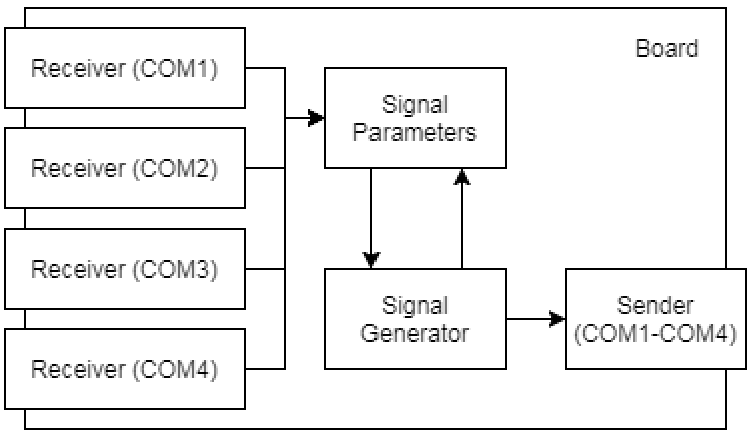
Australian National University

The Report of Synchrony

Assignment for all students of Real-Time and Embedded Systems 2017

Synchronize oscillators

The overview of the submitted design looks like the following diagram.



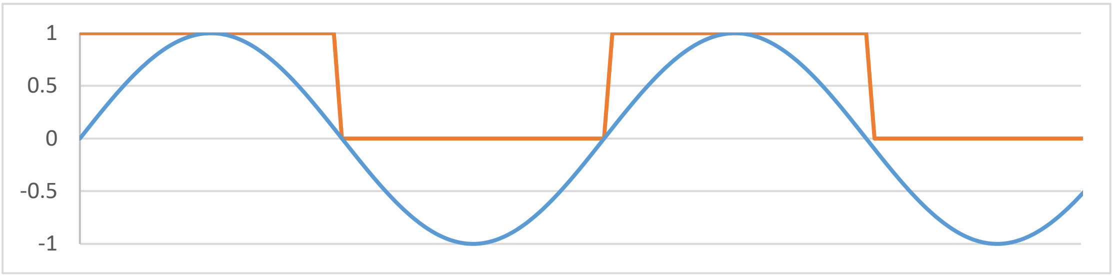
Signal Generator

The generator in this implementation could generate a sine wave. The implementation is using Taylor Expansion to calculate the sine value. It is using a 9th level Taylor Expansion to calculate the sine value, it is the minimum level that could describe a single cycle of sine. Parameters of all levels are pre-calculated (P_i), The value of $\sin(x)$ is now

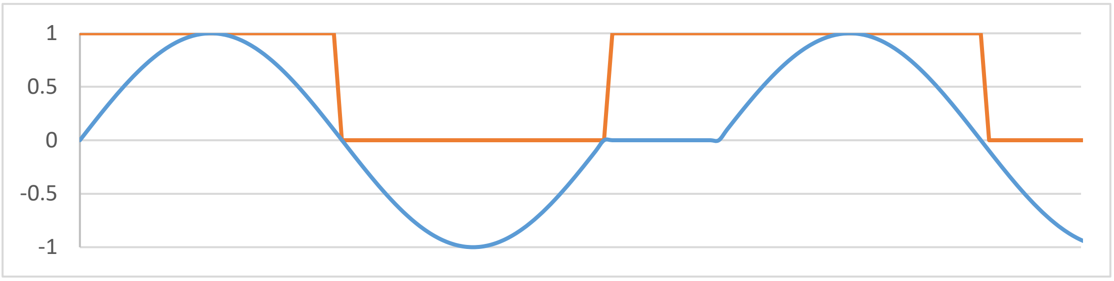
$$\sin(x) = x + \sum_{i=1}^{9} P_i \times x_{i-1} \times x^2$$

and $x_0 = x$, $P_i = \frac{(-1)^i}{(2n+1)!}$. The calculation of a single $\sin(x)$ is fixed, which means it is predictable.

The sine wave has been change into a square wave to send. When the sine value is less than 0, all the values will be treat as 0, or else 1. The following diagram shows the square wave to transfer of the example sine wave.



The synchronize of the square wave will sync the sine wave. The way to synchronize is to delay the signal generating at the end of one cycle. For example, to delay the above sine wave a little bit, the generator will delay the next cycle to the specific time.



In the code, this is implemented as two *delay until*. It will first delay to the expected cycle time. For instance, a 2Hz sine wave, the expected cycle time will be 500ms (suppose it starts at 0ms). Then it delays to the tweak time. For instance, the 2Hz sine wave needs to delay 20ms to sync with another wave, it will delay to 520ms. The pseudo code is:

```
    delay until 500ms; -- (Called as expected cycle end time)
    delay until 520ms; -- (Called as real cycle end time)
```

At the beginning of each cycle, the generator will first calculate the expected cycle end time, and reset the real cycle end time to the expected end time. If the expected cycle end time is the same the real cycle end time, it means the wave do not need to sync.

To send the data, the generator will calculate the sine wave in each cycle as a sample frequency. The default setting is 16. For the 2Hz sine wave, it will calculate $2 \times 16 = 32$ times in one second, hence 32Hz.

After each calculation, the generator will request the sender to update the data. The data to send is the square wave of the sine wave.

Sender

The sender in this implementation is just a procedure to change the port output states. When the value is 0, it will call the *reset* for all the ports. When the value is 1, it will call the *set* for all the ports.

Signal Parameters

The signal parameters contain the following constants:

- π and 2 × π
- Sine Taylor Expansion level parameters and its range

The parameters contain the following values that could be changed to describe the sine wave. It is constant when flashing to the board:

- Sine wave frequency (Hz)
- Sample frequency for each cycle, default 16

The following value could be deduced from the above values:

- Cycle period (P_{Cycle})
- Half cycle period ($\frac{1}{2}P_{Cycle}$)
- Sample calculate period (P_{Sample})

All these values above are constant. The following variables are used to sync the wave:

- Current cycle start time (T_{Start})
- Current cycle expected end time (T_{E-End})
- Current cycle real end time (T_{R-End})

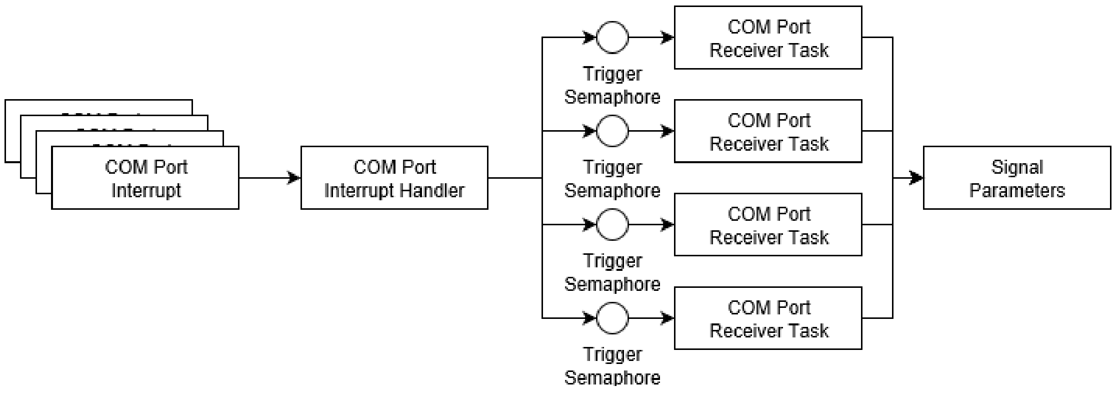
T_{Start}、T_{E-End} and T_{R-End} are updated at every beginning of each cycle. T_{Start} is set as the current clock time. For T_{E-End} and T_{R-End}, they will be reset to the following value after T_{Start} is set:

$$T_{E-End} = T_{Start} + P_{Cycle}$$
$$T_{R-End} = T_{E-End}$$

In each cycle, T_{E-End} is a constant value. T_{R-End} might be updated by receiver.

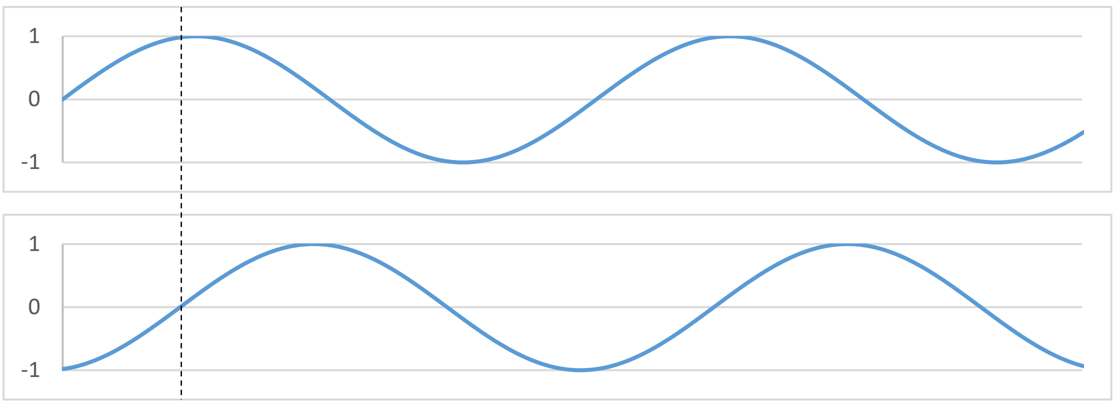
Receiver

The receiver is driven by the interrupts. The structure of the receiver is shown in the following diagram.

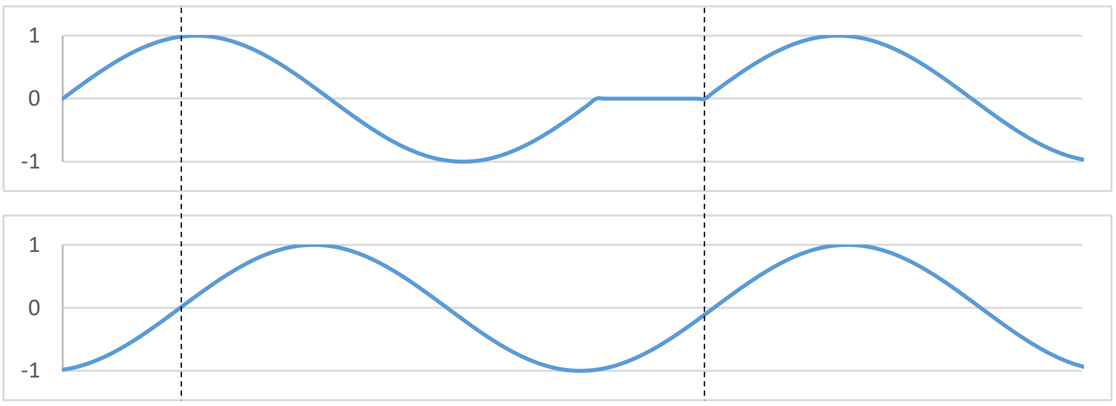


The same interrupt handler handles all the COM port interrupts. It will check all the ports and update the specific semaphore. For each port, it has a task and waits for the semaphore triggered.
The workflow of receiver task is simple. It will wait until the port meets a raising. Save the raising time T_{Raise}, if $T_{Raise} - T_{Start} < \frac{1}{2}P_{Cycle}$, then update the T_{R-End} to $T_{E-End} + (T_{Raise} - T_{Start})$.
Here is the detailed explanation of the workflow:

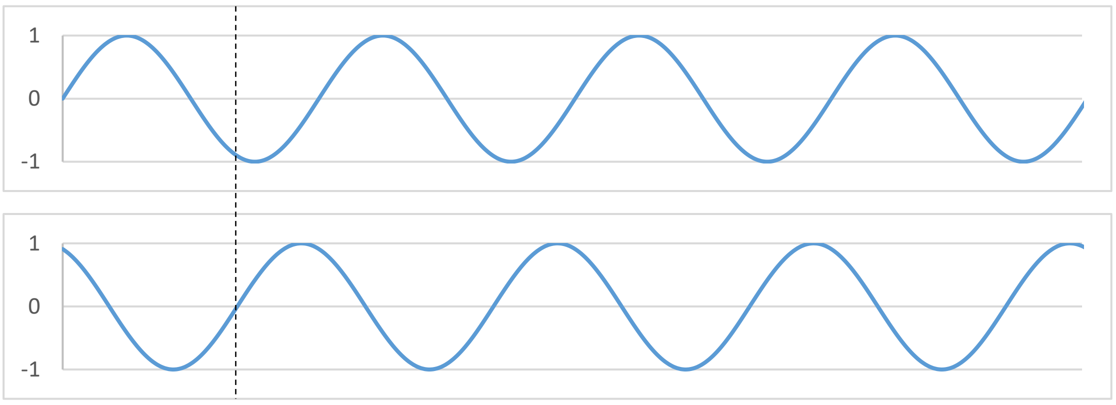
First take a look at two waves sync, suppose we have the two waves W_1 and W_2. There will have two condition if we looked from W_1: the cycle of W_2 arrives later than W_1 or earlier. First we take a look of W_2 arrives later than W_1. Suppose the above diagram is W_1, the below diagram is W_2.



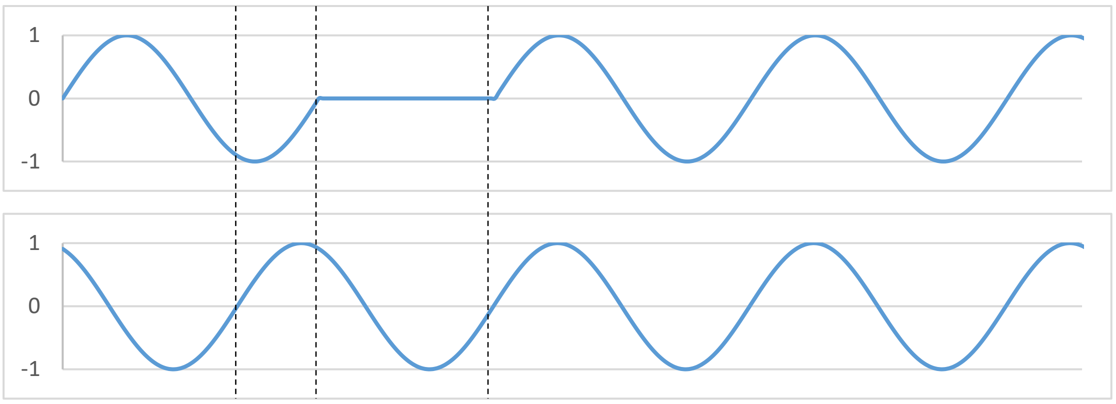
In the sync algorithm, it will let the forward wave sync with the later wave. In the example, W_1 needs to sync with W_2, and W_2 stays the same. W_2 will generate a raising at the COM port of W_1 and W_1 will delay to sync with W_2.



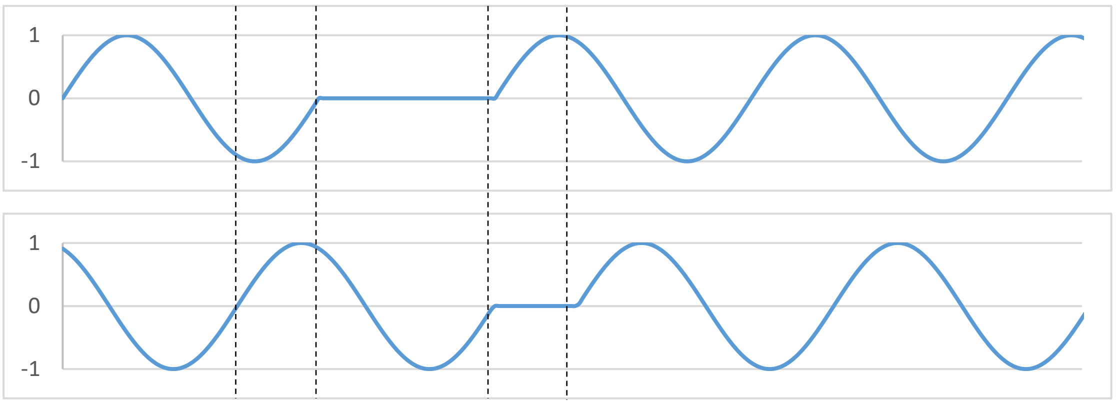
Now W_1 and W_2 are synced. Algorithm complete.
However, this will have a problem that is hard to define earlier and later. Consider the following two sine wave W_3 (above) and W_4 (below).



From the viewpoint of W_3, W_4 is later than W_3. According to the algorithm, W_4 send a raising at the time of the line to W_3, W_3 will take the sync and delay itself to meet the next cycle start of W_4, then those two wave would be

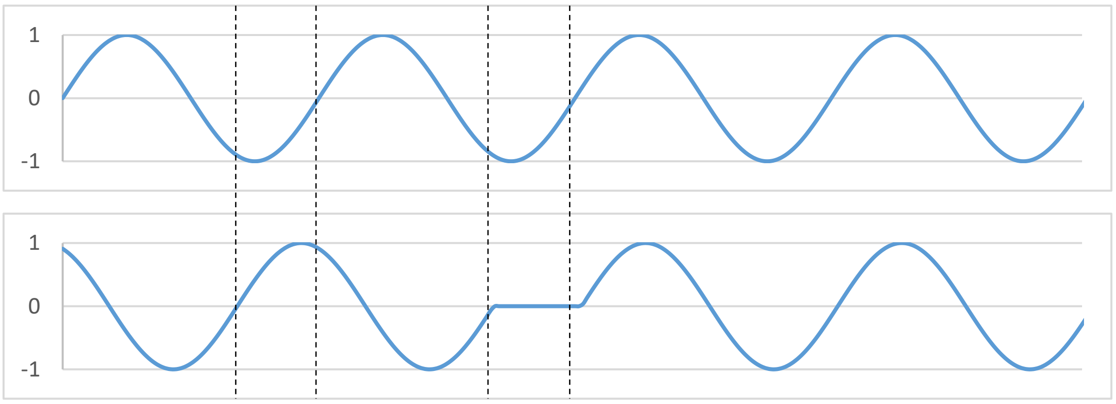


But the problem is when W_3 change the value from minus to 0, it will send a raising to W_4 at the second line above. Hence, W_4 will sync and delay itself of the time W_3 raising arrived to the cycle start. The following diagram shows this condition.



And this change back to the state that is not sync, and the result is the cycle frequency has been doubled and they are not synced.

To avoid this cycling synced, in the implementation the algorithm defines that if the raising of the incoming wave is later than the half cycle period, it will treat that wave as it arrives earlier than current wave, and ignore that raising. In the W_3 and W_4 example, W_3 detects that W_4 arrives at the second half of the cycle period, it will ignore the raising.



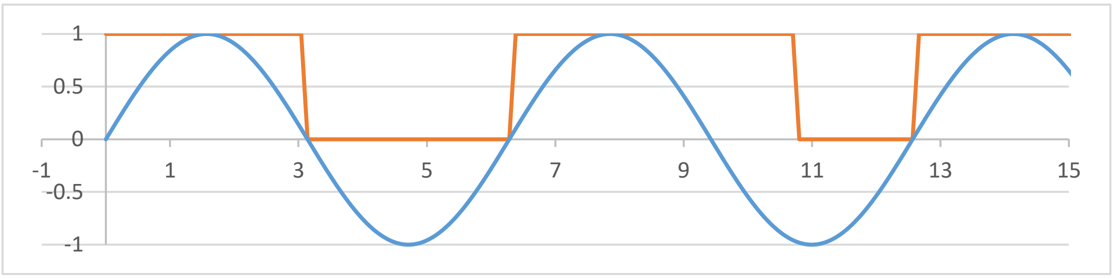
Then W_4 detects the raising from W_3, and it arrives at the first half of the cycle period, it will take the sync. W_3 and W_4 are now synced.

Each single port has a single receiver task works for it. Due to the missing of entry, the suspension objects are introduced to replace the busy waiting of watching signal. The interrupts of all the ports are set to only process the raising of the incoming signal.

As soon as an incoming raising appears, the receiver task will record its arriving time T_{Raise}, and compare the T_{Raise} to T_{Start}, if it is less than $\frac{1}{2}P_{Cycle}$, then it will update the global T_{R-End}.
What will happen if the T_{R-End} is written by two receivers at the same time? T_{R-End} would be the same of these two incoming signal, so the value T_{R-End} will remain correct.
What will happen when two values are synced? The incoming T_{Raise} would be the same as T_{Start}, hence $T_{Raise} - T_{Start} = 0$, then $T_{R-End} = T_{E-End} + 0 = T_{E-End}$, the wave will delay until to the exact the same as T_{E-End}.
For three waves, it works exactly the same. Consider three waves W_1, W_2 and W_3. W_2 arrives later than W_1 but earlier than W_3. W_1 would detect two raising, the earlier one is the one from W_2, later the W_3 one comes. W_2 would only detect one from W_3. W_3 detects no raising from W_1 and W_2. W_2 will sync to W_3 as what explained above. W_1 will first set its T_{R-End} to $T_{E-End} + T_{Raise2} - T_{Start}$ which T_{Raise2} stands for the time of raising of W_2 arrives. Later, the raising of W_3 arrives, T_{R-End} will be set to $T_{E-End} + T_{Raise3} - T_{Start}$ where T_{Raise3} stands for the time of raising of W_3 arrives. Hence W_1 will sync to W_3 as well. Above all, W_1 and W_2 will both sync to W_3. And W_3 will remain the same.
For four waves and more, this would work similar to three waves.
The predictability of the system dependent on the sequence and times of incoming signals. The interrupts might be busy, especially when all these signals get synced. As soon as the raising has been marked, it will be free to others.
This algorithm could avoid oscillations in the control loops themselves. Because all the stations are sync to the latest arriving signal, and the latest arriving station above all the system have methods to know that it is the latest one. Suppose a is the latest node started in the node network N, it will received all the raising at the second half of the cycle, and ignore all the raising. For all the connected nodes to a, the raising of a is the last raising arrives. Hence, all the connected nodes will sync to a and a remains the same as itself. And all the nodes directly connected to a would be the wave of a. And for all the directly connected nodes, this state is facing the recursively. Hence, all the reachable nodes in the network will synced to a. Thus, the algorithm is a topology free algorithm, which means it could work will all kinds of connected topology.

Duty Cycle

The implementation of the duty cycle is to simulate the PWM. The sine wave is described as a square wave now. The falling divided the 0 and 1 of a cycle of the sine wave. Originally, it describes the original sine value is greater than zero or not, but from now on, it is used to describe the value.



In the diagram above, the first cycle is describing 0.5. The one holds for half of the cycle, hence it is 0.5. The second cycle is describing 0.703125, because one holds for $\frac{45}{64}$ of the whole cycle. Calculate the time of the failing to the current cycle start and divided by the cycle duration, the passing value is out.

For this part, several things changes from the first part:

- Sender changed from a procedure into an independent task.
- Generator now only affect Sender implicitly. Only give sender a signal for sending the data.
- Receiver triggered by raising and falling.

The Sender is now triggered by a suspend object. When the Generator starts a new cycle, a suspend object is updated. The Sender will wait until the suspend object has been triggered, and it will calculate how long it needs to send 1, and then change to 0.

The Receiver is will detect it is raising or falling. For falling, it will simply record the duration from the last raising to this falling. When there is another raising, it will check whether the incoming signal is already synced with the Generator or not. If so, the value is valid and it could calculate the value by dividing the duration of 1 by the cycle duration.