Running a Simple DEVS-based Flight Control System on a Target Quadcopter

Ali Salah Eddin
alisalaheddin@cmail.carleton.ca

Abstract

Modeling and Simulation techniques are important in the development of complex projects. They help simplify the system development and analyze the system's behavior before extensively developing and testing it. The DEVS formalism is a popular modeling and simulation technique. This report, presents a DEVS-based flight control system that runs on a DEVS-based bare-metal kernel, ECDBoost. It also shows the integration of such a system on a target quadcopter, Crazyflie 2.0 and the feasibility of flying the quadcopter using the DEVS-based flight system. Finally, a simulation of the complete solution is run and an execution on target shows successful results.

1. INTRODUCTION

Modeling and Simulation (M&S) methods are a great way to test and verify systems during their development phase. They help understand the system's behaviour early on, thus saving on development and prototyping costs before testing and reiterating the production cycle until the final system design is up to standard. M&S methods are advantageous when they follow specific formalisms. This way they are systematic and this makes it easier to verify and validate the models designed using their formalisms. As such, M&S methods can enable large reliable scalability of system models based on the understanding that the composing components of the scaled design were all validated and that their integration wouldn't hinder those validations nor make it harder to verify and validate the larger system design as a whole. Building on these base points, the idea of making a product running a M&S system design does not appear unordinary. After all, this would be a cost effective method of production as the system design that will go into production is the same model that was verified and validated through simulation test.

Previous work has been done to design a platform that can host simulation models and run them on target hardware. An example of such a platform is Embedded CDBoost which is a formal M&S based kernel that runs on bare-metal and executes the original simulation model on the target hardware. ECDBoost runs simulation models that are based on the Discrete EVent System specification (DEVS) [1] M&S formalism.

This project was aimed at developing a DEVS based model that simulates a simplified flight controller of a quadcopter, and running it along with ECDBoost on a commercial quadcopter: Crazyflie 2.0 (CF2). The result of this project was satisfactory and constituted a building block towards a full DEVS based flight control model design for a quadcopter.

2. BACKGROUND

2.1. Discrete EVent System specification

DEVS is a formalism developed to model and simulate dynamic systems. It follows a rigorous rule set to define state changes of the modeled systems with regards to input events or time delay triggers. The DEVS formalism also defines a guideline to decompose complex system designs and precisely hierarchize them into basic models called *atomic* and composite models called *coupled* [2]. A coupled model is composed of a group of atomic and/or coupled models with well-defined coupling connections between its components.

A DEVS atomic model is formally defined by:

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta \rangle,$$

Where:

X = $\{(p,v) \mid p \in IPorts, v \in Xp\}$ is the set of input ports and values;

Y = $\{(p,v) \mid p \in OPorts, v \in Yp\}$ is the set of output ports and values;

S: is the set of sequential states;

$\boldsymbol{\delta_{int}}$: $S \rightarrow S$ is the internal state transition function;

$\boldsymbol{\delta_{ext}}$: $Q \times X \rightarrow S$ is the external state transition function, where:

Q = $\{(s,e) \mid s \in S, 0 < \in < ta(s)\}$ is the total state set, e is the time elapsed since the last state transition;

$\boldsymbol{\lambda}$: $S \rightarrow Y$ is the output function;

ta: $S \rightarrow R^{+}_{0,\infty}$ is the time advance function.

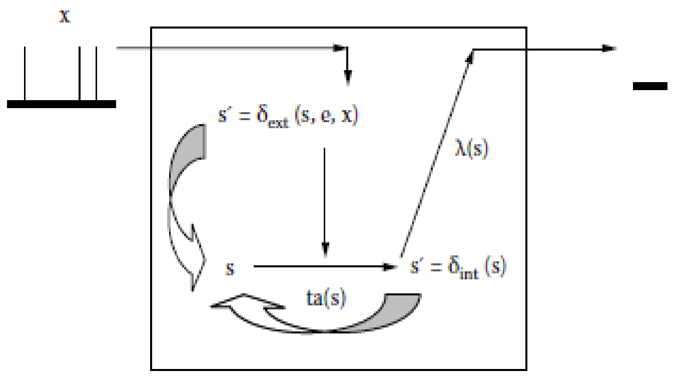

Figure 1: Informal definition of an atomic model [2]

Figure 1 describes the above mentioned specification. An incoming input event *x* triggers an external transition δ_{ext} in the atomic model. This transition is a function of the current state *s*, the input event *x* (set of ports and values) and elapsed time *e* since the last transition of the system. Based on the conditions of *x* and *e*, the external transition changes the state of the system to *s'*. Inside the system, there could be a time advance timer set to trigger when its value ta has elapsed since it was reset in the last transition. This trigger simultaneously causes an internal transition δ_{int} and output function λ, both functions of the current state *s*. The output function generates an output event *y* and the internal transition changes the current system state to a new state *s'*.

The DEVS specification of a coupled model is different from that of an atomic model as it doesn't define system state transitions or output functions but instead defines the formal port connections and their directions between the components. The following figure 2 is a simple example of a coupled model hierarchy:

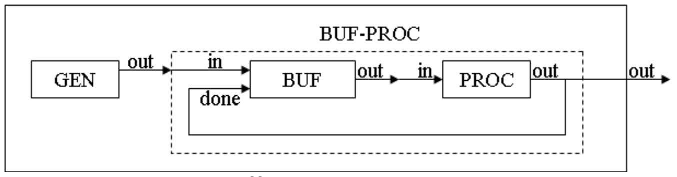

Figure 2: Generator-Buffer-Processor hierarchical DEVS model [3]

The Top model is a coupled model made of an atomic model GEN and a coupled model BUF-PROC, connected via the GEN::out →BUF-PROC::in connection. BUF-PROC is connected to the Top model via the BUF-PROC::out → Top::out connection. BUF-PROC is also composed of two atomic models BUF and PROC whose ports are interconnected as shown in the figure.

2.2. Embedded CDBoost

Embedded CDBoost (ECDBoost) [4] is an extension to the CD++ [5] tool that uses the Boost library and that was developed based on DEVS theory and its extensions [2]. It has converted the virtual time function of CD++ into a real-time function (using a time advance function tied to the real-time clock) and added hardware interaction capability to it. Model implementations are programmed in C++ code. ECDBoost is ran as a bare-metal kernel and has been tested on ARM Cortex-M microcontrollers.

3. QUADCOPTER

A quadcopter is a helicopter with four rotors. It takes off and lands vertically and can hover in mid-air due to its lift and thrust that are supplied by the propelling rotors. Quadcopters are used in many researches and applications aiming to enhance automated flight controls. The aim of this work was also to explore the feasibility of running a simple DEVS-based flight control model on a target quadcopter. The target is a largely commercially available quadcopter, Crazyflie 2.0, that comes with pre-built firmware which is open source.

This section explores the DEVS-based design of the flight control model, gives an insight on the architecture and software system organization of the CF2 and finally explains how the DEVS-based flight control model was integrated into the CF2.

3.1. DEVS-based Flight Controller Overview

The DEVS flight control is made of a single atomic model called *flightDEVS*. This model has a very basic behaviour where it receives an input event holding a thrust power level for the rotors, that it stores and sends out later as an output event. The embedded real-time scope in which *flightDEVS* is simulated, includes input/output port objects that trigger input events to *flightDEVS* and react to output events triggered by *flightDEVS*. In this flight control system, there are one input port, Command Port, and four output ports, MotorPort[1-4]. Figure 3 below illustrates the relation between the ports and *flightDEVS*.

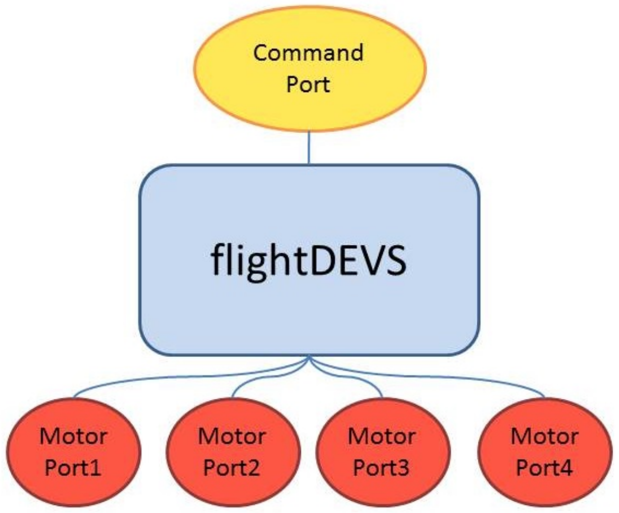

Figure 3: DEVS-based flight control system diagram

The connections between the port objects and the atomic model are based on a design implementation that is understood by ECDBoost, based on the DEVS formalism. As such, Command Port is connected to the External Input Connection of *flightDEVS* and MotorPort[1-4] are connected to the External Output Connection of *flightDEVS*.

The DEVS formal specification of the *flightDEVS* model is as follows:

$$M = <X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ta>,$$

Where
X: (Command Port, {thrust}) Connected to *flightDEVS* input port.
S: "IDLE", "PREP_RX", "WAIT_DATA", "TX_DATA", "PREP_STOP".
Y: (MotorPort1, {thrust}) Connected to the *flightDEVS* output port;
(MotorPort2, {thrust}) Connected to the *flightDEVS* output port;
(MotorPort3, {thrust}) Connected to the *flightDEVS* output port;
(MotorPort4, {thrust}) Connected to the *flightDEVS* output port;
$\boldsymbol{\delta_{ext}}$: Receives inputs from the input port and initiates appropriate state transitions.
$\boldsymbol{\delta_{int}}$: defines state changes based on current state.
λ: based on the input value and the current state sends the output signals to the output ports.
ta: real-time advance function.

The behaviour of *flightDEVS* is described in its external (δ_{ext}) and internal (δ_{int}) transfer and output (λ) functions. When an input event arrives, the model's external function is called. It reads the payload carried by the input event, containing thrust power level, stores it and sets the internal time advance to a very short timeout value, because this model wants to be almost instantly reactive. When the timeout is triggered, the internal transfer function is called. In *flightDEVS*, the internal transfer function is passive and just resets the internal time advance. Also, the output function is triggered and it outputs an output event that contains the thrust power levels designated for each rotor of the quadcopter. *flightDEVS* is a single state model, as such its external and internal transfer functions didn't include any state changes.

The complete system (combination of ports and atomic model) functions in the following manner:

- ECDBoost performs periodic polling on the Command Port to detect new input events;
- If there are new inputs, a new input event is triggered;
- *flightDEVS* receives the input event from ECDBoost and processes it;
- If no new events arrive before the internal time advance timeout, *flightDEVS* outputs an output event to ECDBoost;
- ECDBoost transfers the event to MotorPort[1-4] which will individually use the payload contained in the event.

3.2. Crazyflie 2.0 System Architecture

The system architecture of the CF2 uses two micro-controller units (MCU). The first MCU (nRF51822) is responsible of handling radio communication and power management, and the second MCU (STM32F405) is responsible of handling the CF2's flight control, sensor reading and motor control. The two MCUs are independent and communicate via a UART link. Figure 4 below shows the architecture of the CF2 system:

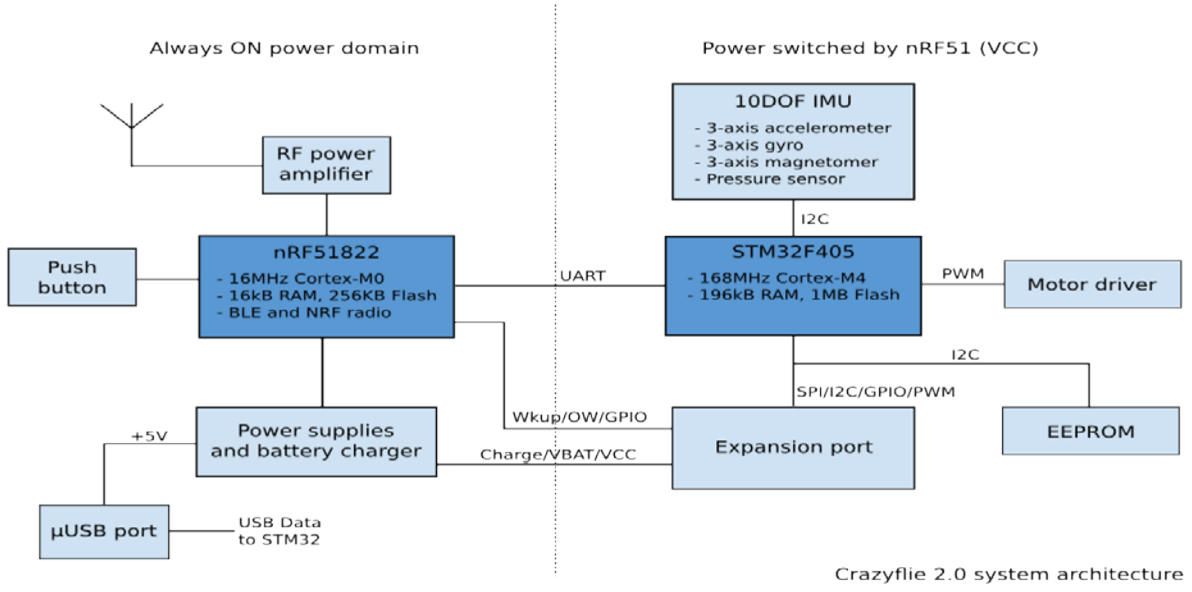

Figure 4: Crazyflie 2.0 System Architecture [6]

The independence of the two MCUs from one another made it easy to focus solely on the system running on the STM32F405. That system is made of the flight control application which gets scheduled, along with other tasks, by a real-time operating system (RTOS) called FreeRTOS and which makes calls to drivers to read data from sensors and send commands to rotors. The firmware of this system is reprogrammable without affecting the system running on the nRF5188 MCU. Thus, developers can modify the CF2's flight control system or create other modules that can be run on the quadcopter. The firmware running on the STM32F405 is comparted into different packages, such as "drivers" for the hardware drivers, "FreeRTOS" for the RTOS, "modules" for the components of the flight controller.

3.3. System Integration

In order to run *flightDEVS* onto the CF2 quadcopter, the DEVS-based system was integrated into the CF2's firmware solution. Firstly, the DEVS-based system code was copied to the CF2's firmware. Then, the CF2's flight control system initialization was modified to bypass calls that start-up the RTOS and schedule the flight control and any other application as tasks. The system initialization was re-written to initialize the DEVS-based system simulation on ECDBoost instead. At this point, *flightDEVS* has replaced the CF2 flight control system, and ECDBoost has replaced the functionality of FreeRTOS in running the DEVS simulation without any RTOS scheduling intervention. Figure 5 shows the CF2's system organization change following the integration.

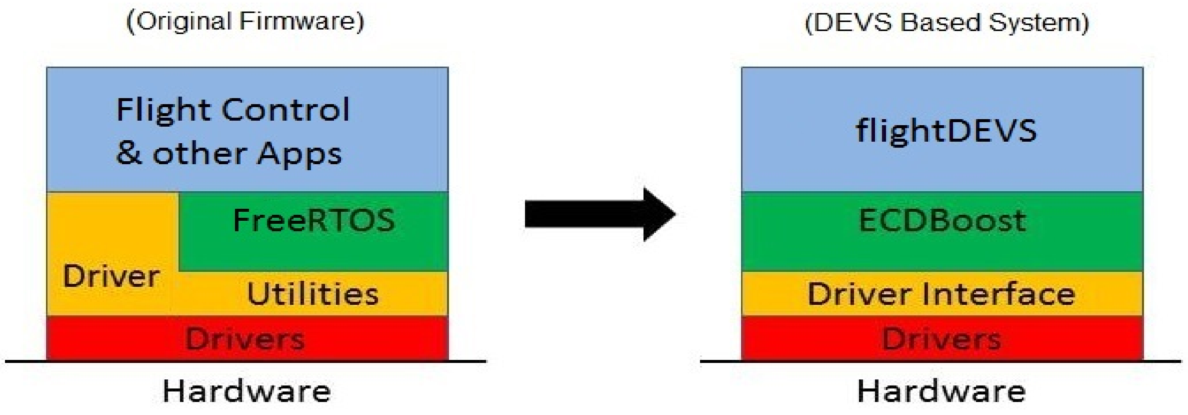

Figure 5: Crazyflie firmware change after system integration

A Driver Interface has also been added. It serves as a bridge between the DEVS-based system's port objects and the hardware drivers from/to which they would get/send data. Hence, MotorPort[1-4] make function calls to the Driver Interface in order to activate the CF2's rotors at the requested thrust power level. For the sake of simplicity, the scope of this project didn't explore reading input commands from the CF2's joystick into the Command Port. The input thrust levels were provided by an array of values that were fed into the Command Port.

As part of making the new flight control solution real-time, the time reading functionality invoked from ECDBoost was planned to be connected to a hardware timer on the STM32F405 MCU. This part did not work well and faced some troubleshooting difficulties. As such, for experimentation purposes, the simulation timing functionality was kept in the new flight control solution, to simulate time advancement.

3.4. Simulation Results

The DEVS-based system made of *flightDEVS* running on ECDBoost was simulated.The following table 1 shows the results of the simulation:

Table 1: Results of *flightDEVS* model simulation

Input	Output
1. [TIME 00:00:00:203:000] [INPUT 10]	-
2.	[TIME 00:00:00:213:000] [MOTOR 3] [THRUST 10] [TIME 00:00:00:214:000] [MOTOR 2] [THRUST 10] [TIME 00:00:00:215:000] [MOTOR 1] [THRUST 10] [TIME 00:00:00:216:000] [MOTOR 0] [THRUST 10]
3. [TIME 00:00:00:406:000] [INPUT 20]	-
4.	[TIME 00:00:00:416:000] [MOTOR 3] [THRUST 20] [TIME 00:00:00:417:000] [MOTOR 2] [THRUST 20] [TIME 00:00:00:418:000] [MOTOR 1] [THRUST 20] [TIME 00:00:00:419:000] [MOTOR 0] [THRUST 20]
5. [TIME 00:00:00:609:000] [INPUT 30]	-
6.	[TIME 00:00:00:619:000] [MOTOR 3] [THRUST 30] [TIME 00:00:00:620:000] [MOTOR 2] [THRUST 30] [TIME 00:00:00:621:000] [MOTOR 1] [THRUST 30] [TIME 00:00:00:622:000] [MOTOR 0] [THRUST 30]

3.5. Execution Result

The new CF2 full flight system solution was run on the quadcopter following the same scenario of events as that of the simulation, except with larger thrust values.

The result was satisfactory, where the CF2 momentarily took flight. A video of the quadcopter flying is available online [7].

4. CONCLUSION

This report demonstrated a DEVS-based flight control system for a Quadcopter. It showed how the system was designed. The system reads in input values of Thrust power from an input port (Command Port). An event is triggered and received by the flight control model, flightDEVS. The flightDEVS processes the incoming event that contains desired thrust values and generates an event that will be caught by the Motor Ports which will use the thrust values sent with the event. The DEVS-based system was then integrated into the Crazyflie 2.0's firmware by modifying the quadcopter's firmware to not use the flight control modules nor the RTOS that were running on it and run the flightDEVS model on ECDBoost instead. The Motor Ports were connected to a driver interface in order to output the thrust values received from flightDEVS the Crazyflie 2.0's rotors and make it fly. Simulation and experimental testing of the full solution were conducted and the results proved to be satisfactory. Future work would look at augmenting the flightDEVS model.

5. REFERENCES

[1] - Zeigler, B. P. 1976. *Theory of modeling and simulation.* New York: Wiley-Interscience.
[2] – Wainer, G. “Discrete-Event Modeling and Simulation, a practitioner’s approach”.
[3] – Moallemi, M; Wainer, G; Awad A.; Tall, D. “Application of RT-DEVS in Military.
[4] - NIYONKURU, D.; WAINER, G. “Towards a DEVS-based Operating System”.
[5] - Wainer, G. "CD++: a toolkit to define discrete-event models". Software, Practice and Experience. Wiley. Vol. 32, No.3. pp. 1261-1306. November 2002
[6] - https://www.bitcraze.io/2014/07/crazyflie-2-0-system-architecture/
[7] – Youtube ARS-LAB