Overview of Off-Processor Serial Communication On The Long Range Autonomous Underwater Vehicle (LRAUV)

01/12/2011

System Overview:

The LRAUV is controlled by an off-the-shelf, single board computer populated with an NXP LPC3250 ARM-9 processor. This computer interfaces with the custom LRAUV motherboard and various sensors and controllers on the LRAUV. The processor itself contains 7 UARTs (3 high speed) and 2 SPI buses. One of the 7 onboard UARTs is configured on the motherboard to operate as RS-485. All others can be configured to operate RS-232 or TTL. To make use of these UARTs and SPI buses, a Linux BSP provided by the manufacturer is run on the processor. While the kernel remains the same (2.6.27.8), some of the drivers have been customized to meet the requirements of the LRAUV. Additionally, a custom program is run on top of the Linux OS that actually controls the vehicle, logs data, and monitors health.

Since the LRAUV carries more than 7 different serial devices, off-chip UARTs are used to provide a communication interface between the processor and various serial devices. The UARTs are MAX3100 devices that communicate via one of the processor's SPI busses. Up to 22 of these UARTs can be installed for a total of 29 individual UARTs in the system. Other devices that are installed on the same SPI bus include up to 24 PIC controllers, and up to 2 SPI A/D chips.

Off Processor Communication:

The MAX3100 UART uses SPI communication on the host side and supports RS-232 and RS-485 communication from 300 to 230k baud on the client side. It consists of an 8 byte deep FIFO and generates an interrupt on receipt of a word. If the FIFO is overrun, it simply clears the data.

Since there are not enough GPIO lines available to individually control 48 different SPI devices, the chip select (CS) lines are muxed into 4 address lines. Two other address lines select one of three backplane cards on which between 6 and 8 of the MAX3100s can be installed (as well as 8 PIC controllers). One additional address line acts as a "gate" or "address valid" signal so errant CS signals are not asserted while the address is being activated in software. Addressing in the current configuration is as follows where "board select" is the value of the 6 address lines:

Board Select 0x	**Device / Slot #**	**Backplane #**
0	SPI A/D 0	0
1	PIC 0	0
2	SPI A/D 1	0
3	PIC 1	0
4	MAX3100 2	0
5	PIC 2	0
6	MAX3100 3	0
7	PIC 3	0
…	…	…
1A	MAX3100 13	1
1B	PIC 13	1
…	…	…

When data arrives in the MAX3100 buffer, an interrupt signal is asserted on the device. This signal from every MAX3100 is OR'ed together into one single general purpose input on the LPC3250 processor. Upon receipt of this interrupt, the kernel runs an ISR that queries every MAX3100 UART that is currently open. Upon clearing the FIFO of the interrupting MAX3100, the interrupt is de-asserted. Figure 1 is a graphical example of how 6 open MAX3100s get serviced by the Linux driver.

Fig: 1

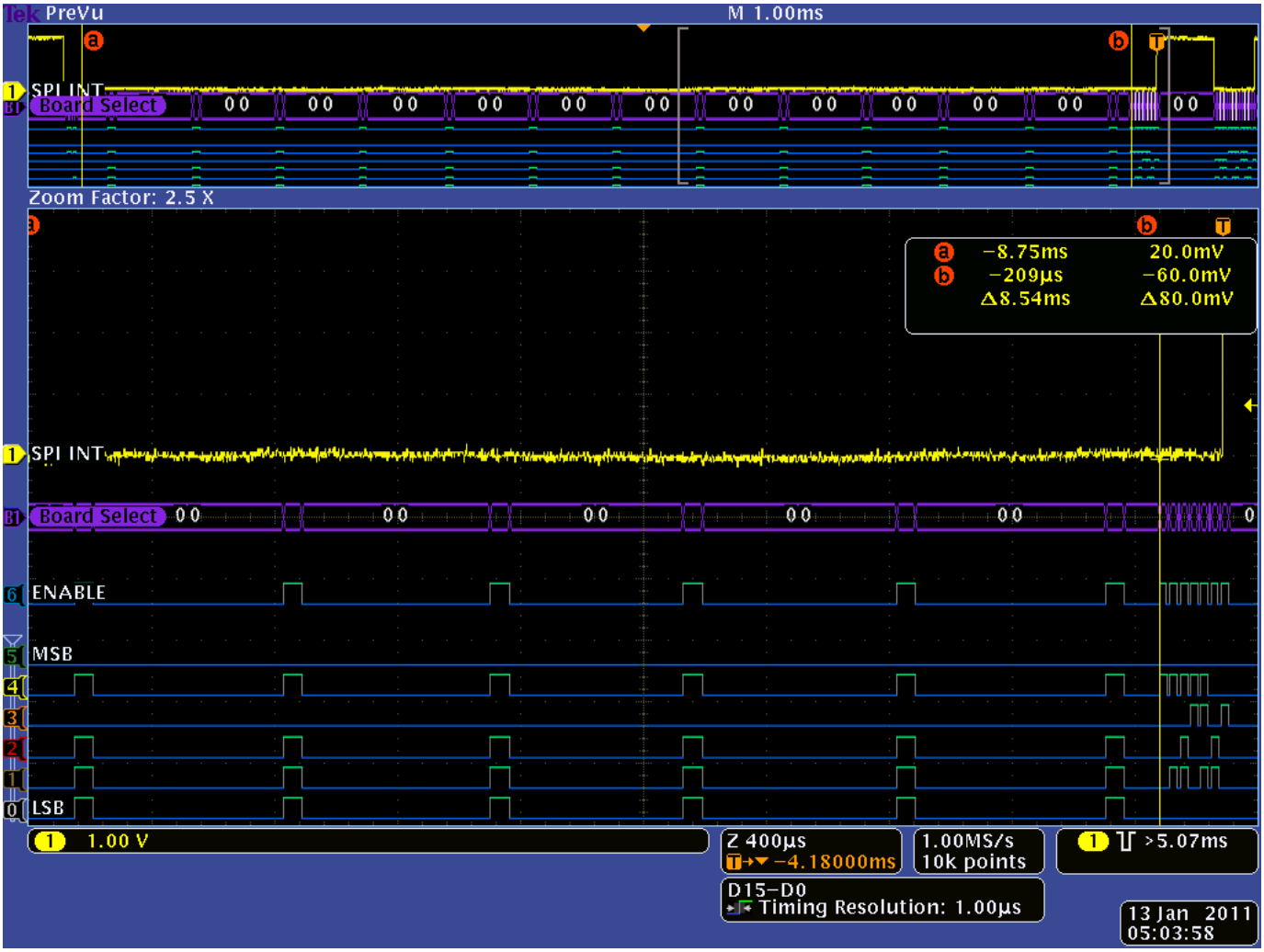
Channel 1 “SPI INT” is the single MAX3100 interrupt input into the processor. The purple “Board Select” bus is the address of the aforementioned 6 address lines (labeled MSB-LSB). The enable line is shown on digital input 6. Note that the entire sequence takes approximately 180 microseconds. Based on the board select addresses, one can see the driver querying MAX 13 (0x1A), MAX 6 (0x0C), MAX 7 (0x0E), MAX 8 (0x10), MAX 9 (0x12), and MAX 12 (0x18). Upon retrieving data from the MAX3100 in slot 12 (i.e. slot 5 on backplane #1), the active low interrupt was de-asserted.

Potential Data Loss:

Spending 180 microseconds servicing an interrupt isn’t an issue with the data rates on the LRAUV and an 8 byte FIFO on the MAX3100. For example, an instrument running at 9600 baud might take around 8 milliseconds to fill the FIFO. Data loss could occur though if the interrupt isn’t serviced fast enough. Currently problems have been observed where the interrupt is

asserted for nearly 9 milliseconds (figure 2). This is more than enough time for devices to overrun the FIFO.

Fig: 2



In figure 2, the PIC at location 11 (bus address 0x17) was unresponsive to requests from the driver. The problem is that throughout the greater than 400 millisecond pauses in between communications with the PIC, no MAX3100s are being queried despite the interrupt being asserted for more than 8.5 milliseconds. It isn't until the right-hand side of the image at cursor B that the MAX3100s are finally handled. By this time, data loss could easily have occurred.

You can see a similar event happening with PIC 13 (0x1B) in figure 3 through 5. In this case, there was nothing wrong with the PIC and everything responded appropriately as far as it can be known. Figure 3 shows the beginning of the interrupt where MAX3100s were being serviced until cursor A. Then figure 4 shows the offending PIC transaction with figure 5 showing the entire 4.73 millisecond interrupt assertion ending with the MAX3100 in slot 4 (0x8) being serviced.

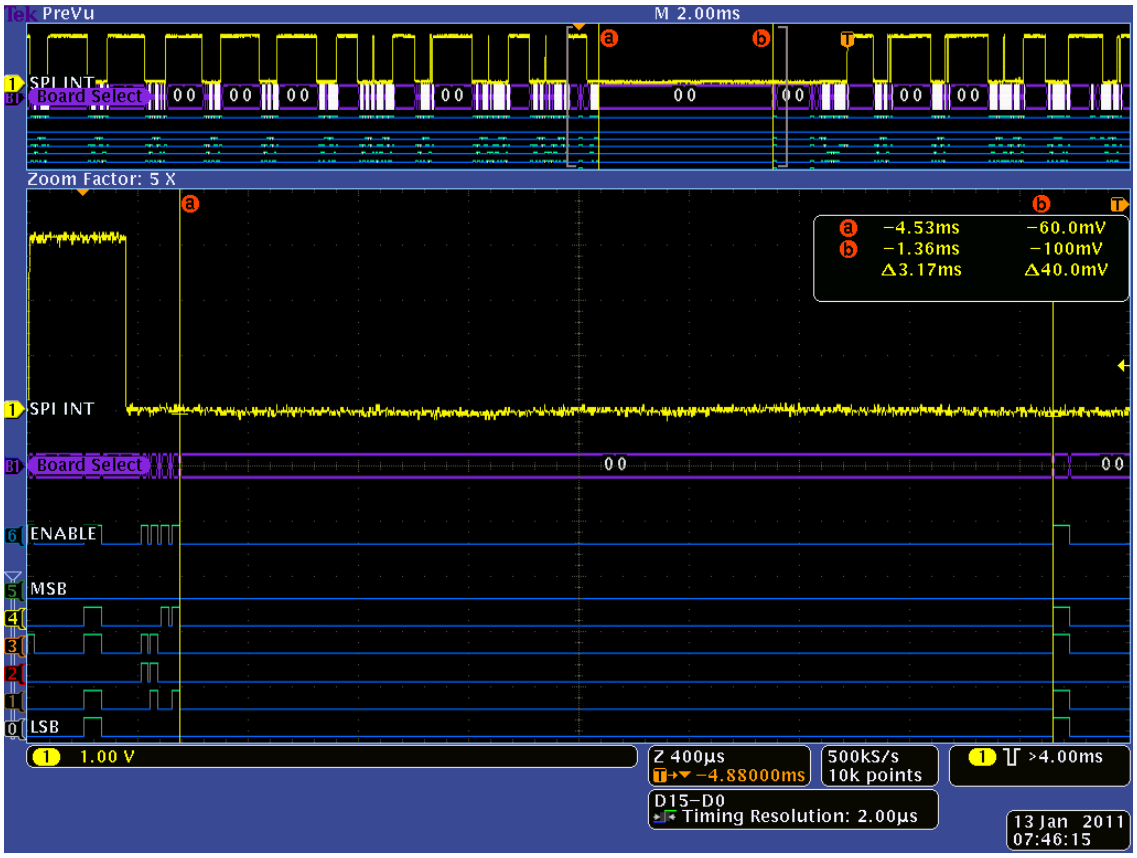
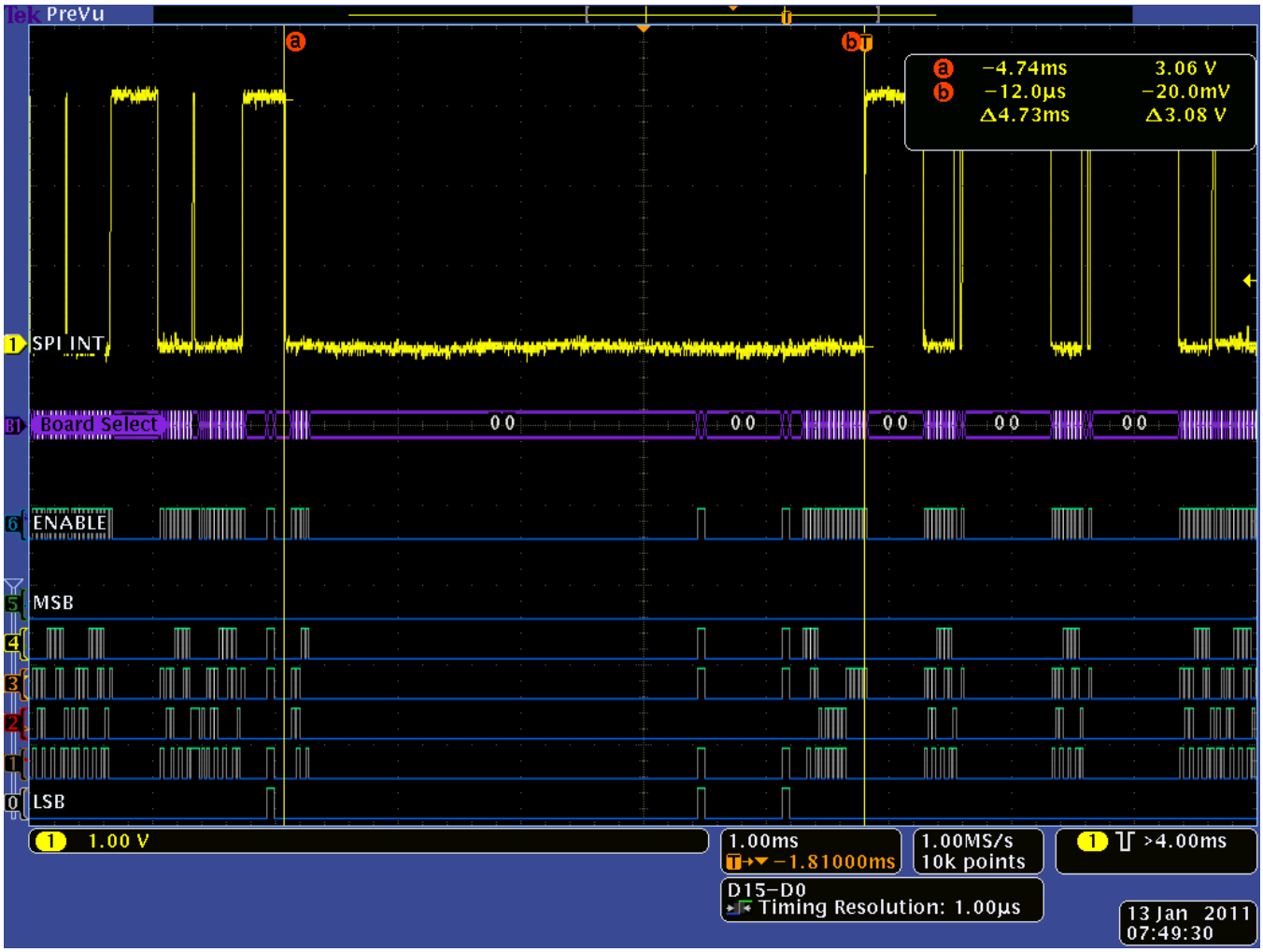

PreVu
M 2.00ms
SPI INT
Board Select
Zoom Factor: 5 X
a -4.53ms -60.0mV
b -1.36ms -100mV
Δ3.17ms Δ40.0mV
ENABLE
MSB
LSB
1.00 V
Z 400µs
-4.88000ms
500kS/s
10k points
>4.00ms
D15–D0
Timing Resolution: 2.00µs
13 Jan 2011
07:46:15

Fig: 3, Fig: 4


PreVu
M 1.00ms
SPI INT
Board Select
Zoom Factor: 10 X
a -4.52ms 20.0mV
b -1.36ms -100mV
Δ3.16ms Δ120mV
ENABLE
MSB
LSB
1.00 V
Z 100µs
-3.23000ms
1.00MS/s
10k points
>4.00ms
D15–D0
Timing Resolution: 1.00µs
13 Jan 2011
07:48:38

Fig: 5



Future Work:

Clearly some driver work must be done to ensure that time isn't wasted while the SPI interrupt is asserted. The SPI driver for the PIC is suspicious since all observed latencies seem to occur prior to a transaction with that device. In addition it would be useful to know about data losses at a higher level. To enable this, a count of errors can be added to any device interface that checks data integrity (e.g. with a checksum). Then error rates could be observed for devices of different baud and sampling rates.