Technical Report for the Implementation of the Restart-based Framework

Chien-Ying Chen
University of Illinois at Urbana-Champaign
cchen140@illinois.edu

1 Introduction

This article is intended to document the implementation of the restart-based framework on a realistic platform. The implementation mainly consists of two parts: *(i)* a main controller platform and *(ii)* a Root of Trust (RoT) module. The main controller runs tasks for user's application. Functions and tasks for the secure execution interval (SEI) are also handled on the same platform. The RoT module acts as a trustworthy hardware that can provide the security guarantees offered by the restart-based framework. An high level overview diagram that shows the connection between the main controller and the RoT module is given in Figure 1.

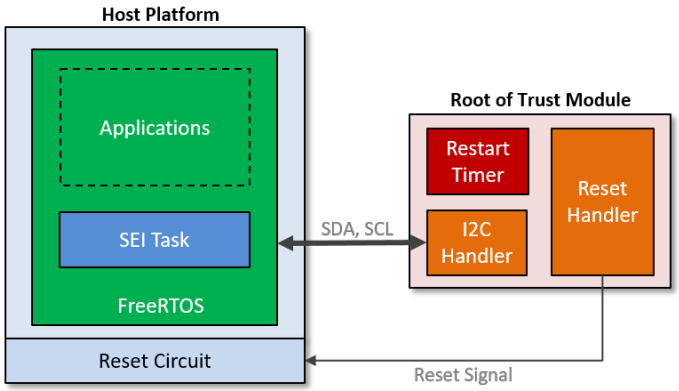

Figure 1: A high level overview of the connection between the main controller platform and the RoT module.

In the following content, §2 and §3 provide the details of the implementation for RoT module and the main controller respectively. §4 talks about the wiring between the RoT module and the main controller. §6 describes a demonstrative application that we have implemented to demonstrate the use of the proposed restart-based protection.

2 Root of Trust (RoT) Module

The main function of the RoT is to act as a trustworthy hardware that generates a configurable reset signal and enforces the secure execution interval. What follows below introduces the details of the RoT module we have implemented.

2.1 Hardware (MSP-EXP430G2)

A MSP-EXP430G2 board is used to realize the RoT module. The MSP-EXP430G2 board is a development board specifically designed for MSP430 series micro-controllers developed by *Texas Instruments* (TI). It is equipped with a MSP430G2452 [5], an instance of the MSP430 micro-controllers. The board is operated under 3.3V power which drives the MSP430G2452 micro-controller as well as its inputs/outputs. Figure 2 displays the board and its pinout.

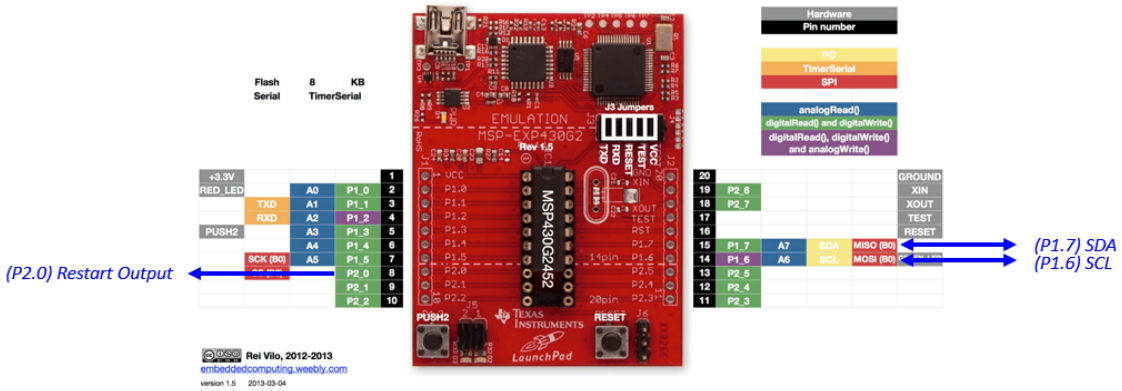

Figure 2: MSP-EXP430G2 board pinout.

The *Timer A* in the micro-controller is used to build the restart timer. It is a 16-bit timer that is configured to run at a clock rate of 1MHz (*i.e.,* 1us per timer count) with using its integrated digitally controlled oscillator (DCO). That is, the minimum restart time supported by the RoT module is $1us$. We use a counter inside the interrupt handler of *Timer A* to extend the timer with an adjustable factor, so the restart timer can count up to the range based on the application's needs.

2.2 I^2C Interface

To make the timer configurable to the host (the platform that uses the RoT module), a communication interface is provided. We choose I^2C as the interface in our design since I^2C is commonly adopted in many passive components (*e.g.,* sensors, memory modules). On the MSP430G2452 micro-controller, pin *P1.7* and *P1.6* can be configured as *SDA* and *SCL* for I^2C respectively, as shown in Figure 2.

2.3 RoT Control Flow

The RoT timer module listens incoming commands from the I^2C interface connected to the host. The RoT timer module maintains two timers: *(i)* a secure execution interval timer and *(ii)* a restart timer.

The restart timer enforces the duration between each restart of the system. When the restart timer is due, a reset signal is issued and is intended to trigger a reset for the host system. The secure execution interval timer is activated when a reset signal is issued (*i.e.,* right after the restart timer is due). The secure execution interval is designed for the host system to run security checks to determine the system's security level and to configure the next restart time. Once the system exits the secure execution interval (either the secure execution interval timer is due or the host actively tells the RoT module to exit the interval), the secure execution interval timer is terminated and the restart timer is activated. The control flow that runs in the RoT module is shown in Figure 3 and Figure 4.

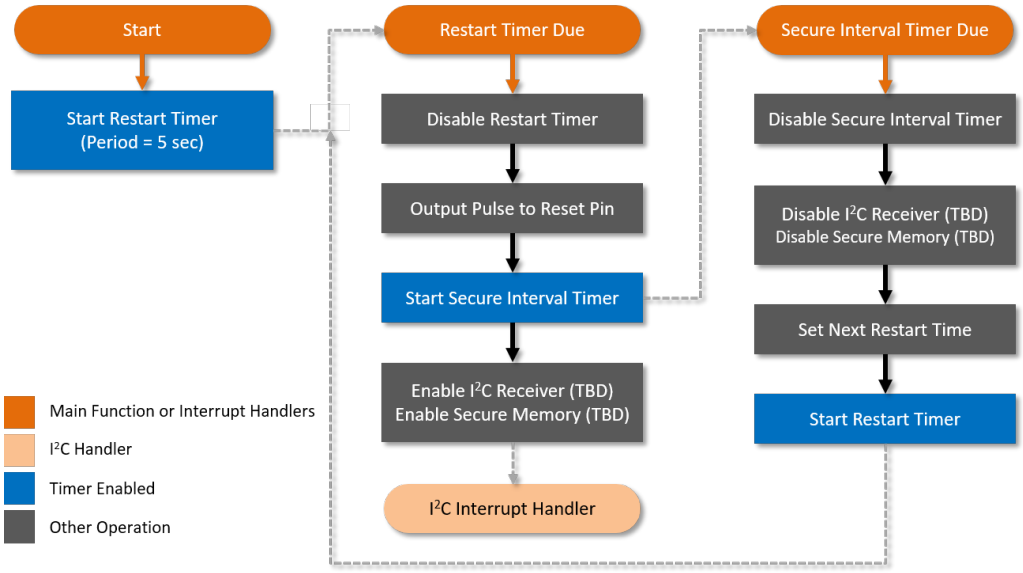

Figure 3: Flow chart of the RoT timer (1^{st} part).

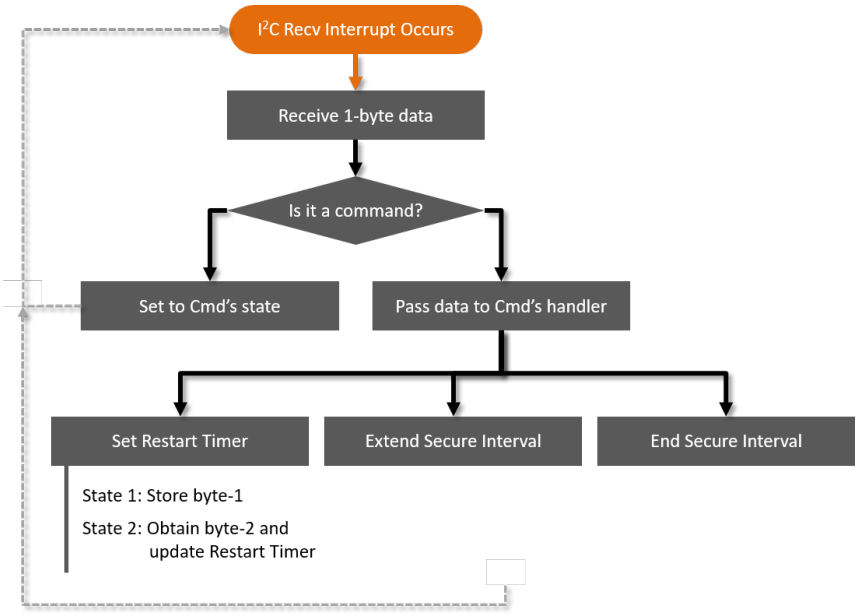

Figure 4: Flow chart of the RoT timer (2^{nd} part).

3 Main Controller Platform

3.1 Zedboard

A Zedboard [2] is used as an application platform in this implementation. It includes a XC7Z020 SoC, 512MB DDR3 memory and an on-board 256MB QSPI Flash. The XC7Z020 SoC consists of a processing system (PS) with dual ARM Cortex-A9 cores and a 7-series programmable logic (PL). The processing system runs at 667MHz. Note that only one ARM core is used in our implementation. The programmable logic is programmed to connect Zedboard's basic functions (*i.e.,* LEDs, switches, buttons). Figure 6 shows the circuit design in the programmable logic.

3.2 Boot Options

There are several boot options available on Zedboard: *(i)* boot via JTAG, *(ii)* boot via on-board flash and *(iii)* boot via external SD card. The boot mode can be configured by adjusting the

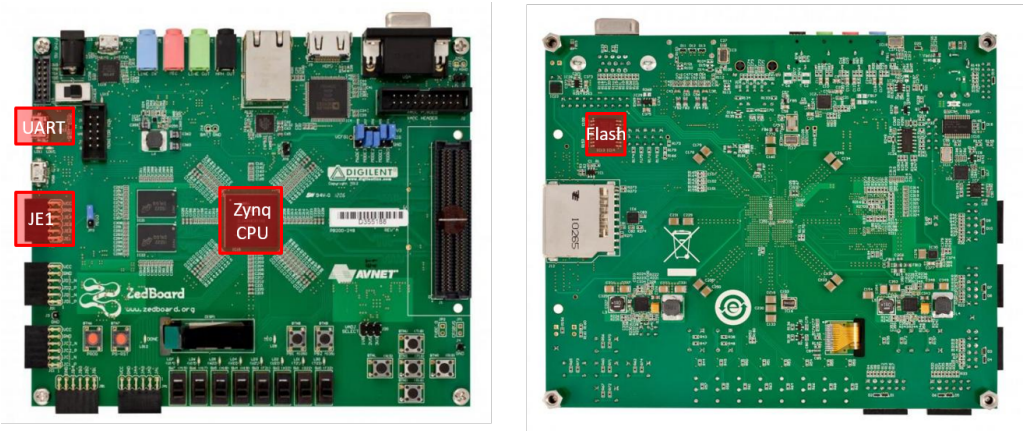

Figure 5: Top and bottom view of Zedboard. The *JE1* header is used to connect RoT module via I^2C interface.

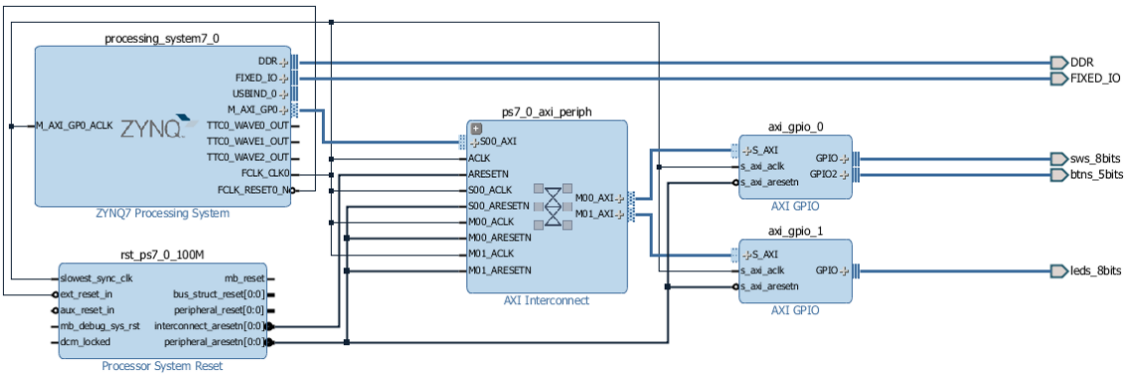

Figure 6: Programmable logic design in Zynq-7000 CPU on Zedboard.

positions of the jumpers on Zedboard as shown in Figure 7.

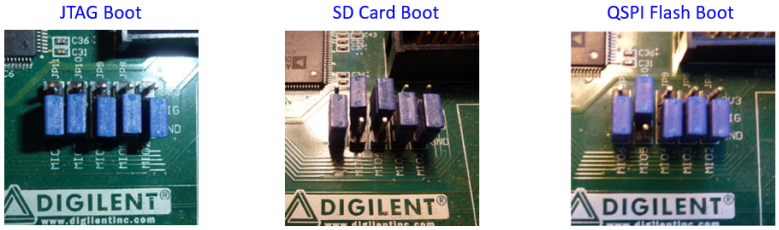

Figure 7: Boot options on Zedboard.

For an efficiency reason, we choose to boot the system via on-board flash. This option yields the fastest boot time among the three. A boot time measurement is given in §5.

3.3 SEI Task and Operating System

The main controller runs *FreeRTOS* [1], a preemptive real-time operating system, for both SEI tasks and application tasks. During SEI, only SEI tasks are created and executed when *FreeRTOS* starts. The SEI tasks set the next restart time to RoT module via I^2C interface. When SEI ends, the SEI tasks are terminated and the application tasks are created.

4 Interfacing Host Platform and RoT Timer

4.1 I^2C Interface

As we have introduced earlier, RoT timer provides I^2C for a host to configure its timer. In this case, Zedboard acts as a I^2C master while RoT timer acts as a I^2C slave. On Zedboard, ARM CPU's *MIO14* and *MIO15* are configured as *SCL* and *SDA* for I^2C respectively. As shown in Figure 8, ARM CPU's *MIO14* and *MIO15* are available on Zedboard as *JE9* and *JE10*. Therefore, Zedboard's *JE9* is connected to MSP430G2452's *P1.6* and Zedboard's *JE10* is connected to MSP430G2452's *P1.7*.

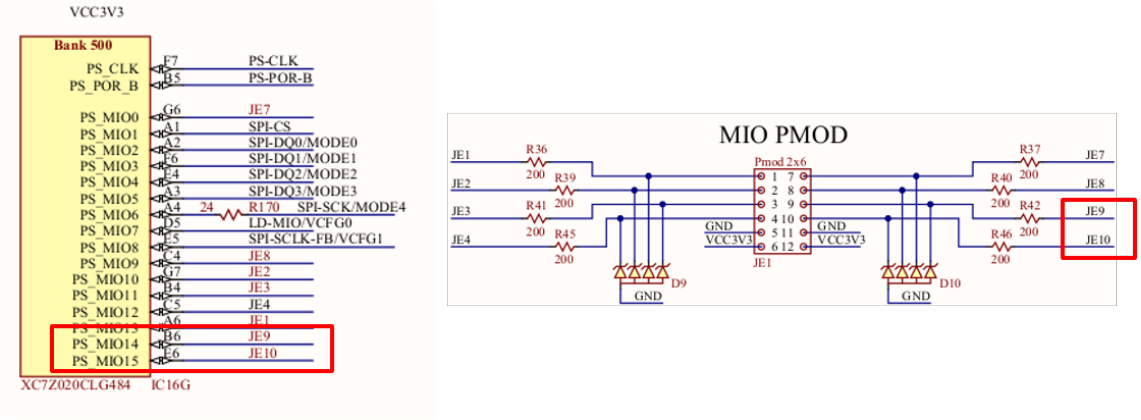

Figure 8: *MIO14* and *MIO15* pin locations on Zedboard.

To make I^2C interface work properly, a pull-up resistor is required on both *SDA* and *SCL* pins. Here, we use internal 3.3V pull-ups provided by the ARM core on *MIO14* and *MIO15*. Figure 9 shows the pull-up configurations for *MIO14* and *MIO15* in *Vivado*.

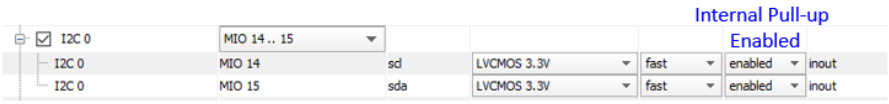

Figure 9: Configuration details for *MIO14* and *MIO15* in *Vivado*. The internal pull-ups for *MIO14* and *MIO15* are enabled.

4.2 Reset Signal

The main issue we have when connecting the reset output from the RoT module to Zedboard is the inconsistency of the I/O voltage. On Zedboard, the reset input uses 1.8V which is lower than RoT module's output voltage 3.3V. To avoid any damage caused by the voltage surcharge, a level shifter that converts 3.3V signal from RoT module to 1.8V signal for Zedboard is necessary. We use a BSS138, a N-channel logic level enhancement mode field effect transistor, to build a 3.3V to 1.8V level shifter (this is a bidirectional level shifter). Figure 10 shows the circuit design of the level shifter and Figure 11 shows the captured waveform of the reset signals on both 1.8V and 3.3V sides.

5 System Restart

The restart procedure activates by sending out a low state signal from RoT to Zedbaord. Upon restart, several things happen:

1. RoT triggers the reset pin. A complete reset signal include pulling the reset signal from high to low and then release it from low to high.
2. Zedboard is reset. Both PS and PL in the XC7Z020 SoC are reset. Note that, upon reset, the image for PL is cleared.

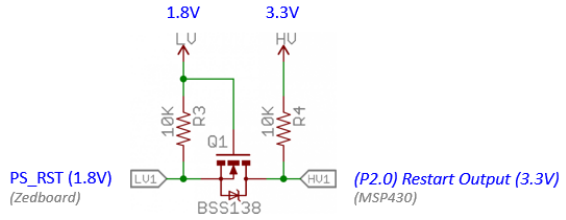

Figure 10: Converting 3.3V output signal from RoT module to 1.8V signal for Zedboard's reset input.

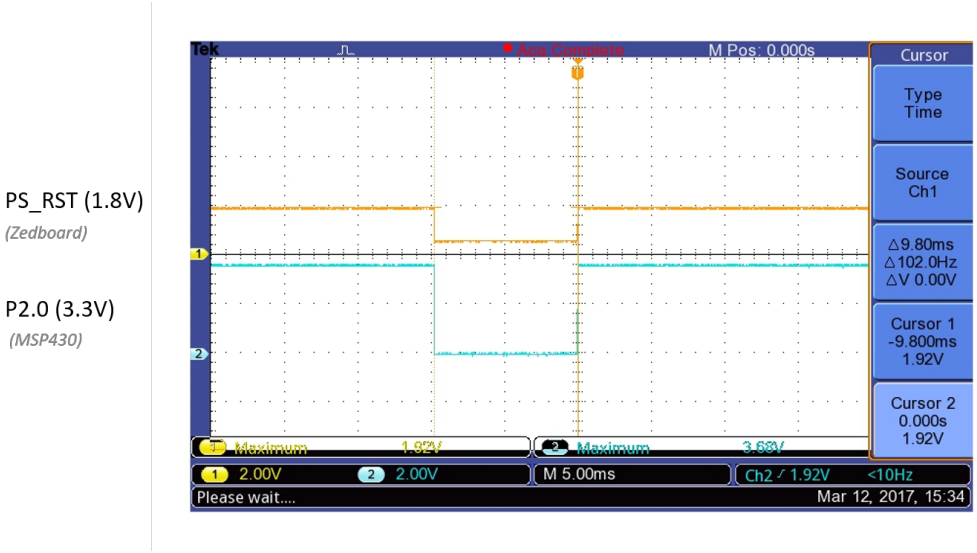

Figure 11: Waveform of the reset signals on RoT side and Zedboard side.

3. XC7Z020 loads a bootloader from the on-board Flash via QSPI to PS (it is configurable as described in §3.2).

4. The bootloader then loads an image from the on-board Flash to program PL. Note that configuring PL is necessary for PS to run correctly for following applications.

5. Once PL is ready, the bootloader loads the application image from the on-board Flash. The control of PS is handed to the loaded image once the loading is done.

From an experiment on the implemented system, the restart time (booting from the on-board QSPI Flash) measures 390ms as shown in Figure 12. As a side note, the restart time for booting from an external SD card (mentioned in §3.2) measures 680ms.

6 Demonstrative Application

In this section, we describe the use of restart-based framework under a more realistic configuration. Note that we leave the detail of the methodology in the main paper. We only focus on the implementation in this report.

6.1 3DOF Helicopter

3DOF helicopter (displayed in figure 13) is a simplified helicopter model, ideally suited to test intermediate to advanced control concepts and theories relevant to real-world applications of flight dynamics and control in the tandem rotor helicopters, or any device with similar dynamics [3]. It is equipped with two motors that can generate force in the upward and downward direction, according to the given actuation voltage. It also has three sensors to measure elevation, pitch and travel angle as shown in Figure 13. We use the linear model of this system obtained from the

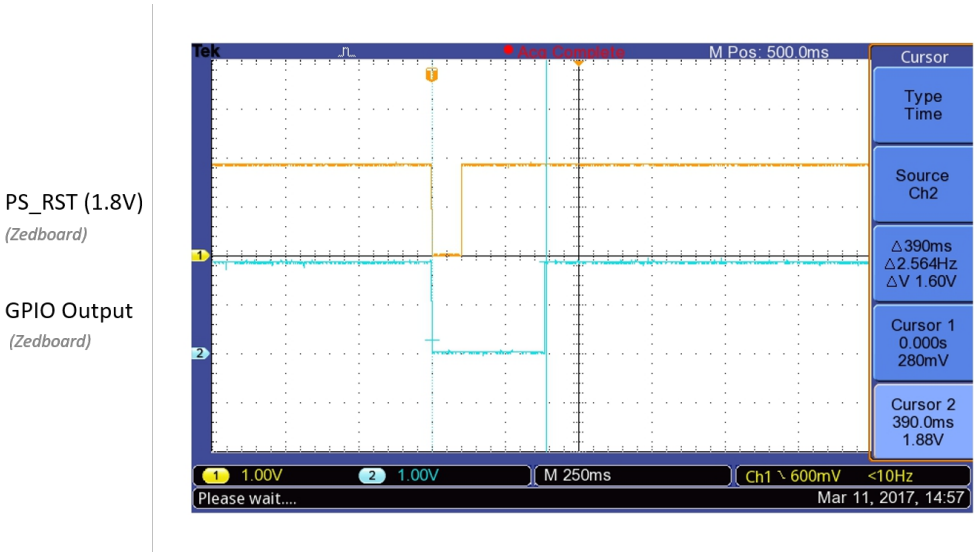

Figure 12: Measurement of the reboot time via on-board flash. It measures 390ms, from the triggering of the reset signal to the first executed instruction.

manufacturer manual [3] for constructing the safety controller and calculating the reachable set in runtime.

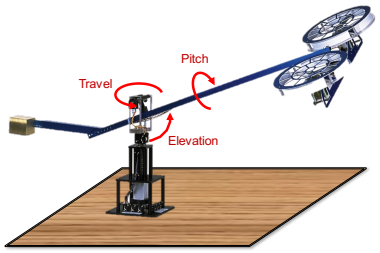

Figure 13: 3 Degree of freedom (3DOF) helicopter.

6.2 Interfacing 3DOF Helicopter and Zedboard

The main controller unit interfaces with the 3DOF helicopter through a PCIe-based *Q8 High-Performance H.I.L. Control and data acquisition unit* [4] and an intermediate Linux-based PC. The PC communicates with the ZedBoard through the serial port. At every control cycle, a task on the controller communicates with the PC to receive the sensor readings (elevation, pitch, and travel angles) and send the motors' voltages. The PC uses a custom Linux driver to send the voltages to the 3DOF helicopter motors and reads the sensor values.

6.3 Main Controller Design

6.3.1 SEI Tasks

Immediately after the reboot, `SafetyController` and `FindRestartTime` tasks are created and executed when the *FreeRTOS* starts. `SafetyController` is a periodic task with a period of $20ms$. It implements a simple 3DOF flight control function that keeps the flight in a safe state during SEI. And, `FindRestartTime` is a single task that returns before the end of SEI. Length of the SEI is set to $30ms$. `FindRestartTime` task is executed in a loop, and it only breaks out when a positive restart time is found. At this point, the restart time is sent to the RoT module via I^2C interface, `SafetyController` and `FindRestartTime` tasks are terminated and the main control application tasks are created.

6.3.2 3DOF Tasks

A periodic task is used to control 3DOF. The task implements a `ComplexController` with PID algorithm that controls 3DOF. It is similar to the controller used in `SafetyController` except that `ComplexController` controls 3DOF to make it move toward the configured set points.

References

[1] FreeRTOS. `http://www.freertos.org/`.

[2] Xilinx Zedboard. `http://zedboard.org/`.

[3] Quanser inc., 3 dof helicopter, 2016. `http://www.quanser.com/products/3dof_helicopter`, accessed: September 2016.

[4] Quanser inc., q8 data acquisition board, 2016. `http://www.quanser.com/products/q8`, accessed: September 2016.

[5] T. Instruments. Msp430x2xx family user's guide. 2013.