



**Hochschule Darmstadt**

**Fachbereich Elektrotechnik und Informationstechnik**

**International Master of Science of Electrical  
Engineering and Information Technology**

**System Driven Hardware Design**

**Software Development Assignment**

**14.06.2020**

**Lecturers: Prof. Dr.-Ing. BANNWARTH, Stephan**

**M. Sc. HUBRICH, Michael**

**AGUILAR AGUILA ISAIAS, Oscar      Matr. 766468**

## Table of Contents

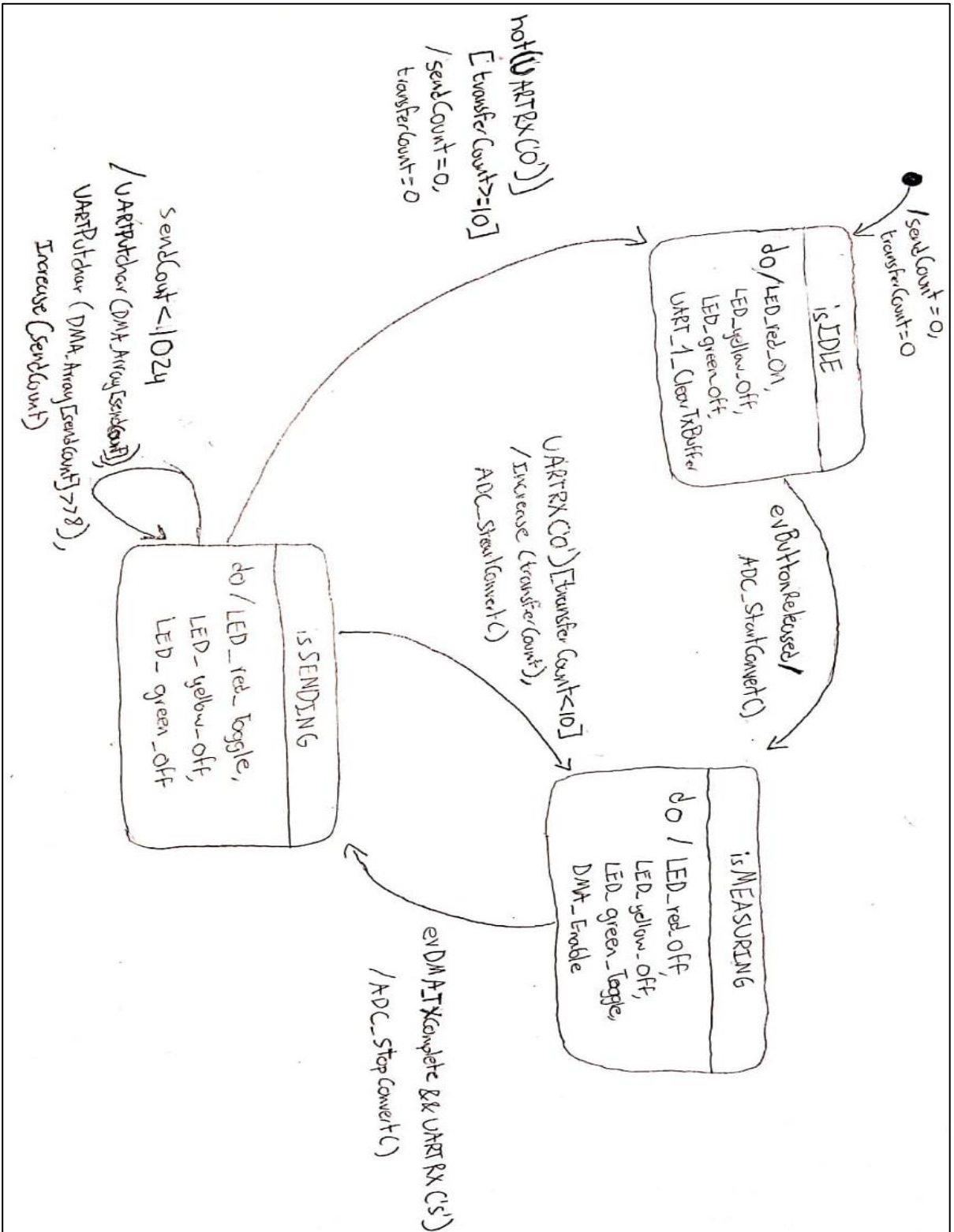
Table of Contents .....	1
Introduction.....	2
Finite State Machine .....	3
Test Bench .....	4
Ideas for Further Improvements.....	6

## Introduction

In the following document, the written part of my deliverable for the Software Development Assignment of the System Driven Hardware Design lecture will be presented.

- This document contains the following points:
- My design for the Finite State Machine
- A picture of my breadboard setup (my test bench)
- Ideas for further improvements

# Finite State Machine



## Test Bench

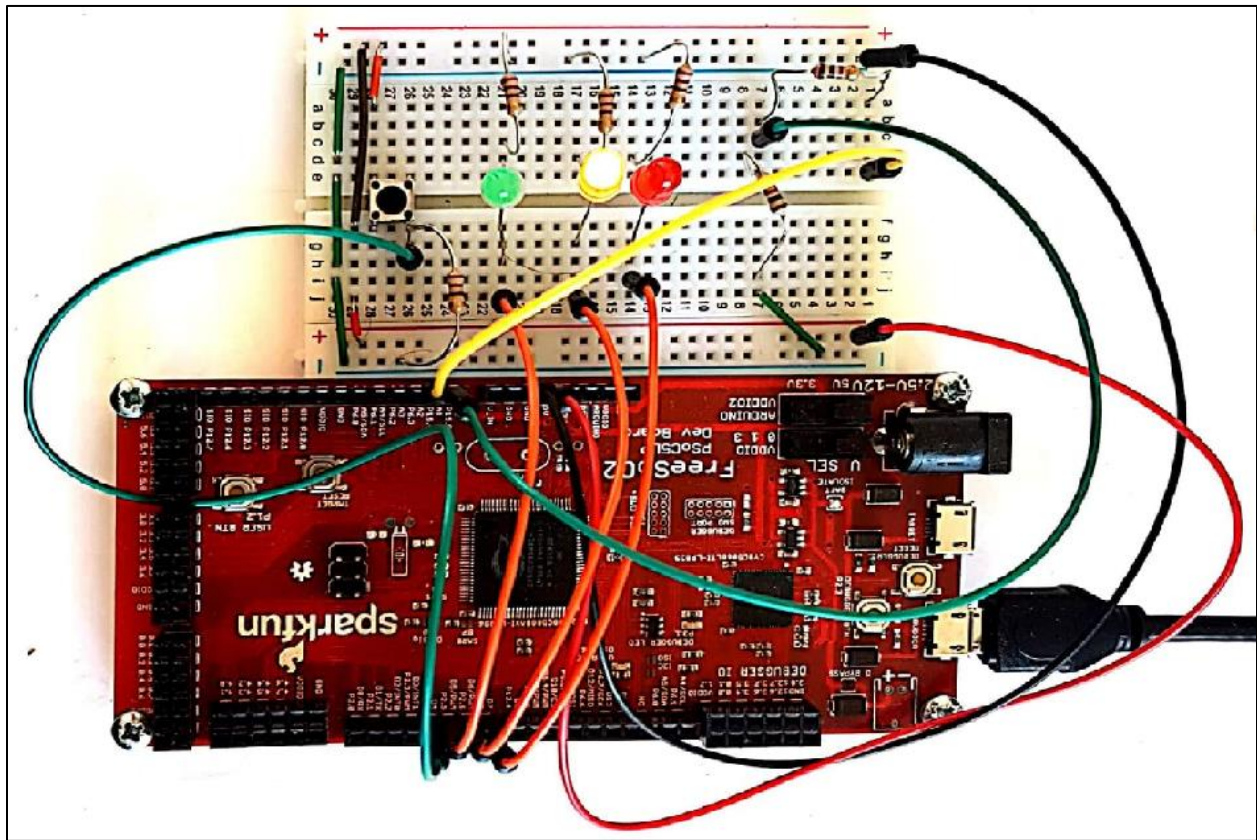


Figure 1 Breadboard Connections/Test Bench

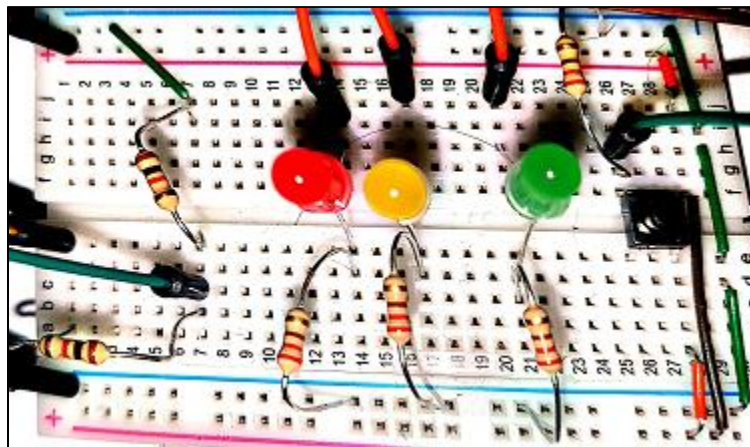


Figure 2 Closer look: Voltage Divider of 1kOhm









	Name	Port	Pin	Lock
	ADC_IN	P15[5]	94	<input checked="" type="checkbox"/>
	DAC_OUT	P15[4]	93	<input checked="" type="checkbox"/>
	LED_green	P2[5]	1	<input checked="" type="checkbox"/>
	LED_red	P2[7]	3	<input checked="" type="checkbox"/>
	LED_yellow	P2[6]	2	<input checked="" type="checkbox"/>
	PushB	P2[4]	99	<input checked="" type="checkbox"/>
	UART_1_RX	P2[0]	95	<input checked="" type="checkbox"/>
	UART_1_TX	P2[1]	96	<input checked="" type="checkbox"/>

Figure 3 Pin Configuration

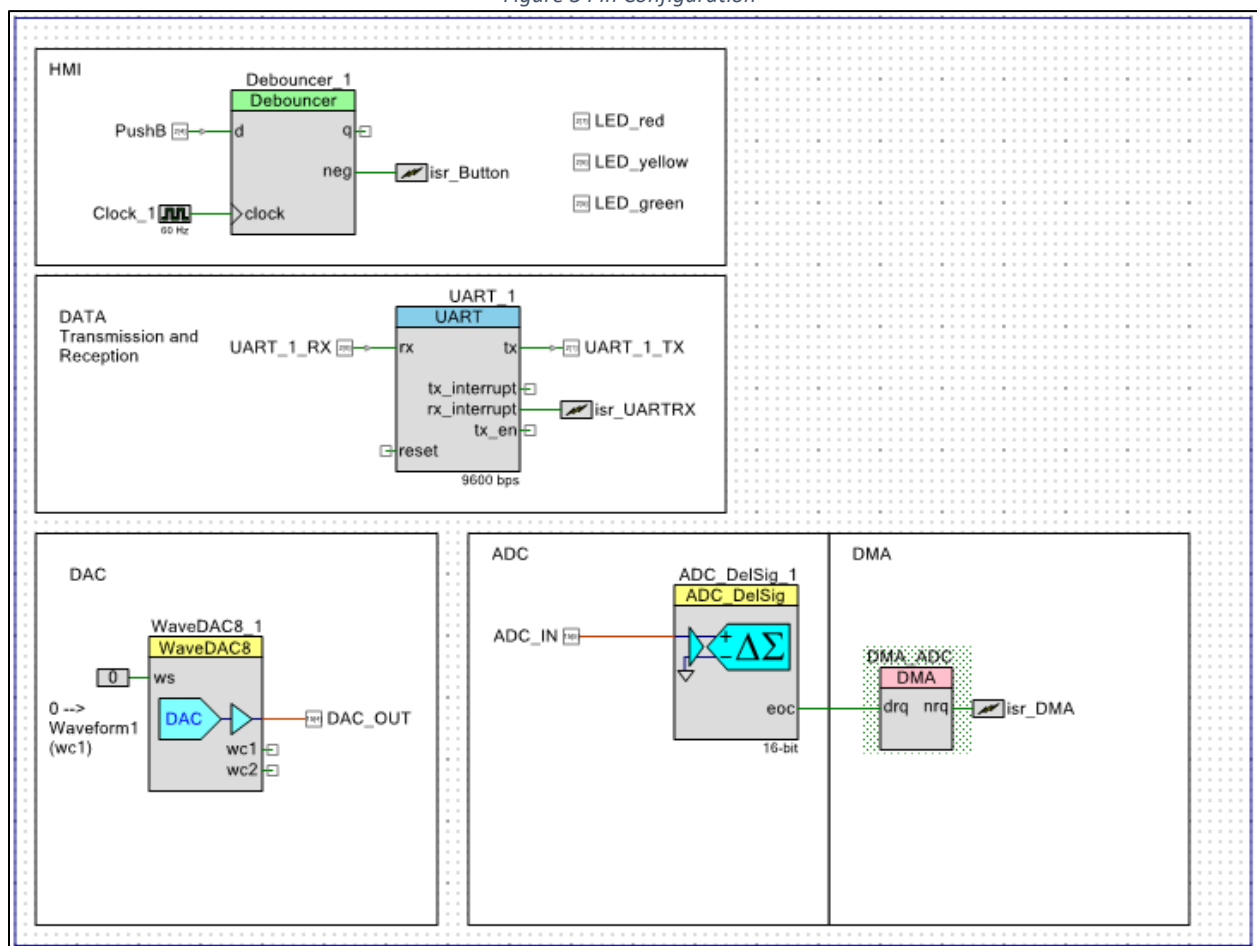


Figure 4 PSOC Top Design

## Ideas for Further Improvements

There are some ideas regarding further improvements for this system:

**1) The implementation of a real-time operating system for multithreading and more efficient handling of interrupts.**

This was not done in this iteration of the project, because for the current version of the software, with the current requirements, a bare-metal implementation is enough to comply to all requirements in a safe and efficient way. Hence, no operating system is explicitly needed.

However, if the system were to grow, and more use cases were to be implemented, the usage of said RTOS would greatly benefit said system.

**2) More extensive implementation of error handling.**

Insofar, the current error handling takes care of several use cases. For example the button being pressed in a state in which it shouldn't, as shown in the next code snippet.

```
buttonFlagClear(); // clear button to ignore any button pushing prior to returning to idle--> guard against 'impatient' users
*currentState=IS_IDLE;
```

However, there are some deeper errors in which no error handling was yet implemented.

Namely, a ticking timer paired with a watchdog would be a great implementation, to make sure that the code isn't stuck in any state or a condition that times an expected reception of an 'o' character after certain time, in order to achieve certain robustness against incomplete transmissions.

However, all of these makes more sense in a multithreading system, in order to make sure that an interrupt correctly handles timer ticks and that the watchdog will be kept alive in the lowest-priority task, for example.

**3) A transition modification.**

According to the current requirement, "Press the external Push Button to start measuring", as soon as the button is pressed, measuring should start. I have a suggestion, that measuring should start after the s is received AND the button has been pressed.

Although this slightly changes the currently desired algorithm, one might argue that if the system which will receive the data is not yet ready to receive anything, it will not send an 's' to the FreeSoc2. If a button is pressed and an s is not yet received, the FreeSoc2 will then stay in a measuring state without really being required anything, causing some irrelevant data measurement, hence noise.

This proposal could be easily taken into account as depicted in the following

code snippet:

```
//if (UARTRX_s_Flag) //would avoid unnecessary recording of  
//{  
CyDmaChEnable(DMA_ADC_Chan, 1); //No need to disable.... it  
ADC_De1Sig_1_IsEndConversion(ADC_De1Sig_1_WAIT_FOR_RESULT);  
//}
```