University of Technology, Sydney
Faculty of Engineering and Information Technology

Subject: **48434 Embedded Software**

Assessment Number: **1**

Assessment Title: **Lab 1 – Tower Serial Communications**

Tutorial Group:

Students Name(s) and Number(s)

Student Number	Family Name	First Name

Declaration of Originality:

The work contained in this assignment, other than that specifically attributed to another source, is that of the author(s). It is recognised that, should this declaration be found to be false, disciplinary action could be taken and the assignments of all students involved will be given zero marks. In the statement below, I have indicated the extent to which I have collaborated with other students, whom I have named.

Statement of Collaboration:

Signature(s)

key

Assessment Submission Receipt

Assessment Title:	**Lab 1 – Tower Serial Communications**	**Mark**
Student Name(s):		
Date Submitted:		
Tutor Signature:		Office use only ☺

Assessment Criteria

Your lab will be assessed according to the following criteria:

Item	Detail	Evaluation	Mark
Opening comments	File headers are correct.	**G A P**	**/0.5**
Function descriptions	Function descriptions are appropriate and correct.	**G A P**	**/0.5**
Naming conventions	Names conform to the Software Style Guide.	**G A P**	**/0.5**
Code structure	Code structure conforms to the Software Style Guide.	**G A P**	**/0.5**
UART functions	All control register bits are set correctly. Baud rate set correctly. Correct use of FIFOs. Polling is correct.	**G A P**	**/1**
Packet functions	Packet functionality is correct. Errors are handled correctly.	**G A P**	**/1**
FIFO implementation	Functionally correct, easy to modify (read) and efficient.	**G A P**	**/2**
Protocol implementation	Protocol response is correct, including ACK/NAK.	**G A P**	**/2**
		TOTAL	**/8**

Evaluation

When we evaluate an assessment item, we will use the following criteria:

- **G =** All relevant material is presented in a logical manner showing clear understanding, and sound reasoning. For software - evidence of correct coding style, efficient implementation and / or novel (and correct) code.
- **A =** Most relevant material is presented with acceptable organisation and understanding. For software – some code may be prone to errors under certain operating conditions (e.g. input parameters) or usage, style may have inconsistent sections, occasional inefficient or incorrect code.
- **P =** Little relevant material is presented and/or code displays poor organisation or understanding of the underlying concepts.

Oral Defence

During the demonstration session you will be asked a number of questions based on material which you have learnt in the subject and then used to implement the assignment. You are expected to know exactly how your implementation works and be able to justify the design choices which you have made. If you fail to answer the questions with appropriate substance then you will be awarded **zero** for that component.

Lab 1 – Tower Serial Communications

Port access. UART initialization. Polling. Circular buffer. Packet decoding.

Introduction

The Universal Asynchronous Receiver / Transmitter (UART) is a simple yet extremely important peripheral of the K70 microcontroller. On the Tower board it is connected to a serial-to-USB chip that enables the board to communicate to a PC running Windows 7. The Tower board is to implement the serial communication protocol as outlined in the separate document entitled "Tower Serial Communication Protocol".

Objectives

1. To set up a UART on a microcontroller.
2. To implement a circular buffer in the C language.
3. To decode and respond to packets according to a defined protocol.

Equipment

- 1 TWR-K70F120M-KIT
- 1 USB cable
- Freescale Kinetis Design Studio

Safety

This is a Category A laboratory experiment. Please adhere to the Category A safety guidelines (issued separately).

Cat. A lab

L1.2

Software Overview

The serial communication between the Tower board and the PC uses 5-byte packets. The task of the Tower software is to receive and send packets of information asynchronously.

This is a classic producer-consumer problem. One solution to the asynchronous nature of the communication is to implement a first-in first-out (FIFO) buffer between the producer and consumer. This is shown diagrammatically below:

Data flow graph showing two FIFOs that buffer data between producers and consumers

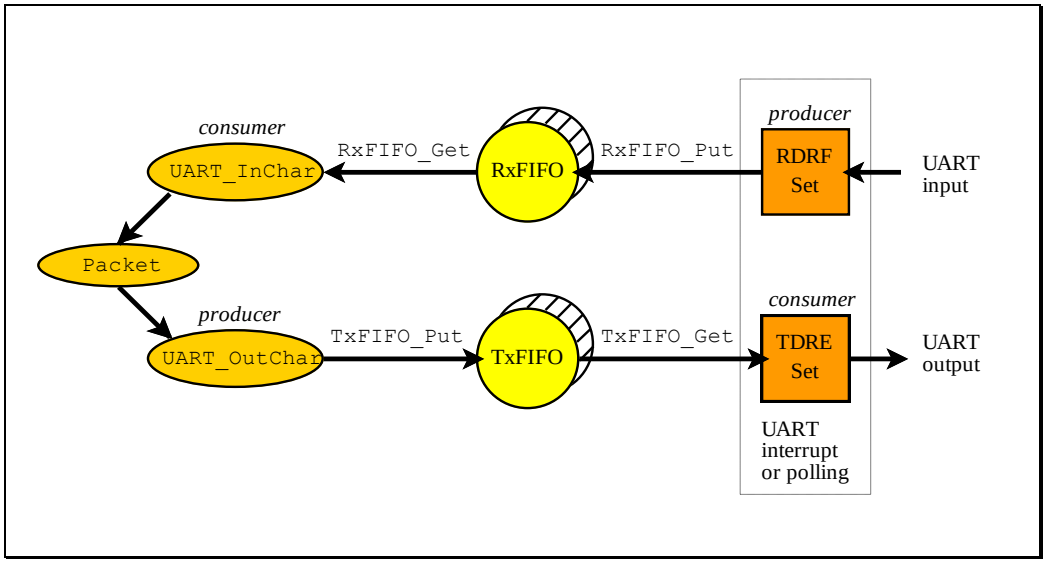


Figure L1.1

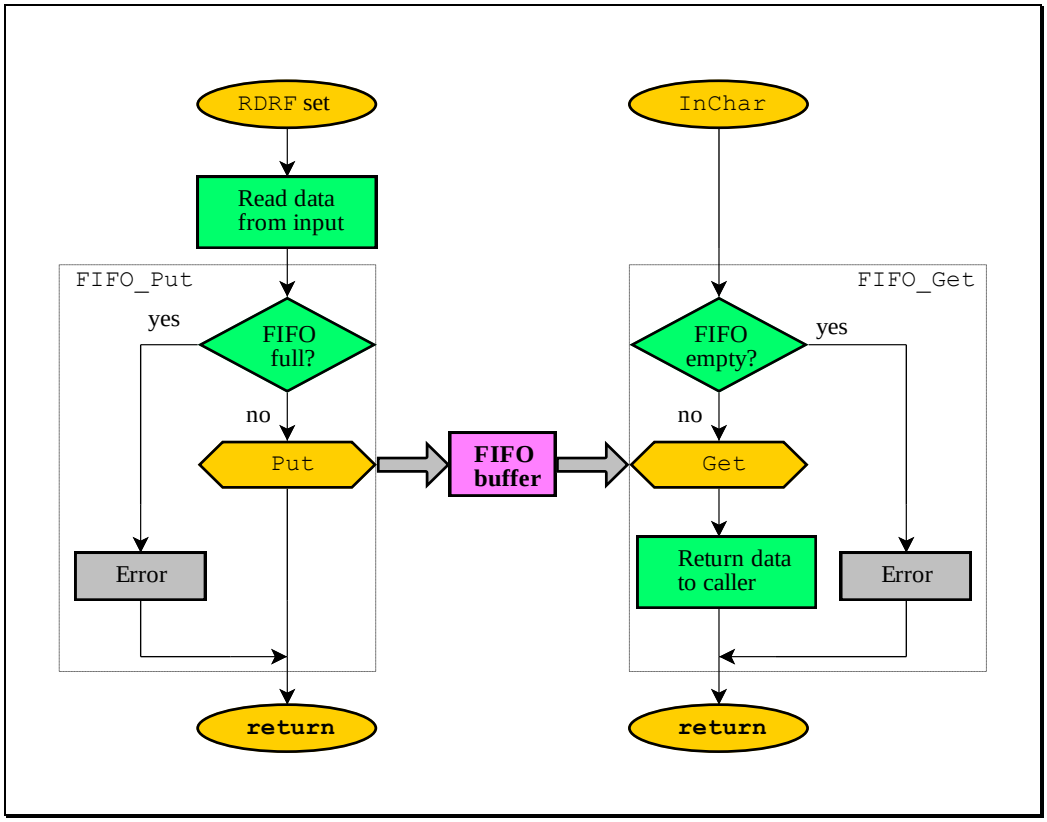
A simple (but inefficient) way of handling the asynchronous nature of the UART communication is to check the status of the UART hardware repeatedly by calling a function in the main loop of the program that *polls* the status of the UART, and calls `RxFIFO_Put` or `TxFIFO_Get` as required.

Receiving Data

When the packet module wishes to receive input, it calls `UART_InChar`, which will attempt to get data from the `RxFIFO` (it will fail if the FIFO buffer is empty). How does data get in the `RxFIFO`?

The incoming serial data will set the Receive Data Register Full (`RDRF`) flag in the UART Status Register 1 (`UART_S1`) register, indicating that the receiver hardware has just received a byte of data. In the main loop, a poll of the `RDRF` flag is performed. If it is set, then the program tries to accept the data and put it in the `RxFIFO`. The `RxFIFO` buffers data between the input hardware and the main program that processes the data. If the `RxFIFO` becomes full, then data will be lost. This is illustrated below:

In this lab, the receive interrupt service routine is replaced by a polling operation



A FIFO queue can be used to pass data between an input device and a main thread

Figure L1.2

FIFO full errors will always occur if the average input rate (number of bytes arriving per second from the input hardware) exceeds the average processing rate (number of bytes processed per second by the main program). In this situation, either the output rate must be increased (by using a faster computer

or by writing a better software processing algorithm), or the input rate must be decreased (by slowing down the arrival rate of data). The second way the `RxFIFO` could become full is if there is a temporary increase in the arrival rate or a temporary decrease in the processing rate. For this situation, the full errors could be eliminated by increasing the size of the `RxFIFO`.

It is inefficient, but not catastrophic, for the main program to wait on an empty `RxFIFO` in some cases. Efficiency can be improved for the buffered input problem by performing other tasks while waiting for data.

Sending Data

When the packet module wishes to output, it calls `UART_OutChar`, which will put the data in the `TxFIFO` and arm the output device.

The setting of the Transmit Data Register Empty (`TDRE`) flag by the UART hardware signals that the output shift register is idle and ready to output more data. In the main loop, a poll of the `TDRE` flag is performed. If it is set, then the program tries to retrieve the data in the `TxFIFO`., and send it out the serial port. If the `TxFIFO` becomes empty, then no data will be sent out the serial port. This is illustrated below:

In this lab, the transmit interrupt service routine is replaced by a polling operation

A FIFO queue can be used to pass data between a main thread and an output device

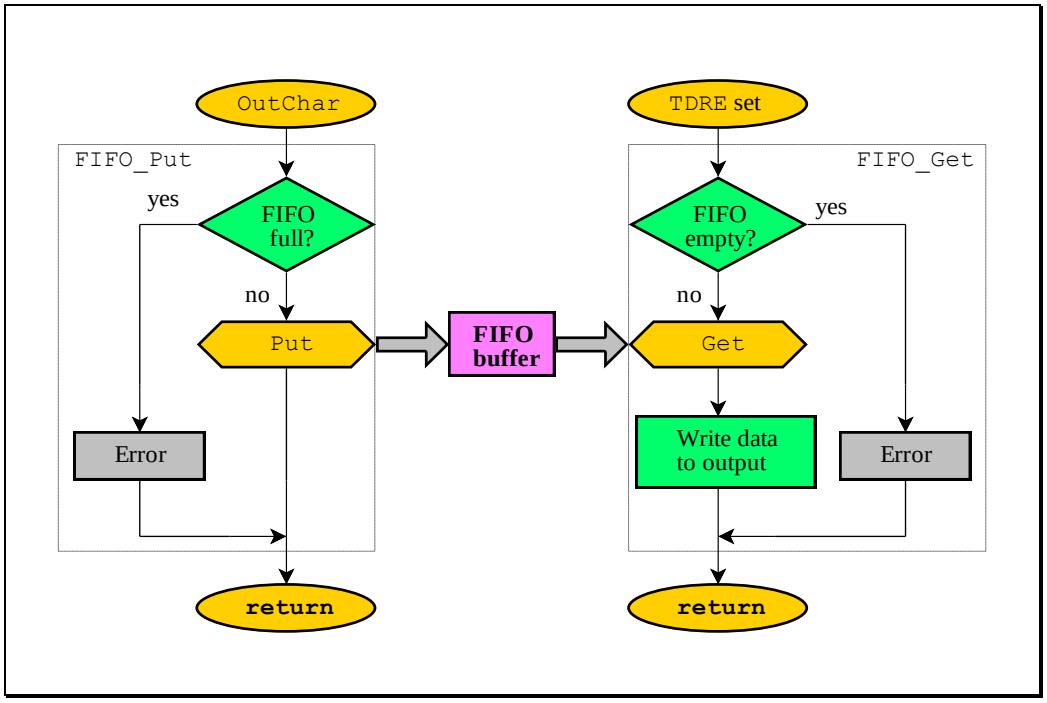


Figure L1.3

It is inefficient, but not catastrophic, for the main program to wait on a full `TxFIFO`. Efficiency can be improved for the buffered output problem by increasing the `TxFIFO` size.

Software Modules

Various modules should be written to support the serial communication protocol.

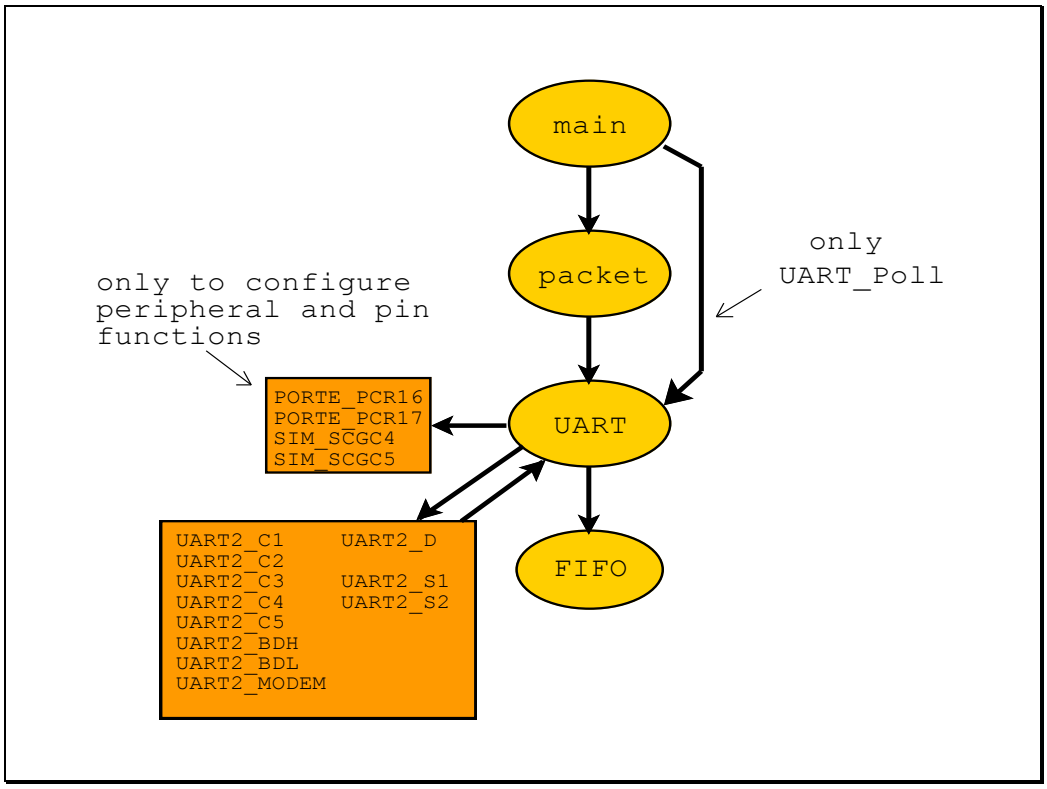


Figure L1.4

L1.6

Software Requirements

1. The serial-to-USB bridge uses UART2. The baud rate is to be 38400 baud.
2. The CPU bus clock frequency can be found in "CPU.h". The CPU bus clock is supplied to UART2 as its "module clock" (see Table 5-2 in K70P256M150SF3RM.pdf). The UART2 module clock is needed for baud rate divisor calculations.
3. The software must be able to handle at least 40 packets in the receive buffer and at least 40 packets in the transmit buffer (each packet is 5 bytes).
4. A frequent polling operation in the main loop should be used to check the status of the serial port.
5. The commands of the Tower serial protocol to be implemented (with packet acknowledgement) are:

Tower to PC	PC to Tower
0x04 Tower startup	0x04 Special – Get startup values
0x09 Special – Tower version	0x09 Special – Get version
0x0B Tower number	0x0B Tower number (get & set)

6. In response to reception of a "0x04 Special – Get startup values" packet from the PC, the Tower should transmit three packets:
    - a "0x04 Tower startup" packet
    - a "0x09 Special – Tower version" packet
    - a "0x0B Tower number" packet
7. Upon power up, the Tower should send the same 3 packets as above.
8. Use the last 4 digits of your student number to initialise the Tower number. Note that it may be changed via the "0x0B Tower number" packet at a later time.
9. The "0x09 Special – Tower Version" should be V1.0.
10. TortoiseSVN must be used for version control.

Marking

The software should be ready for marking on the date specified in the Timetable in the Learning Guide.

Software marking will be carried out in the laboratory, in the format of an oral exam.

Marking criteria are on the front page. Also refer to the document "Software Style Guide".