HAN_UNIVERSITY
OF APPLIED SCIENCES

Group assignment

Distributed Systems

Minor Project

Documentation on Local Time implementation

Semester 2 Term 2

Abstract

In this document an implementation of local time in an STM32 board (Olymexino and E407) using HANCoder is presented. A hardware clock from the board is used as reference to generate interrupts in the code to update the value of the local ticks counter. The ticks of the local time have been measured using HANTune and processed in Matlab, with a result of $gr = 0.00011 \pm 0.00002$ s in the Olymexino board and $gr = 0.0001 \pm 0.0000$ s in the E407 board (100 μs). It has been found that the boards program crashes when the granularity set is 10 μs.

Group 1 - Global Time Team

Diego Martín López
Jordy Koole
Jamie Cheng

5th May 2021

1 Background

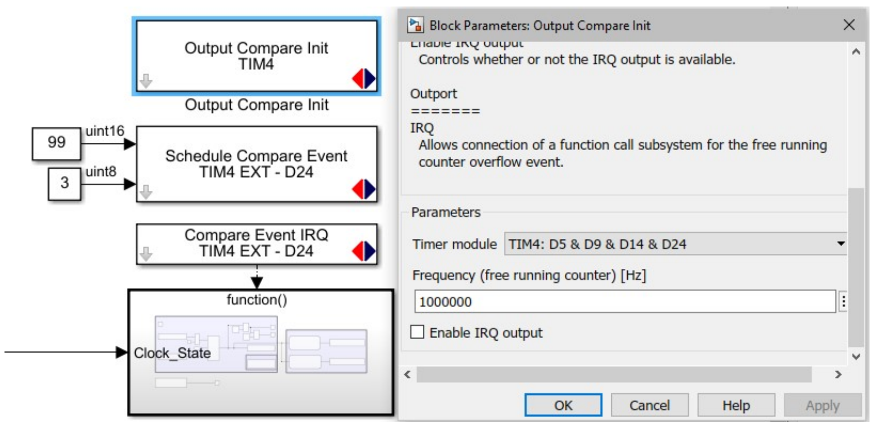
In this Minor Project we are interested on setting the fundamental blocks required for Time Triggered CAN (TTCAN) communication in a network of nodes coding the software within HANCoder. A first stepping stone on this challenge is setting the local time on a single node. The idea is to have a counter of *ticks* (*local_ticks*) that will be incremented every certain number of times a hardware clock ticks.

There are different blocks in HANCoder that already allows us to use a hardware clock as the source for our local time.

- **Output Compare Init**
  In this block it is possible to define the frequency of the free running counter (which is the representation of the hardware clock ticks in the software). The frequency chosen for this program is 1 MHz.
- **Scheduler Compare Event** and **Compare Event IRQ**
  The first input for the *Scheduler Compare Event* block is the value of the free running counter at which the *Compare Event IRQ* outputs an interrupt signal. The second input for the *Scheduler Compare Event* block allows changing the value of the physical pin of the counter. It is set to 3 to make no changes on the pin when the interrupt occurs.
- **Get Free Running Counter**
  This blocks returns the actual value of the free running counter.
- **Reset Free Running Counter**
  When this block receives a rising edge as an input the free running counter is reset.

It is important that all blocks refer to the same counter.

The strategy followed with these blocks is using the interrupt signal from the *Compare Event IRQ* block to wake up a function caller subsystem, where the local tick counter is incremented and the free running counter is reset. The blocks outside the function caller subsystem are shown in figure (1).



Enable IRQ output
Controls whether or not the IRQ output is available.

Outport
=======
IRQ
Allows connection of a function call subsystem for the free running counter overflow event.

Parameters

Timer module: TIM4: D5 & D9 & D14 & D24

Frequency (free running counter) [Hz]: 1000000

☐ Enable IRQ output

Figure 1: HANCoder blocks outside the function call.

2 Local Time counter

The function caller subsystem includes both the local tick counter and the reset for the free running counter. If the local counter is activated, every time the function is woken up it is increased in one unit and the free running counter (running at the hardware physical clock pace) is reset so the granularity of the local time is controlled. This granularity was set to 100 μs, choosing the number 99 as the first input for the *Scheduler Compare Event* block outside the function caller subsystem. The block connections inside the function caller subsystem can be seen in figure (2).

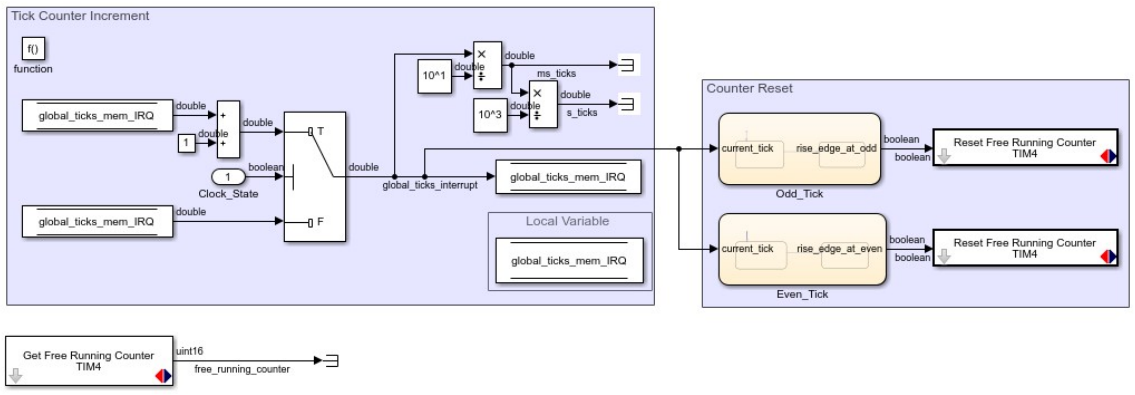


Figure 2: HANCoder blocks inside the function call.

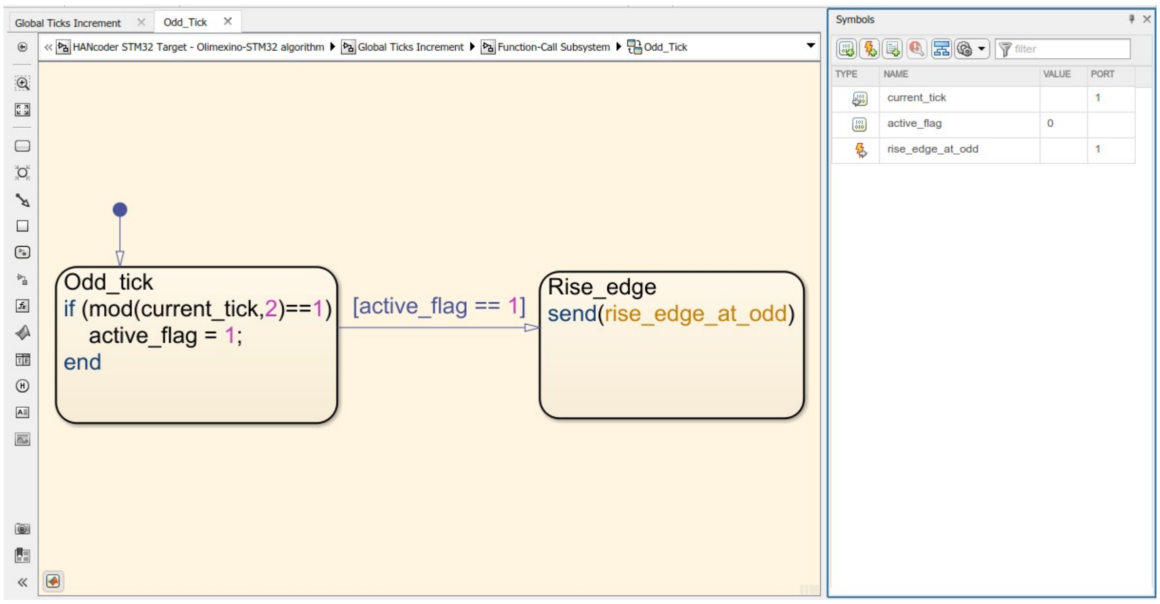
In order to activate the *Reset Free Running Counter* block a rising edge is needed. This is difficult to achieve inside a function caller subsystem, as there is no concept of running time to create a signal rising up from a previous value. It is necessary to create an artificial rising edge event.

3 Rising edge on function call subsystem

Fortunately, a way to create either edge events was found within simulink. These signals can be generated without a time basis using stateflow chart blocks. The code and window used to create this event signal is presented in figure (3).

Because every time these charts generate the output edge event it alternates between rising and dropping edges, it was necessary to combine two stateflow charts, making at least one of them generate a rising edge to reset the free running counter every time the function caller subsystem was activated. The two charts are in counter-phase, so when one is sending a rise edge event the other sends a drop edge event. This alternation is achieved checking the parity of the local ticks counter. When the counter has an odd value the first chart rises the edge, and when it has an even value it is the second chart the one rising the edge.

In order to set the output of the stateflow chart in simulink as either edge it is important to open up the window *Symbol* from the *View* panel. There are presented all the different variables, signals and events from the stateflow diagram. Each of them must be assigned to a particular category, such as local variable, input variable or output event. When right clicking the event choose *Inspect* to open up the *Property Inspector* where we can set the *Trigger* to be *either edge* instead of *function call.*



```
Odd_tick
if (mod(current_tick,2)==1)
    active_flag = 1;
end
```

[active_flag == 1]

```
Rise_edge
send(rise_edge_at_odd)
```

TYPE	NAME	VALUE	PORT
	current_tick		1
	active_flag	0	
	rise_edge_at_odd		1

Figure 3: Stateflow chart with a rising edge output event every time it receives an odd local tick.

4 Matlab processing

The local time ticks counter was monitored and logged using HANTune. The logging frequency was 100 Hz (HANTune maximum), which is two orders of magnitude slower than the program counter. The counter ticks were measured for over 40 s and the ticks per second were analysed with the following code.

```
% Loop along the whole data vector
for i = 1:m-1
    % Time frequency
    freq_HAN(i) = HANTune_time(i+1) - HANTune_time(i);
    % Ticks between samples
    dt_ticks(i) = ticks(i+1) - ticks(i);
end
% Ticks per second
ticks_time_HAN = dt_ticks./freq_HAN;
```

The results obtained when getting the mean and standard deviation of the $ticks\_time\_HAN$, for an expected granularity of 100 μs, are $gr = 110 \pm 20$ μs in the Olymexino board and $gr = 100 \pm 0$ μs in the E407 board.

If the first input of the *Scheduler Compare Event* block is set ten times smaller, the granularity of the local clock is reduced in one order of magnitude. However, it has been found that the boards program crashes when the granularity set is 10 μs.