ELEC241-P1-2020-21 Group Project

Prepared by: Nicholas Outram

1 Revision History

Revision	Changes	Date
0.9	Initial version presented to students as draft	16/04/2021
0.91	Reviewed by Chris Tipney.	17/04/2021
0.92	Minor corrections	19/04/2021

ASSESSED LEARNING OUTCOMES: (additional guidance below)
At the end of the module the learner will be expected to be able to:

1. Design, critically evaluate and implement scalable, reliable and testable embedded systems that interface to real-world signals in real-time.
2. Capture, process and generate real world signals using appropriate techniques in hardware and firmware.
3. Apply appropriate strategies and techniques for verification, validation and documentation of both hardware and firmware elements of an embedded system.
4. Demonstrate and reflect on current best practice in both individual and group management of firmware and hardware development.

This coursework covers all 4 learning outcomes.

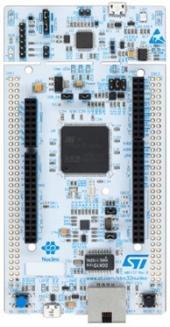
SUMMARY

Introduction

This module has two assessed elements: An individual coursework (C1) and a group practice element (P1). Group management is a requirement of this module. The rationale for this is to teach some basic management practice in a relevant context, and to prepare students for their final year project. Time is short, and you will be working under pressure. Effective team work is essential.

Goals

Greater opportunity for students to work remotely; upgrade repertoire / capability of final stage ELEC and ROBOTICS projects; further enhance student employability; students to build the confidence and capability to correctly use modern technology in their final-year projects, ensure all students are familiar with the following:



(1) Working with real time signals.

(2) The use of modern digital interfaces.

(3) Managing real-time problems in software and hardware.

(4) Programming and interfacing to a Field Programmable Gate Array (FPGA).

Overview– Real Time Interfacing

The theme of this coursework is real-time interfacing. Microcontroller Units (**MCU**) are programmable, low-cost components found in many modern devices. And are often tasked to perform multiple roles. They are designed to interface to other peripheral devices in real-time such that data is never lost and processed in a timely fashion. To make software easier to write, it is helpful if the peripheral devices can be designed to provide some "timing slack". As a MCU can only perform one task at a time, it is helpful if peripheral devices do not require the MCU to block and wait.

Consider the analogy of a team of journalists working to meet a deadline in a busy office.

The lead journalist (person A) is writing an article that requires some additional information. One article is stored in a basement vault and another is in an electronic database. Retrieving both pieces of information take a significant amount of time to perform, so the task is delegated to another person B.

Person A provides person B with a **list** of tasks, with the following benefits to the overall productivity of the business:

- *Person A can continue to work – **they do not need to stop** and wait for person B to complete the task.*
- *Person B can go and retrieve each piece of information and email them in turn to person A.*
  - *Both pieces of information are now held (buffered) in the inbox of Person A.*
- *Person A can pick up the information from person B **when they are ready to process it** (as long as their inbox does not get full)*

We say that person's A and B are working **concurrently** and are also synchronized. Furthermore, when person B reports back via email, we say the communication is asynchronous – this helps person A if they are busy doing something more important or pressing. We say that person B is providing some timing slack for A.

In the electronic world, Person A could be a MCU and person B a peripheral device. The MCU may **synchronously** instruct a device to perform a task, then proceed to perform others tasks itself. The peripheral will then perform some parallel operation on behalf of the MCU. When

done, the data is written to a **buffer,** ready for the MCU to collect when convenient. This gives the MCU some **timing slack**. A key point is that the data is not lost – it is held in a buffer.

Similarly, the MCU may wish to send a peripheral device several instructions or data values. It needs this to be a fast transaction, so the peripheral may also buffer incoming data (remember how Person B was given a list of tasks?). This allows the MCU to provide information very rapidly.

Key to the above is **buffering**, typically using a first-in-first-out (**FIFO**) model.

> Think how a restaurant processes table orders. They are taken to the kitchen and pegged up in a line. Each order is processed in sequence. Waiters add orders to the queue, and chefs pull tasks from the queue. This is a real world first-in-first-out buffer (FIFO)!

The good news is that a FIFO device is so common, that Intel/Altera provide one for us (SCFIFO is a good one!). Along with components such as registers and multiplexers, this is one of our **data-path** components.

In this coursework, the MCU is interfaced to an FPGA as depicted in Figure 1. Note how most peripheral devices are connected to the FPGA. Each peripheral can therefore be managed concurrently (using parallel hardware).

The MCU is also connected to the module support board, so also has access to some peripherals (such as LEDs and a buzzer).

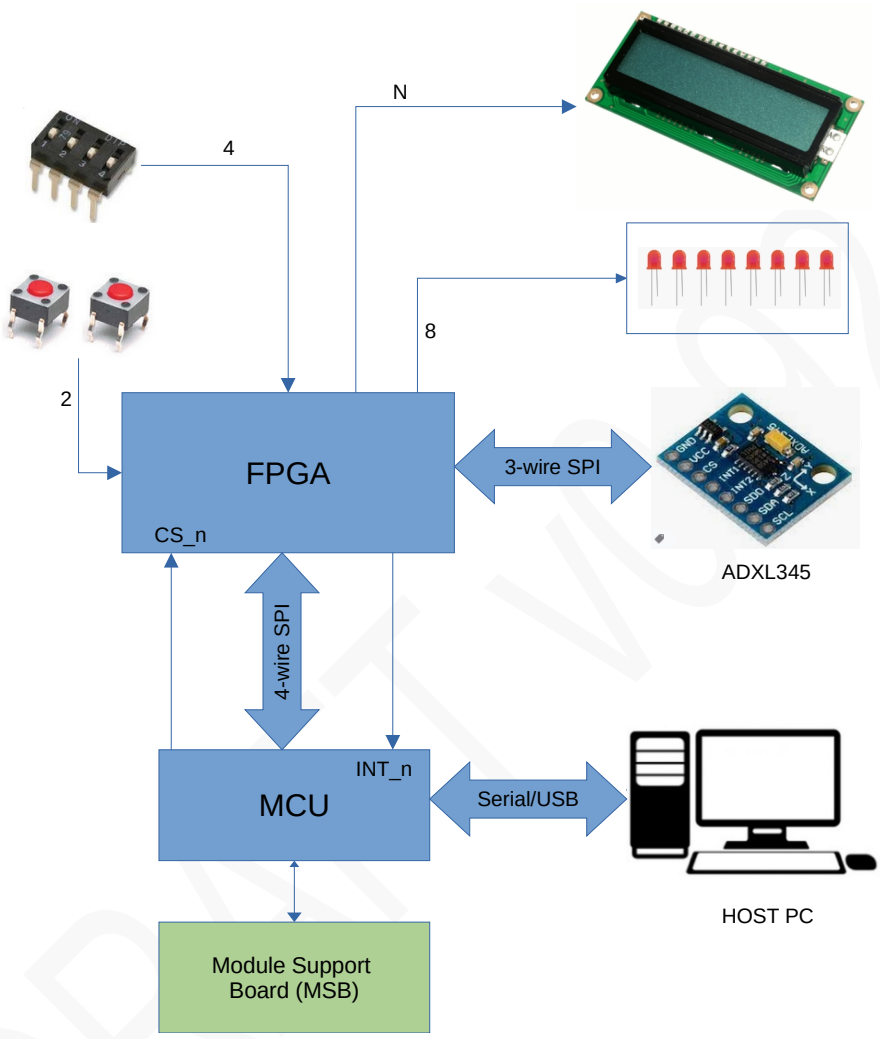


Figure 1. Block diagram. The MCU communicates with the FPGA using 4-wire SPI. The FPGA can host one or more SPI slave components, each with a separate Chip Select (CS) line. There are a number of spare GPIO lines which the MCU can use as chip select (CS) outputs or interrupt signals (inputs).

The FPGA

You are going to be provided with a number of dataflow components. Connection to peripherals will also be pre-configured.

Interfacing to the MCU using the Serial Peripheral Interface (SPI)

The MCU is interfaced to the FPGA using a **4-wire Serial Peripheral Interface** (SPI), where the MCU is known as the master device and the FPGA presents itself as one or more SPI slave devices. The MCU is able to send data and/or instructions to the FPGA, and where required, read a response back. 4-wire SPI is a **full-duplex** interface, so data can be simultaneously written and read. When working with multiple devices, there are a few options. Two are considered here:

One master device + multiple slave devices

Simpler designs can use multiple SPI slave devices, whereby data and clock signals are shared and each additional device has a dedicated **Chip Select** (CS) line.

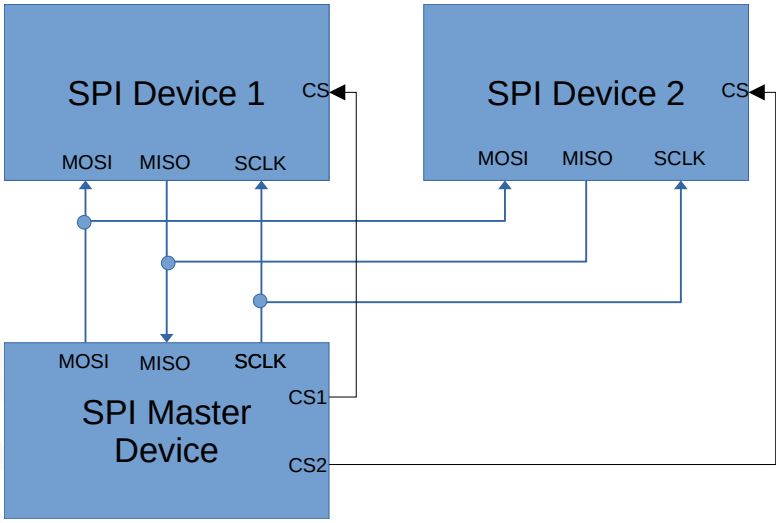


Figure 2: Interfacing to two devices using separate SPI Slave Components. Here is assumed each device has it's own dedicated SPI slave component.

The advantage of this scheme is that it is simple. The disadvantage is that each additional device needs a separate SPI slave unit and a dedicated GPIO output for the chip select. You also need to be careful to ensure that ONLY one device is selected at a time.

One master + one slave device

More advanced designed can implement everything using a single SPI Slave Device.

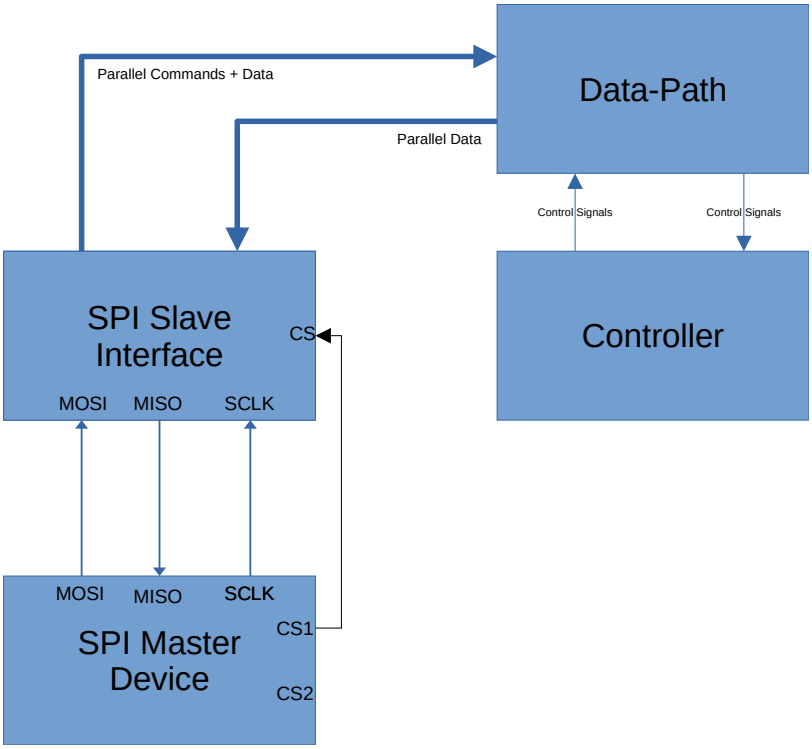


Figure 3: Using a single SPI slave device to issue commands and transfer data. The data path may include any number of peripheral devices. The controller is typically a stage machine.

This approach is more complex. It is somewhat reminiscent of a microprocessor decoder. The MCU will typically first send a command (and possibly ignore anything returned). The controller will then decode the command through a sequence of steps typically involving multiplexers and standard components. The resulting data is then returned.

Full Duplex Communication and SPI Transactions

One of the attractive features of 4-wire SPI is that you can both send and receive data at the same time (full-duplex). However, unless the design is very simple, we often need to consider the timing. Consider the following examples:

Example 1 – The MCU requests to read a value stored in "register 1" (an internal register in the device). It sends a unique numerical **command** that represents "read register 1". As it does this, data is also returned, but this data has no meaning so is ignored. Once the command is fully received, the controller routes the register to the SPI parallel data output using multiplexers. This is a fast operation. The MCU can then issue a second command (commonly we send a 0 to simply represent "read the result of the previous command"). The data returned in this transaction is the register data. Therefore, it took two SPI transactions to perform a selective read.

Example 2 – The MCU requests to read an ADC. This is not such as fast operation and takes several clock cycles to complete. It sends a unique command that represents "Start ADC". The controller then begins to perform the necessary operations to acquire a sample and store the result in a designated register. When it is complete it also sets the "ADC Ready" bit HIGH in a designated status register. The MCU must then request to read the status register and checks the result. If the ADC has finished, it can then ask to read the data itself. As you can see, this can take multiple transactions. Although SPI is full-duplex, it is common that the transactions are skewed (typically where the returned data is the result of the previous command[1]). This is depicted in figure 4.

1 With some ingenuity, you can sometimes issue a command where the first few bits contain all the information needed. The slave device can then rapidly assemble the return data so it can be read in the remaining time of the same transaction.

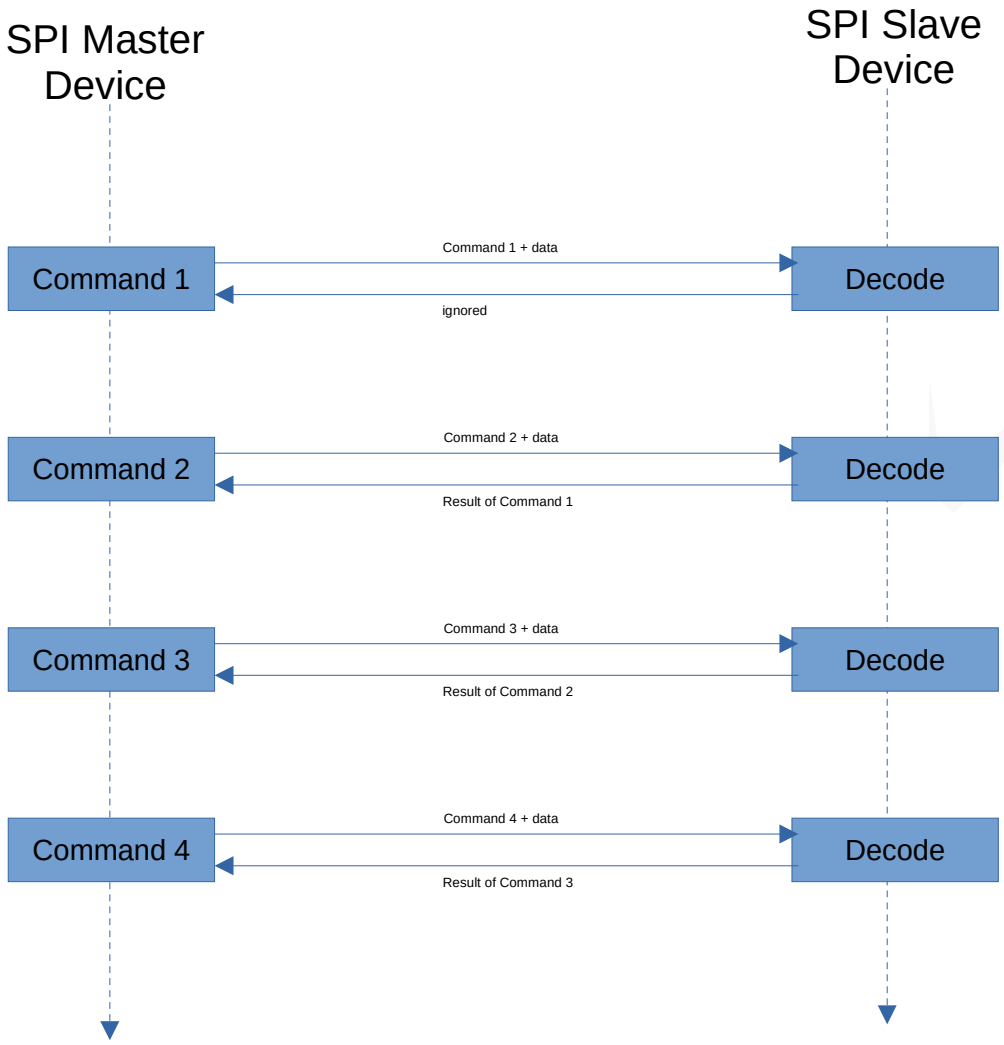


Figure 4: SPI Transactions are FULL DUPLEX. However, they are often skewed as commands need to be first received and operations performed before data can be returned.

Data-path Components

Within the FPGA, you are strongly encouraged to follow a controller data-path architecture as depicted in Figure 5 (there are more marks for this).

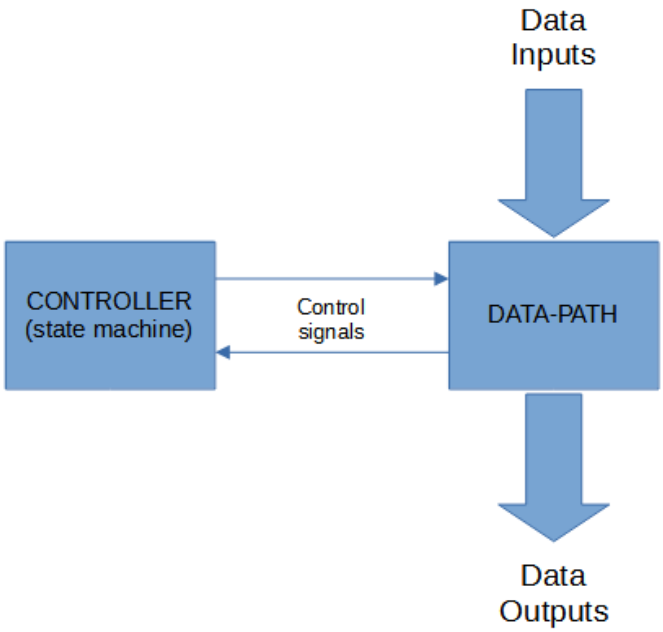


Figure 5: Controller data-path architecture

Peripherals (data inputs and outputs)

There are 5 peripherals to consider.

- Two push switches referred to as KEY0 and KEY1. Note that KEY0 is used to reset the FPGA. KEY1 is available for other purposes.
- Bank of 4 Switches
- Bank of 8 LEDs
- An accelerometer (connected via a 3-wire SPI interface – this is provided)
- An LCD display (connected via a parallel interface)

Generic Components

You will find a number of data-path components already included in the examples provided. These include:

- 4-wire SPI slave component (for interfacing to the MCU)
- 3-wire SPI master component (for interfacing to the accelerometer)
- N-bit Registers for storing values
- Multiplexers for routing signals
- FIFO Buffers (SCFIFO in the Altera library)

You should NOT create your own SPI interface components.

Controller Components

In lab "08-CaseStudy-SPI", a controller is used to orchestrate communication between different data-path components. If you are ambitious, it is suggested you study this closely. It also **decodes** instructions from the SPI interface so that the same SPI slave component can be used for multiple purposes. You are welcome to adapt this example for your own purposes.

Assessed Tasks: Overview

There are three elements to this coursework.

1. Research Task (on SPI and I^2C interfacing) – 20%
2. Technical Task – 60%
3. Group and Individual Management – 20%

All three require some submitted work and are presented by the group at the end in a group viva.

Peer Assessment

All students are expected to contribute to **all** elements of this work. For example, you cannot just to research and management and avoid VHDL.

At the end of the module, you will be asked to assess the contribution of each other in all three elements. This will be backed up with meeting notes and GitHub submissions.

Each task will now be outlined in greater detail.

Task 1 – Problem Based Learning on SPI (20%)

In this task, your group will need to research two technologies and give a 15 minute presentation at the end. You will not be able to overrun.

The process you are to follow must be adhered to, and you must keep notes.

1. Read the "problem statement"
2. Identify all terms and concepts you do not yet understand
3. Each person must research all items in (2) and take notes
4. You meet as a group, compare notes and discuss – then try to reach some form of consensus
5. Go back to 1 until you are all happy that you have reached a deep enough understanding to present your findings
6. Write a PowerPoint presentation that must last no more than 10 minutes. Allow a further 5 minutes for questions.

Problem Statement

You are working on a new product design which employs a **MCU** and **FPGA**. You are to **interface** an **FPGA** to an **accelerometer** device (e.g. ADXL345) so that movement data can be **evenly sampled** in each the three directions (x, y, z). Such devices typically offer three different interfacing options: **4-wire SPI**, **3-wire SPI** or **I^2C**. You are to present a summary of each as well as the advantages and disadvantages of each, such that you can make an informed choice about which is the best to use.

If using SPI, you will also need to know the **mode of operation**, or to be specific, the **clock phase and polarity**. Present a **timing diagram** to explain this.

Compare and contrast 3-wire SPI, 4-wire SPI and I^2C. Present the different benefits and disadvantages as appropriate. Also consider any legal factors. Is this licensed (patented) technology? You do not have much time, so keep the presentation factual, very focused and concise.

What to Submit

Power point presentation – maximum of 10 slides

Task 2 – Reaction Time System (60%)

In this coursework, you are going to build and test a number of subsystems that together, will assist cognitive psychologists in conducting reaction-time experiments. To give this some context, the basic idea is that a participant reacts to a stimulus (light or sound), either by pressing a button or tapping the desk (detected by accelerometers). The timing between the stimulus and the reaction time (in ms) is measured and recorded. The intention is to discover if the push button or accelerometer add a different bias to the results.

Device	Purpose
DIP Switches	Select the experimental configuration
KEY0	Reset the FPGA
KEY1	Used as an input for the participant
Accelerometer	Alternative input for the participant
LEDs	Used as a stimulus for the participant to react to
Buzzer*	Used as a stimulus for the participant to react to
LCD	Display reaction time results
Blue User Button**	Used to set and start experiment
Black user button**	Reset the system

*Located on the module support board. See example code on how to use it. **Located on the MCU board

Experimental Design

The basic experiment always follows the same basic pattern.

1. The operator selects the experimental configuration using the DIP switches (see below).
2. The operator presses and releases the Blue button to confirm the choice
3. The LCD informs the user how to respond (button or table tap)
4. User presses and releases the Blue button to start the experiment
5. At random intervals (between 10 ms and 2000 ms), a stimulus is provided (LEDs or Buzzer) N times (where N can be changed)
6. The participant reacts as quickly as possible (using KEY1 or the accelerometers)
7. The reaction time is recorded each time
8. When complete, the average of all N reaction times is displayed on the LCD

Experiment Configuration

Using the DIP switches, there are a number of configurations the psychologist can select:

DIP SWITCH	**Function**
DIP0	0=>Buzzer; 1=>LEDs
DIP1	Input select: 0=>KEY1; 1=>Accelerometer
DIP2	Speed (N_0)
DIP3	Speed (N_1)

The number N is set with DIP2 and DIP3: 00=>10; 01=>15; 10=>30; 11=>60

Formal Assessed Requirements

For the technical aspects of this work, you are marked on how many requirements you have achieved **with evidence** (see section below). The more you can evidence, the greater the mark in this section.

Display (15%)

DISP1. It shall be possible for the MCU to send characters to the FPGA over SPI so they are displayed on the LCD Screen. All LCD display timing must be performed by the FPGA. It is suggested you use full 8-bit communication.[6%]

DISP2. Characters and commands should be buffered on the FPGA (use a SCFIFO component) in order to give the MCU additional timing slack. [3%]

DISP3. It shall be possible for the MCU to send commands to the FPGA over SPI to clear the screen, set the row and set the column.[3%]

DISP4. It shall be possible for the MCU to read the space available in the character FIFO before sending (It is the responsibility of the MCU to not fill the character FIFO)[3%]

Accelerometer (15%)

ACC1. It shall be possible for the MCU to send commands to the FPGA over SPI to initialise and start the accelerometer (use the example code to see how this is done in software). For full marks, this should be a single command whereby the detailed initialisation is performed by the FPGA (and not the MCU as in the example).[6%]

ACC2. Accelerometer data shall be buffered in a FIFO on the FPGA to give the MCU additional timing slack. It is suggested to use the SCFIFO component provided by Altera/Intel.[3%]

ACC3. It shall be possible for the MCU to read ALL the buffered accelerometer data via the SPI interface. The first value should equal the number of samples to follow, followed by the actual samples.[3%]

ACC4. If the FIFO buffer(s) in ACC2 were to become full, this should cause the LEDs on the FPGA board to flash to indicate a critical error (until the board is reset).[3%]

DIP Switches and KEY1 (5%)

DIP1. It shall be possible for the MCU to read the DIP switch positions and the state of KEY1 via the SPI interface. As switches are asynchronous, you should also take steps to reduce the probability of a switch causing metastability (see lecture on meta-stability).[5%]

LEDs (5%)

LED1. It shall be possible for the MCU to send commands to the FPGA over SPI to control the state of the LEDs.[5%]

FPGA Architecture (5%)

FPGA1. You should use only ONE 4-wire SPI Slave device on the FPGA to control multiple devices on the FPGA.[2.5%]

FPGA2. You should employ a controller-datapath architecture in your FPGA system design.[2.5%]

MCU Software (15%)

SOFT1. You should write software, in C/C++ using Mbed OS, so perform the experiment as described in the section "Experimental Design". Demonstration of each step carries equal marks. Consider using mocked data if the FPGA hardware is not available or working [10%]

SOFT2. Your software should be correctly structured, including: [2.5%] Indention

1. Appropriate factoring into functions and separate C/C++ and header files
2. Commenting (top of every source file, at the start of every function or logical block, and line level unless the code is self-describing)
3. Use of meaningful variable and function names

SOFT3. You should optimise performance by avoiding excessive blocking through techniques such as rapid polling and/or the appropriate + safe use of interrupts. [2.5%]

Methodology and Evidence

To be awarded marks for a given requirement, you need evidence. There are two key documents you need to provide in order to evidence the extent to which you have met each requirement:

1. **Verification Record Document** – short document, recording the extent to which each requirement is met + any anomalies / observations. Each requirement should be itemised, giving the actual result.
2. **Verification Specification Document** – a sufficiently detailed description of **how** you evidenced **each** requirement (known as experimental **methodology** in scientific circles). It should be possible for an independent engineer to follow this specification and reproduce your results in (1). You should create an itemised section for each requirement. State the procedure to follow and state the *expected* results.

There are various ways to approach this, and sometimes you need to be quite creative. They might include:

- Simply operating the device and observing output (e.g. SOFT1)
- Using signal tap to generate detailed real-time output
- Stepping through code and observing outputs
- Logging data to the serial output (on the MCU) or LCD
- Mocking – sometimes, you can provide "fake data" to evidence a particular subcomponent. For example, the MCU software can use fake data so that it can be tested without the FPGA; you can create separate FPGA projects to just test a particular subsystem.
- Code inspection in some rare cases (e.g. SOFT2)

What to submit

The following should all be archived into a single ZIP file: All Quartus and Mbed Studio Projects, such that the tutor can build and test your submissions independently.

- Verification Record (MS Word or LibreOffice)
- Verification Specification (MS Word or LibreOffice)

Other evidence may be accepted.

Task 3 - Group Management (20%)

Group work is a learning outcome of this module and is assessed. You are assessed on how each group manages itself.

Each group should appoint a team leader.

- The team leader should create a team using Microsoft Teams with the name ELEC241-2021-P1-GroupXX, where XX is the group letter. Group members should be added to that team.
- The team leader should create a **private** repository on GitHub and add all members to the team.

All group activities are to be recorded in shared OneNote and Planner documents and will be monitored by the module team.

All code should be stored on GitHub. For everything else, use Microsoft Teams to store common files and notebooks. Third party tools will not be accepted as evidence of groupwork with the exception of GitHub.

You must use your University account to sign into Teams.

Formal Group Meetings:

There should be **at least** two formal group meetings each week. Where face-to-face meetings are not possible, you should use **Microsoft Teams**. Notes from each meeting shall be recorded on a separate page in a single group OneNote document. For each group meeting, the following minimum information must be recorded by the group manager:

- Who is present, and who is not (with any apologies given and reason for absence if provided). **Include their GitHub names.**
- Items from last meeting
    - Summary report from each individual mapped against the assigned actions
    - If no progress has been reported, record that also (possibly with reasons)
- Any Discussion points.
    - Consider the actions set in the previous week and reflect on how viable they were. Was it too much to ask, or maybe the individual needs some assistance?
    - Consider each action and readjust the work assignments. Don't be afraid to put two people on a single task so they can work together and learn from each other.
- Consider this as a good time to review each others code
- List of Actions for next week
    - Reallocation of tasks against individual names
        - You are under time pressure, so try to prioritise and assign your group members as efficiently as possible.
        - Support those who are struggling.
        - Consider pair programming, especially where one person can learn from the other
    - Updates to the Kanban Board (**paste in a screenshot or PDF into each meeting note**)

Item Tracking with Kanban

You are to use Planner to maintain a Kanban Board to keep track of all outstanding items, assign names to tasks and to help visualize workload.

- This must be updated and recorded at the end of each group meeting.
- Try to avoid asking anyone to multi-task more than 2-tasks at a time.

Version Control

You are to use version control to manage both individual changes and to consolidate code as a group. This applies to both VHDL and C/C++ files. Some tips:

- Get into the habit of branch-modify-merge. Try to avoid making too many changes between version control commits.
- Use version control as best practice for both personal and group activity.
- Make use of branches to keep individual work separate and to avoid breaking the master branch.
- Use pull-requests to merge changes into the master branch
- Learn to use tags to mark specific milestones

Best Practice Tips

- Make use of test benches to test the components you create. Always use assert commands to automate testing.
    - It may be very tempting to create a component and try to integrate it with others. Resist the temptation to do this before it is tested! **Always create a test-bench and test it first**. *This may provide vital evidence towards meeting some of the requirements*.
- Consider creating separate projects to develop and test individual requirements. You might find this surprising, but you can score very high marks this way, even if you don't integrate everything together.
- Make sure you check in your code regularly. Review what you have changed. When something breaks, this can provide vital clues.
- Consider pair programming as an efficient way to allocate human resources. Use the screen sharing facility of Microsoft Teams where face-to-face is not possible.
- Use mocking. Sometimes you want to test a component, but other parts of the system are not finished. In such cases, find creative ways to mock real data. VHDL simulation is very suited to this, but you can even store data in an array (ROM) and test on real silicon.
- Use signal tap to test on hardware if needed
- Bring signals out onto external pins and use your Picoscope to inspect real-time signals
- If you have developed a component, maybe suggest someone else tests it. They are more likely to find errors than you.
- Consider getting someone else to read through your code, or maybe, talk it through with them. This will help pick out errors.
- Use Microsoft Teams for group communication (voice, video, chat). You must be signed in with your University credentials if this is to be used as evidence of group work.

You may include additional documents as evidence as you see appropriate. However, it must be made clear to the examiners how to navigate this. Remember, you must explain how you verified each requirement and where the evidence is located.

The Viva

An online viva presentation will be booked once the last piece of work is submitted. A link will be shared and the team-leader will select a slot. The viva will take place on either Teams or Zoom.

There will normally be two examiners at the viva. In the event there is only one, then the viva will need to be recorded.

During the viva, the following structure will be followed: Introductions and sound checks (5 mins)

- Presentation on SPI and I2C (10 mins)
- Q & A (5 mins)
- Students to present evidence of technical requirements met (30 mins)
- Students to answer questions on group management (5 mins)
- Wrap up and feedback (5 mins)

Note that all group members will need to be present. If they are not available, they may be asked to attend a separate session individually.