt modes
    Load stack pointers
    Point to top of stacks (full descending)
2. Enable GPIO1 module clock
    Write 0x2 to CM_PER_GPIO1_CLKCTRL at 0x44E000AC
3. Enable UART2 module clock
    Write 0x2 CM_PER_UART2_CLKCTRL at 0x44E00070
4. Set up falling edge detection on GPIO1_30 (button)
    Load word from GPIO1_FALLINGDETECT at 0x4804C14C
    Set bit by “ORing” word with 0x40000000
    Write result back to GPIO1_FALLINGDETECT at 0x4804C14C
5. Enable button as interrupt source
    Write 0x40000000 to GPIO1_IRQSTATUS_SET_0 at 0x4804C034
6. Initialize interrupt controller (INTC) and unmask button interrupt (INTC #98), UART2 interrupt (INTC #74)
    Write 0x4 to INTC_MIR_CLEAR3 at 0x482000E8 to unmask INTC #98
    Write 0x400 to INTC_MIR_CLEAR2 at 0X482000C8 to unmask INTC #74
7. Set UART2 baud rate
    Change to mode A by write 0x83 to LCR_UART2 at 0x4802400C
    Write 0x00 to DLH_UART2 at 0x48024004
    Write 0x4E to DLL_UART2 at 0x48024000
8. Set UART2 16x divisor
    Write 0x00 to MDR1_UART2 at 0x48024020
    Write 0x03 LCR_UART2 at 0x4802400C to switch back to operational mode
9. Disable UART2 FIFO
    Write 0x6 to FCR_UART2 at 0x48024008
10. Change pin MUX mappings on BBB P8/P9 connectors from HDMI framer to UART2 signals (RMW)
    “OR” 0x26 to UART2_CTSN at 0x44E108C0 for CTS signal
    “OR” 0x6 to UART2_RTSN at 0x44E108C4 for RTS signal
    “OR” 0x21 to UART2_RXD at 0x44E10950 for RXD signal
    “OR” 0x1 to UART2_TXD at 0x44E10954 for TXD signal
11. Initialize TIMER2 interrupt (INTC #68)
    Write 0x010 to INTC_MIR_CLEAR2 at 0x482000C8 to unmask INTC #68
12. Turn on TIMER2 clock
    Write 0x2 to CM_PER_TIMER2_CLKCTRL at 0x44E00080
13. Set TIMER2 clock frequency for 32.768 KHz
    Write 0x2 to CLKSEL_TIMER2_CLK at 0x44E00508
14. Initialize TIMER2 registers for desired count, overflow interrupt generation
    Write 0x1 to TIMER2_CFG at 0x48040010 to reset TIMER2
    Write 0x2 to TIMER2 IRQENABLE_SET at 0x4804002C
    Write 0xFFFB0000 to TIMER2 TLDR at 0x48040040 to reload counter to 10 seconds
    Write 0xFFFB0000 to TIMER2 TCRR at 0x4804003C to start counter at 10 seconds
15. Enable interrupt requests by clearing bit 7 in CPSR (where Rx = register TBD)
    MRS Rx, CPSR
    BIC Rx, #0x80
    MRS CPSR_c, Rx

Shaun Crippen
ECE 372 Project 1

16. Wait for interrupt
    LOOP:
        NOP
        b LOOP

INT_DIRECTOR
1. Push R0-R10, R14 to IRQ stack
2. Check if interrupt from UART2 signal in INTC
    Test 0x400 (bit 10, INT #74) in INTC_PENDING_IRQ2 at 0x482000D8
3. IF YES, check if IT_PENDING bit (bit 0) in UART2_IIR at 0x48024008 (0 in bit = IRQ pending)
    IF bit = 0, go to TALKER_SVC
    ELSE check if IRQ is from TIMER2
4. Check if IRQ from TIMER2
    Test 0x400 (bit 10, INT #68) in INTC_PENDING_IRQ2 at 0x482000D8
5. IF YES, check if TIMER2 overflow interrupt
    Test 0x40000000 (bit 30) in GPIO1_IRQSTATUS_0 at 0x4804C02C
    IF bit 30 set, update BUTTON_FLAG
        Test 0x1 in BUTTON_FLAG
        IF BUTTON_FLAG = 0, change BUTTON_FLAG = 1
        IF BUTTON_FLAG = 1, change BUTTON_FLAG = 0
        Go to BUTTON_SVC
6. ELSE IF check if interrupt from TIMER2
    Test 0x10 (bit 4, INTC #68) in INTC_PENDING_IRQ2 at 0x482000D8
    IF NOT TIMER2, enable new IRQ and return to wait loop
7. ELSE check if TIMER2 overflow interrupt
    Test 0x2 (bit 1) in TIMER2 IRQSTATUS at 0x48040028
    IF NOT overflow, enable new IRQ and return to wait loop
    ELSE go to BUTTON_SVC
8. Check if IRQ from button
    Test 0x4 (bit 2, INT #98) in INTC_PENDING_IRQ3 at 0x482000F8
9. IF YES, check GPIO pin
    Test 0x40000000 (bit 30) in GPIO1_IRQSTATUS_0 at 0x4804C02C
    IF pin, go to BUTTON_SVC
10. Update BUTTON_FLAG
    Test #0x1 in BUTTON_FLAG
        IF BUTTON_FLAG = 0, update to 1 and go to BUTTON_SVC
        IF BUTTON_FLAG = 1, update to 0 and go to TURN_OFF
11. Enable new interrupt and return to wait loop
    Write 0x1 to INTC_CONTROL at 0x48200048
    Pop registers R0-R10, link register ("LDMFD SP!, {R0-R10, LR}" instruction)
    Go back to last instruction in supervisor mode ("SUBS PC, LR, #4" instruction)

BUTTON_SVC
1. Turn off button interrupt
    Write 0x40000000 to GPIO1_IRQSTATUS_0 at 0x4804C02C
2. Enable new interrupt in INTC
    Write 0x1 to INTC_CONTROL at 0x48200048
3. Enable UART2 interrupts
    Write 0xA to UART2_IER at 0x48024004
4. Return to wait loop
    Pop registers R0-R10, LR
    Go back to last instruction in supervisor mode (dead loop)

Shaun Crippen
ECE 372 Project 1

TALKER_SVC

1. Check if clear to send (CTS#) is asserted low
   - Test 0x10 (bit 4) in UART2_MSR at 0x48024018 (bit 4 = 1, CTS# is low)
   - IF bit 4 = 1, check is THR is empty
   - ELSE bit 4 = 0, BEQ NOCTS, check if THR is empty
2. Check if THR is empty
   - Test 0x20 (bit 5) in UART2_LSR at 0x48024014 (bit 5 = 1, THR empty)
   - IF bit 5 = 1, send character (BNE SEND)
   - ELSE return to wait loop and enable new IRQ
3. NOCTS (branch)
   - Test 0x20 (bit 5) in UART2_LSR at 0x48024014
   - IF bit 5 = 1 (THR empty, CTS# not asserted low)
     - Write 0x8 to mask THR IRQ (disable) in UART2_IER at 0x48024004 and return to wait loop
   - ELSE enable new IRQ and return to wait loop
4. SEND (CTS# asserted low, THR empty)
   - Create a message character pointer CHAR_PTR and message character count CHAR_COUNT in .data at end of code (allocate space in memory)
   - Load CHAR_PTR that points at next character to send
   - Load CHAR_COUNT to get number of characters left to send
   - Load current character to send from CHAR_PTR, and post increment to next character
   - Point to next character in CHAR_PTR
   - Decrement CHAR_COUNT and store in memory (.data)
   - Store current character to send in UART2_THR at 0x48024000
   - IF no more characters,
     - Load message into CHAR_PTR (point at first character again)
     - Load character count into CHAR_COUNT (message length)
     - Disable UART2 interrupts
       - Write 0x0 to UART2_IER at 0x48024004
     - Start TIMER2 and set auto reload
       - Write 0x03 to TCLR at 0x48040038
     - Enable new IRQ and return to wait loop
   - ELSE enable new IRQ and return to wait loop

TURN_OFF

1. Mask TIMER2 interrupt
   - Write 0x10 to INTC_MIR_SET2 at 0x482000CC
2. Reset character pointer and character counter for message, and disable UART2 interrupts
   - See “IF no more characters” in SEND above