Author: Abhishek Dhital

Course: CSE 5356 (System on Chip Design)

Organization: University of Texas at Arlington

Serial IP Core and Linux Device Drivers

1) Introduction

The target hardware platform for the implemented serial IP module is Xilinx XUP Blackboard. The IP module is programmed on FPGA fabric and then configured from the processor subsystem over AXI4 bus using a virtually mapped memory interface. There are four registers used to control and configure the IP module. It has a 4 word-long, each word with 4 bytes, register space for that purpose. In short, the module works as a standard UART receiver and transmitter with both having a 16-byte long and 9-bit wide FIFO buffer, with configurable but common baud rate, data length, parity bit and one/two stop bits options. It generates an interrupt when the receiver FIFO is not empty, which is fed into the processor subsystem for interrupt handling. On the processor subsystem that runs Linux operating system, a kernel module is implemented, that creates a sysfs interface for configuring the module, the interrupt handler which captures any incoming data to a software FIFO and a tty device driver which is used to send/receive characters using the IP module.

2) Register space

The serial module can be accessed via AXI4-lite bus at an offset of 0x20000 from the base address of the bus – 0x43C00000. The register space has four 32-bit registers at certain offsets, that have various fields used for setting the serial communication channel.

a. DATA (R/W) (Offset 0)

This is a r/w register at an offset of 0. Out of the 32 bits, only bits 8:0 are used. A read on this register reads data from the RX FIFO, while a write to this register sends data to the TX FIFO for transmission. The module supports variable length of data, which can be adjusted with the CONTROL register.

b. STATUS (R/W1C) (Offset 4)

This register stores various flags that indicate the state of the serial module. The RXFO, TXFO, FE and PE fields can be cleared by writing a 1 to them. The other fields include TX and RX watermarks, TXFF, RXFF, TXFE and RXFE.

c. CONTROL (R/W) (Offset 8)

The bits 1:0 of this register are used to configure the data length for TX and RX, with data lengths from 5-bit to 9-bit supported. Parity can be set to off, even and odd using the bits 3:2. Bit 4 enables and disables the transmitter, receiver and brd generator. Bit 5 is used to send the output of baud rate generator to pin 0 of GPIO on PMODA. Bits 6 and 7 are used to enable and disable interrupt for RX and TX respectively.

d. BRD (R/W) (Offset 12)

The last register at offset 12 is the BRD register. Bits 7:0 are used to write the fractional portion of the baud rate divider and bits 31:8 are used to write the integer portion of the divider.

3) Serial IP Components

a. Baud Rate divider

This module uses the PLL (100MHz) as a reference, to output a data recovery clock (f_{clk}) with a clock rate of (16 * baud rate) using the integer and fractional portions of the baud rate divider stored in the BRD register. This module is enabled and disabled using the bit field 4 in the CONTROL register. An example of this module is shown in the image below. When the baud rate is set at 9600, the output of this module is a clock signal with frequency close to 16 * 9600 = 153.60 kHz.

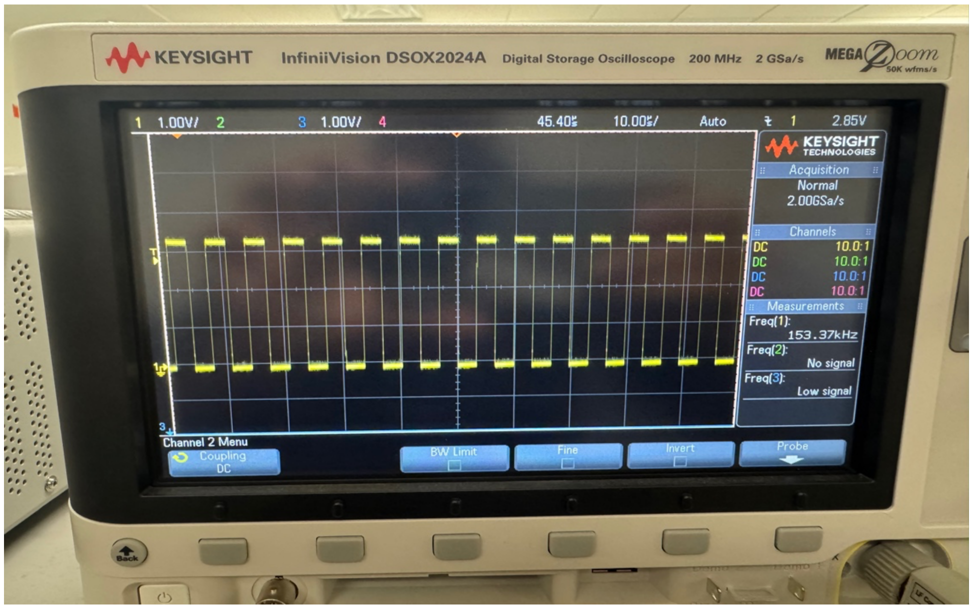


Figure 1 Waveform of BRD generator output on GPIO 0

b. Transmit FIFO and Transmitter

The transmit FIFO is an instantiation of a FIFO16X9 module, which is 9bit wide and 16 level deep FIFO used to store the data to transmit. When the FIFO is full, TXFF bit is set in the STATUS register which indicates the FIFO is full, and data must be transmitted out before any more data can be stored. If additional data is attempted to transmit, TXFO bit is set in the STATUS register to indicate there was an overflow. When the FIFO is empty, the TXFE bit is set in the STATUS register.

The transmitter module, when the serial IP is enabled and TXFE bit is not set, starts sending a start bit and data bits from LSB to MSB, the parity bit if required and one or two stop bits depending on the configuration via TX pin on PMOD A. The rate of transmission is the baud rate which is (f_{clk} / 16). The waveform below shows a transmission of the character 'B' (ASCII – 66), when the data length is 8 bits, parity is set to even and the baud rate is 115200.

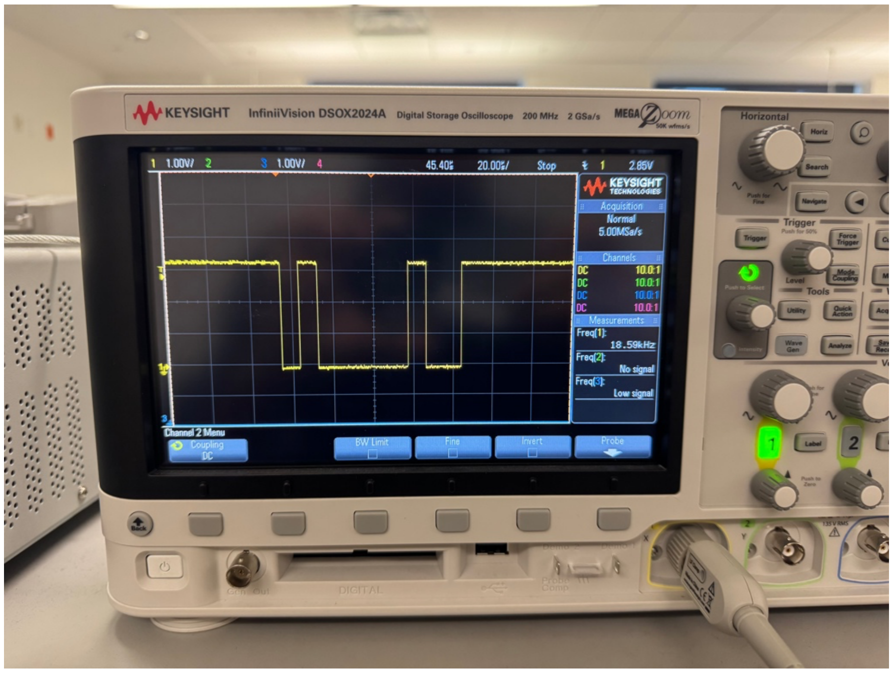


Figure 2 Waveform of data transmitted by transmitter module via GPIO 1

c. Receiver FIFO and Receiver

The receiver FIFO is an instantiation of a FIFO16X9 module, which is 9bit wide and 16 level deep FIFO used to store the data received on the RX pin on PMODA. When the FIFO is full, RXFF bit is set in the STATUS register which

indicates the FIFO is full, and data must be read out before any more data can be stored. If additional data is sent on the RX pin when the FIFO is full, RXFO bit is set in the STATUS register to indicate there was an overflow. When the FIFO is empty, the RXFE bit is set in the STATUS register.
The receiver module, when serial IP is enabled, operates on f_{clk} and implements an FSM, sampling for IDLE, START, DATA, PARITY and STOP bits at every 7th, 8th and 9th cycle of a 4-bit phase counter which increments every f_{clk}.

4) Interrupt Generation

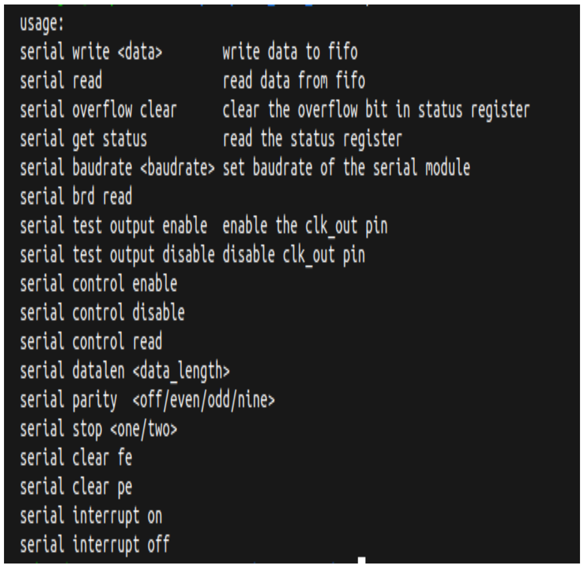
An interrupt is generated, when interrupt is enabled in the CONTROL register and either the RX FIFO is not empty, or TX FIFO is empty. This interrupt signal is fed into the processor subsystem for interrupt handling.

5) Reset

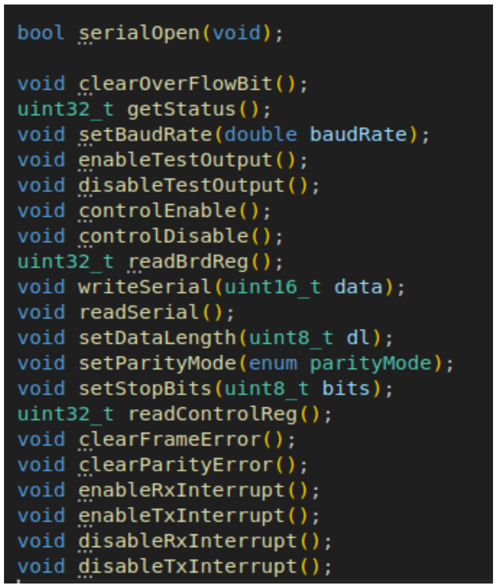
The baud rate generator, FIFO, transmitter and receiver modules are designed to accept an active low reset signal which will reset the hardware to a known state when the system is reset/restarted.

6) Virtual Memory (/dev/mem) Control

/dev/mem is used to memory map and write to all the registers for specific configurations. A Serial IP library is written in C which provides various operations on the registers using the same memory mapping. A test application is also developed that make calls to these library functions. They can be seen in the snippets below:



```
usage:
serial write <data>       write data to fifo
serial read               read data from fifo
serial overflow clear     clear the overflow bit in status register
serial get status         read the status register
serial baudrate <baudrate> set baudrate of the serial module
serial brd read
serial test output enable  enable the clk_out pin
serial test output disable disable clk_out pin
serial control enable
serial control disable
serial control read
serial datalen <data_length>
serial parity  <off/even/odd/nine>
serial stop <one/two>
serial clear fe
serial clear pe
serial interrupt on
serial interrupt off
```



```c
bool serialOpen(void);

void clearOverFlowBit();
uint32_t getStatus();
void setBaudRate(double baudRate);
void enableTestOutput();
void disableTestOutput();
void controlEnable();
void controlDisable();
uint32_t readBrdReg();
void writeSerial(uint16_t data);
void readSerial();
void setDataLength(uint8_t dl);
void setParityMode(enum parityMode);
void setStopBits(uint8_t bits);
uint32_t readControlReg();
void clearFrameError();
void clearParityError();
void enableRxInterrupt();
void enableTxInterrupt();
void disableRxInterrupt();
void disableTxInterrupt();
```

Figure 3 Usage of test application and list of functions available in the serial_ip library

7) Virtual File System (sysfs) Control

A kernel module is written to leverage the sysfs interface which can be used to perform same tasks as the IP library. When the module is installed, /sys/serial will be created, with baudRate, parity, stop, tx_data and rx_data as the available files. Read/writes from/to these files will perform the operations suggested by their names. The tx_data file is a write only file with permissions (-w--w--w-), meaning you can only write to this file for data transmission and there is nothing to be read. Similarly, the rx_data file is a read only file (r--r--r--) which can be used to read bytes received. Other files are readable and writable.

8) Interrupt Handler

When an interrupt is detected, the handler (ISR) reads from the DATA register, stores the read data to a much larger software FIFO maintained by the kernel module, which is 1024 bytes in length, and this data can be read by reading the /sys/serial/rx_data file. The TX interrupt is not handled for this project. This ensures that more than 16 bytes of data can be received without causing an overflow on the RX FIFO.

9) TTY device (/dev/ttySOC0)

The same kernel module also creates a tty driver. This driver is responsible for reading and writing data from and to this tty device. When an interrupt is generated, the service routine (ISR) also sends this byte to the tty core for this tty device. This data can be read by opening the tty device using any serial communication utility.
'Microcom' was used for testing:

```
'sudo microcom -p /dev/ttySOC0'
```

A write to this tty device will result in a write to the DATA register which is then transmitted by the transmitter module via TX pin on PMODA. These bytes were sent and received back and forth on another serial interface with an FTDI (USB to serial) chip connected to the RX and TX pins on PMODA.

The following sources were used to research and learn about tty device drivers and their implementation:

1) Docs.kernel.org/driver-api/tty/tty_driver.html
2) Linux Device Drivers: Where the Kernel Meets the Hardware (Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman)