Australian
National
University

SYNCHRONY

Assignment for all students of Real-Time and Embedded Systems 2019

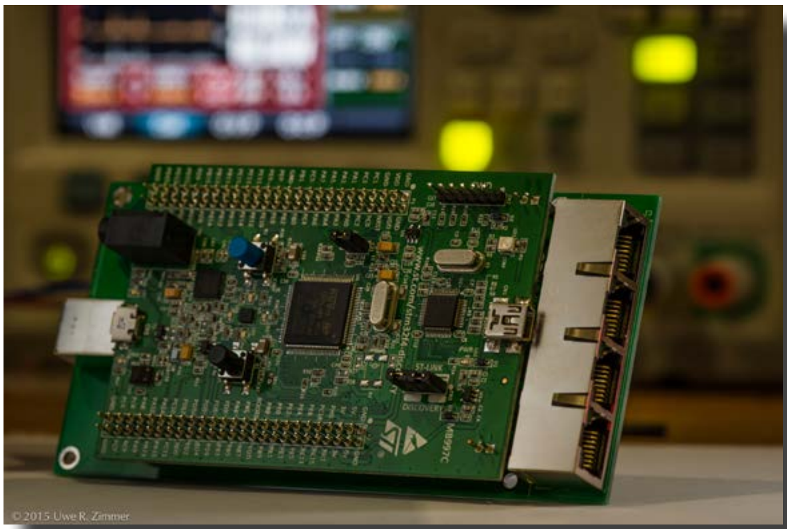
This is a carefully evaluated and marked assignment. Thus extra care is expected on all levels.

Overview

This central assignment for the real-time and embedded systems course is about keeping things in synchrony. The example at hand are local computer nodes which are running a heavy set of machinery (like a power generator) which requires to be synchronized to a global power grid. Small variation in the phase of power plants can cost lots of money in lost energy or worse, result in a breakdown of the power grid. It is therefore essential to keep many nodes over a complex network in smooth synchronization. Potential issues are oscillations and other forms on unwanted feedback ripping through such a system.

The hardware

What you have at hand is actually not a power plant, but something more handy. The embedded system which you can exploit in any way you see fit for this problem is an STM32F4 development board with an additional break-out board which allows you to comfortably connect boards by RJ45 cross-over cables[1].



Do not be fooled by its physical size: The STM32F407VGT6 micro controller is a power house of functionality. You will need to make close friends with its data sheet and reference manual (linked from the course-site).

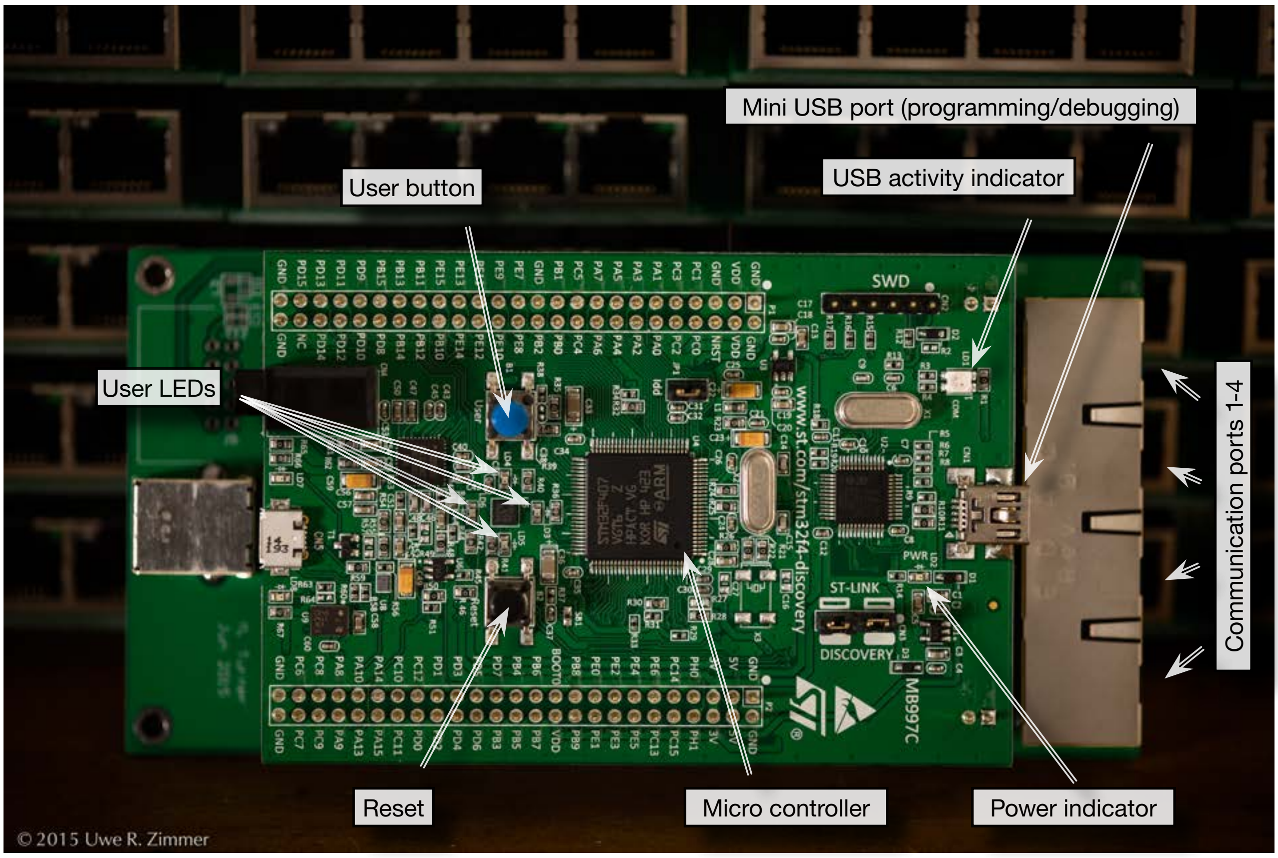
Some major components have been modelled for you in Ada on a highly type-safe level - nevertheless you are handling physical hardware and nothing prevents your program from full access to all physical components of the system. So it is easy to do damage – always think before you test.

General electrical considerations apply obviously. If you never handled hardware before you should talk to your tutor first to collect some advice about treatments which bare hardware does not take kindly to. If you are the constantly-statically-charged type of person for instance, you will need to ground yourself[2] before touching any boards.

1 While the connector plugs may remind you of Ethernet plugs, do not fall for the temptation of plugging them into your computer via Ethernet - resulting smoke on either side won't be covered by us.
2 If you think this means you are not allowed to leave the house, then you need to talk to us about some electronics basics.

Basic setup

Place the board on solid ground and make sure nothing metallic can touch it. Connect the Mini-USB cable to the lab computer (or your own computer) and two lights (close to the USB connector) should come up already[3]. At this stage the board will automatically execute whatever is to be found in its current flash memory. You can reset whatever is executed by



pressing the black button (after which execution will automatically start over again).

On your computer you now issue the command `st-util` from a commandline which should result in something like this:

```
INFO src/stlink-common.c: Loading device parameters....
INFO src/stlink-common.c: Device connected is: F4 device, id 0x10016413
INFO src/stlink-common.c: SRAM size: 0x30000 bytes (192 KiB), Flash: 0x100000 bytes
(1024 KiB) in pages of 16384 bytes
INFO gdbserver/gdb-server.c: Chip ID is 00000413, Core ID is  2ba01477.
INFO gdbserver/gdb-server.c: Target voltage is 2883 mV.
INFO gdbserver/gdb-server.c: Listening at *:4242...
```

If this command is not installed on your own computer (it is installed in the labs) or does not work as above, turn to the *instructions on the AdaCore website* on how to install this driver in Windows or Linux.

A green light close to your Mini-USB plug indicates that a communication link ready for programming has been established. Your Cortex-M4 processor and the peripherals have been stopped, but not reset at this point. Before you continue and program it with the next steps it is

3 If not, then unplug your board again and talk to your tutor.

a good idea to press the black reset button on the board to set the processor to a predictable state[4].

From inside your arm-gps environment (in the labs you start this IDE from the command line by saying e.g. `arm-gps&`) you can now initiate the upload of your program (after you built it of course). You do this via the menu Debug ➟ Initialize ➟ <name of your main program>. You will see some action on the command line which should end in a "`Flash written and verified! jolly good!`", telling you that your upload command was successful.

Your program is now ready to run and will start with the menu command Debug ➟ Continue[5]. There is no performance or predictability penalty from starting your program via the debugger.

You conclude your test by releasing the communication link via the menu command Debug ➟ Terminate. This will also stop the `st-util` command.

Your program is now stored in flash memory on your STM32F4 which means that it is still there and will run automatically after a reset or power cycle.

Debugging your program

After you initialized the debugger, your IDE interface will change a little you will see debugging options. This is basically a graphical interface to `gdb` and those who are used to debugging in `gbd` in the command line can continue to do so by typing in `gdb` commands on the debugger console.

You may have noticed that you now see dots to the left of every line of your source code. If you click on them you will have set a breakpoint at this spot in your code. Many other options of debugging and data inspection are now available. Yet this assignment cannot be an introduction to general debugging techniques, and so I'll leave it to you to explore or use it in the way how you always did. You may well not need the debugger after all and your programs could also just work.

Software framework

The framework project which you downloaded from the course web-site and already use in your first initial steps above provides you with a set of hardware interfaces to implement most of the concepts which you may think of in this assignment. The `Sources/STM32F4` directory contains enumeration types and register definitions to directly access essential components of your micro controller. You will likely want to check out at least those core ones:

- **STM32F4**: Some basic type definitions which are used across multiple sub-modules.
- **Reset and clock controller**: You will need this module to activate the components which you want to see active (i.e. provided with a clock signal). There is fine grained control over what clock signals are being generated and who receives a clock signal. As you can for instance see in the code provided, you cannot use general purpose IO lines before you enabled a clock signal for the specific port to which this IO line belongs.
- **System configuration controller**: This module is required to associate interrupts with specific IO lines (among a few other features).
- **Interrupts and events**: You will find the register in there to configure and enable specific external interrupts. You can also clear an interrupt there - once you dealt with inside your interrupt handler.
- **General-purpose I/Os** (or **GPIOs** for short): Plenty of options to configure the general purpose external connections of your micro controller. Particularly important is to configure the electrical details of your pins correctly – for instance whether a certain line is driving a signal or detecting a signal. All of your communication lines and enablers signals are connected to GPIO pins. The file `ANU_Base_Board.Config` shows you what goes where.

4 Continuing without resetting here is a bit of a gamble as the processor could be in a state which makes it impossible for the debugger to start a program.
5 You may want to assign keyboard shortcuts to those menu commands as you will likely do this often.

- **Timers**: You have 14 timers with varying feature sets available. Find out which one can do what you need for your algorithm and how to configure it such that it does just that. Timers are complex and powerful in this controller and so you will likely spend some time reading up on them before you can use them to their full potential.
- **Universal synchronous asynchronous receiver transmitter** (or **USART** for short): These are handy hardware communication controllers which can take some of the work off your hands when you for instance want to combine incoming signals into bytes or want to buffer some data before you want to look at it with the CPU. You have six of them and four are already hard-wired to your communication lines.
- **DMA (Direct Memory Access) controller**: Timers, USARTs and GPIO lines can trigger interrupts which can directly drive DMA transfers. You have two separate DMA controller modules which can run eight streams each. DMA controllers in your micro controller can transfer data between memory locations, but also between periphery modules and memory.
- **Random number generator**: The by far simplest module which you saw so far, yet it could come in handy in case you would have some Monte-Carlo algorithm as part of your design.

This covers some of the basic modules inside your micro controller, but it is far from the end of the feature set available to you. Browse through the reference manual beyond those chapters above and see whether anything would be useful for your design.

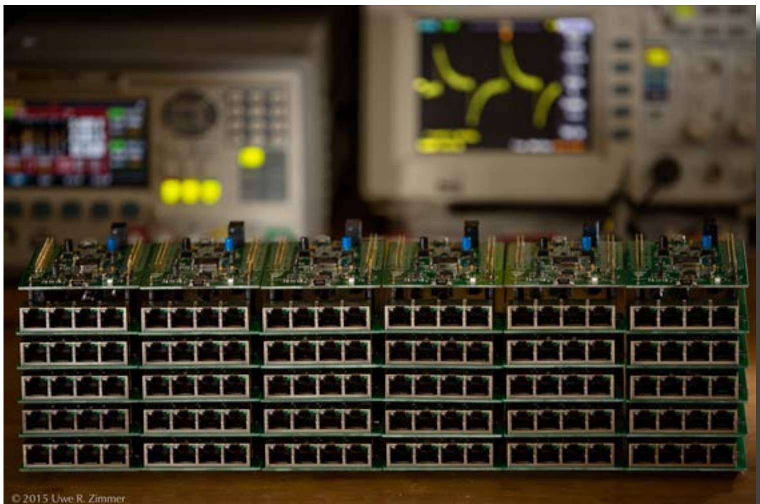
In order to understand in detail what your options are and how the basic components work you will need to work your way through parts of the reference manual (linked from the course-site). The schematics of the ANU-Baseboard which provides the drivers for communication lines are also available from the course-site.

Exercise 1: Synchronize oscillators

The task at hand is to synchronize (simulated) generators (oscillators). Those can be multiple oscillators on the same board or oscillators on separate boards. Assume that are all supposed to run at a fixed, absolute frequency (say 50 Hz), but their actual frequency will slightly vary and especially their phase will not be identical at all times.

Define a predictable, responsive communication interface which will be able to provide the required information exchange between your oscillators as well as a control algorithm which will adjust the frequency and phase of your oscillators such that the a whole network of such oscillators will run in close-to-perfect synchrony.

You have a lot of freedom in how you implement a convincing system. For starters think about how you would measure and display the frequency and phase shift on each board (compared to what it believes is the frequency and phase of up to four neighbouring modules. Those modules run the same algorithm and so the network could eventually span all 30 available computer nodes in the course.



If you meet your class mates in the labs (or anywhere) you should form groups where you can connect multiple boards and allow each other to upload each algorithm on each board. If you can agree on a common protocol you can also experiment what happens if the boards run different algorithms. The protocol of signal format which you use is all at your discretion, but nobody stops you ether from agreeing on a common format with your friends in the course. You still need to implement your code yourself in full or reference the code sections precisely where you relate to or where inspired by concepts of a class mate.

By the minimum you should discuss at least two different ways how to solve this problem in your report and you should have implemented at least one solution on your actual hardware.

The three main problems which you need to address are:

a. *Generate:* How do you generate and communicate a signal of a precise frequency?

b. *Measure:* How do you detect the frequency and phase shift of the incoming signal precisely?

c. *Control:* How do you adapt your own oscillator to eliminate the detected offsets?

A few potential issues which you might consider in your algorithm design:

- Is the predictability of your system dependent on the sequence and times of incoming signals? For instance: are there any resources (which are handling events) which could be busy sometimes and available at others?
- Will your overall frequency stay close to the target frequency or will it potentially drift far away from it?
- Can you avoid oscillations in the control loops themselves? For instance, two neighbouring nodes could constantly shift their frequency up and down in an alternating fashion (in an attempt to adapt to each other) without ever stabilizing to a common frequency. Symmetry might not be your friend here.
- Can you system stabilize if you have a network topology which contains cycles, like a ring, hypercube, mesh, torus, or butterfly network?

Exercise 2: Duty cycle

As an advanced extension to the previous task, you can encode additional synchronized information onto the same channels by introducing a duty cycle to your oscillator signals. This will require additional detection facilities. In contrast to the common frequency and phase across the network, there is no need to keep the duty cycle at a given global value and individual nodes could attempt to drive the common duty cycle up or down to their locally preferred value.

Deliverables

The assignment will be assessed in form of an interview. To prepare us for your algorithms, your implementation(s) and your experiments, you are required to submit a report about your design. This has also the effect that your concepts are well structured in your head when you come to your interview … and that you may have a few diagrams which you can point to during your explanations. There are no constraints about the format or size of the report (we assume that report writing is a routine exercise for you) as long as you can communicate your basic concepts and findings. For those who did not have the pleasure of writing a report in any of my classes, please take note that I am notoriously sensitive to atrocious typesetting, pixelated diagrams, missing units and wrongly formatted graphs or tables. As a final year student, I expect a professionally looking report which won't make my eyes tear. (If you are unsure, there are a few hints posted on the course-site.)

Your code will also need to be submitted before the interview via the SubmissionApp. - Please only submit your zip-compressed Sources directory.

Support

If you run into trouble, do not hesitate to contact us straight away and anytime. The forum is a good place to get help (chances are that other students have or had similar issues) but direct e-mail to any of us works as well.