CMPE 490/450 Project:

Fail-Safe Module for Autonomous Airplane

Lai Nguyen
lain@ualberta.ca

Jesse Xi Chen
jesse.chen@ualberta.ca

Preferred lab day: Wednesday
Also available: Monday

Declaration of original content

"The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:"

Description of UAARGS[1]
Xilinx Data Sheets
Group 2 App notes
Schmitt Trigger for FPGA design

Table of Contents

Abstract

The University of Alberta Aerial Robotics Group (UAARG) is a group of students with various levels of experience with planes and from various fields. Their main goal is to build autonomous airplanes to enter into competitions[1]. Currently the plane only has rudimentary failsafe behaviour in case the auto-pilot fails.

The goal of this project is to:

Implement failsafe behaviour for the following two events:

1) RC receiver on the airplane experiences a loss of signal.

2) The autopilot fails.

We specify two operational modes: short and long-distance operation.

In short-distance operation mode (intended to be used if they are no urban areas close around where a crash is undesirable):

On the event that 1) happens, the servos will be locked into a known good failsafe state which will glide for a short amount of time and then attempt to perform a forced landing.

In long-distance operation mode (intended to be used when airplane is to be run outside RC of range):

If 1) happens, then switch to auto-pilot.

If 1) and 2) happens , then glide for a short time and attempt a forced landing.

Other enhancements such as receiving some level of feedback from the system to observe its status and the ability to monitor onboard systems in different manners (like sniffing serial telemetry streams) will be considered

The VHDL codes for the hardware implementation are put through IC design process and a chip is generated.

Functional Requirements

1)

All functionality shall be implemented on an FPGA

2)

Provide functionality of a Mux that switches between RC and AP mode.

3)

Provide the following functionality for short-distance operation:

1. Monitor RC line and if RC fails then go into fail-safe mode.
2. Fail-safe mode will lock servos into a glide for a timeout period.
3. After the timeout period it will attempt a forced landing.
4. If at any time during fail-safe mode RC signal is regained, then return to manual control. [1]

4) Provide a long-distance mode of operation to be used when there is little risk of plane wandering into populated area. Tentatively will have the following behaviour:

1. Monitor RC line and if RC fails then go into auto-pilot mode.
2. If auto-pilot fails go into fail-safe mode.
2. Fail-safe mode will lock servos into a glide for a timeout period.
3. After the timeout period it will attempt a forced landing.
4. If at any time during fail-safe mode RC signal is regained, then return to manual control. [2]

5) Provide a hardware (only VHDL) and software (using a VHDL soft core) implementation of the above.

6) VHDL codes for hardware version used to generate a chip.

1, 2, 3, 4 have been completed in the software implementation and 1 and 3 have been completed in the hardware implementation. Therefore the hardware implementation requirements were not met, however the software requirements were met. Additionally the hardware implementation has issues with loss of signal (see Future Work).

[1] Currently it's iffy if this is still a requirement by UAARGS. This is because they a good "fail-safe" servo-lock position cannot presently be determined. As of now we do have the functionality to lock servos in a given position, and, after a timeout, a second new position. An option we can explore is to provide a software framework that the UAARGS can use to UAARGs can use to customize a failsafe state solution. i

[2] See short-distance operation footnote on previous page

Future Work

Provide un-glitched input[2] using a median filter implemented in VHDL. Note that in the original failsafe module from UAARGS used a Schmitt trigger on inputs to clean up noisy signals as well; however, the development board (Virtex 2 Pro) currently used for prototyping of this project does not provide hysteresis on inputs.

However the Spartan XC3000, XC4000, XC5000, XC9000 families do all have hysteresis on input[3] and these are likely the family UAARGS will want to use on the actual plane. Also, it is possible to provide hysteresis externally using the development board using two resistors and an extra FPGA input pin[4] .

Optional:

- Integrate everything onto a single PCB
- Provide feedback systems to observe status and behaviour in fail-safe mode. This would be useful in the case that locking the servos in a known-safe position is not enough. e.g. if the plane is in the middle of a dive.
- Gather and use telemetric information from other plane systems
- Integrate auto-pilot onto FPGA
- Redundant system using majority voting. (maintain 3 copies of the system on the FPGA and use majority to determine output)
- Integrate 3-8 demux for servo motors[3] into FPGA to further reduce weight/complexity of plane.
- Logging number of input glitches

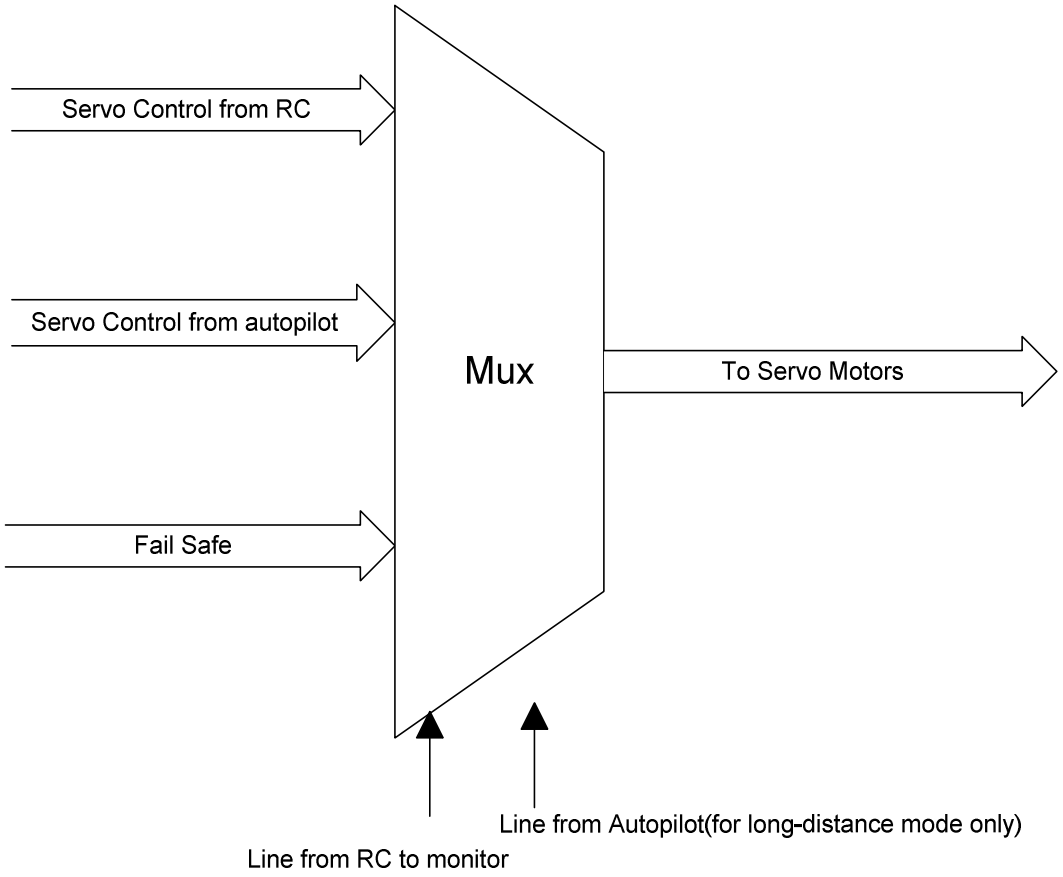
Currently detecting LOS on the hardware implementation Is buggy. As mentioned in the presentation, the hardware implementation works by looking at consecutive rising edges and finding the time difference between them. If it is outside a certain range we say this decision is a LOS and we input a '1' inside a shift registers used to keep a history of the last n decisions. We shift a '0' if that decision is not a LOS. We then perform a majority vote to determine if we have an actual LOS or not. The software implementation by taking sum of the difference between consecutive past pulse widths of the last n pulse widths. The software solutions seems to work properly(detects LOS correctly). Another thing to explore is that we consider it a LOS in the software solution also if are out of range (below .9 or above 2.1ms), the hardware implementation does not do this. Perhaps this is the actual reason he software implementation works better.

Not detecting LOS properly is a big problem because every time we detect LOS we start locking the servos into a glide position. This means the servos will be moving back and forth from glide position even in normal operation. Additionally if we are in failsafe and we LOS intermittently is not detected, the timer (to go into landing) resets. This means

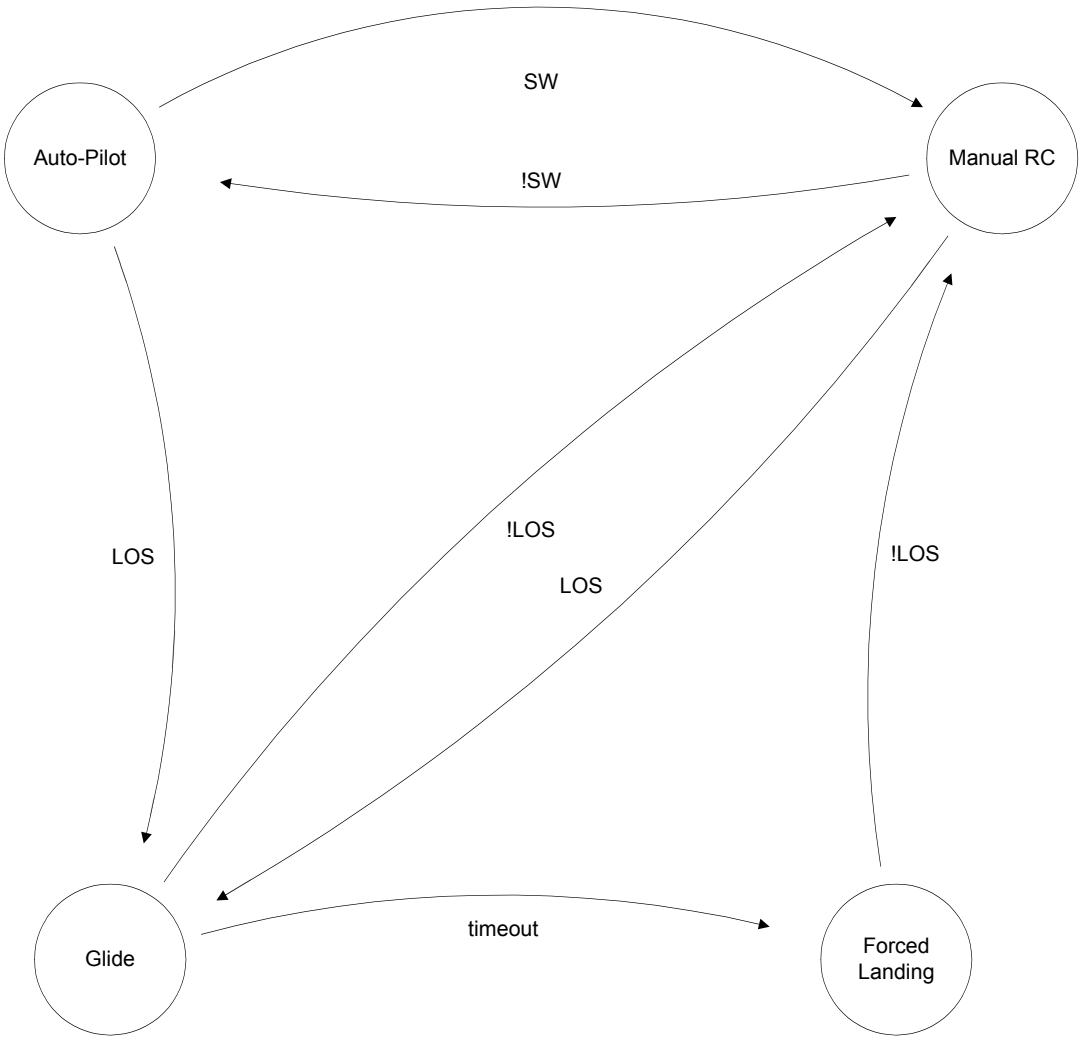
[3] Talk to UAARGS about what this was again exactly

we will never enter the landing state since the timer keeps resetting. To fix this currently in the hardware solution we do a small hack by putting a timer on the timer such that the timer only resets if it detects we have !LOS for a certain amount of time.

Design and Description of Operation



The device is intended to operate like a sophisticated multiplexer, choosing it servo control signal from multiple inputs. How which input is chosen depends on the state machine below:



SW: Signal from RC to switch between Manual and Autopilot control[4]

LOS: Loss of signal. Line monitored from RC to determine if signal was lost[5]

Timeout: Signal asserted when a certain amount of time has passed in the glide state and we haven't regained control of the plane

State	**Input chosen by Mux**
Manual RC	Servo Control from RC
Auto-Pilot	Servo Control from autopilot
Glide	Fail Safe

[4] Obtained from one of the servo control lines but have to ask UAARGS how exactly we know when we have LOS or when SW occurs, i.e. what kind of signals are we looking for here, constant 0 input when we have LOS?.
[5] Same as above

Forced Landing	Fail Safe

Signal	**Descriptions**
Servo Control from RC	Input user signal
Servo Control from autopilot	Input user signal
Fail Safe	Signal generated internally by FPGA. The signal generated depends on the current state the failsafe module is in. Currently it is unknown what type of signal UARRGS wants and if locking the servos in a static position is good enough for them.
LOS	Signal generated internally by FPGA derived from Servo Control from RC. LOS shall be determined by monitoring one of the RC Servo lines. If the pulse width had a duty cycle of less than 0.9ms or larger than 2.1ms, this shall indicate LOS.
SW	Signal generated internally by FPGA derived from Servo Control from autopilot. SW shall be determined by monitoring the same RC Servo as described as above. If the pulse width is from 0.9ms to 1.5ms, then SW is 1 (RC is in control), if the pulse width is from 1.5ms to 2.1ms then SW is 0 (Autopilot is in control)
Autopilot Health	Currently not used in short-distance mode FSM diagram, but would be in long-distance operation. Shall be determined by monitoring one of the Auto-pilot servo lines. If from 0.9 to 1.5 means healthy. If from 1.5 to 1.7 means unhealthy. From 1.7 to 2.1 is be left in case UAARGS wants to use another state. [6] Since the signals from the autopilot, the exact values is adjustable by the UAARGS. It has also been proposed to use one of the serial lines for Autopilot health monitoring. In which case autopilot health is indicated by both the servo lines and the serial line indicating unhealthiness.

The fail-safe control servo signal itself is will be generated by the FPGA, and depends on whether we are in Glide or Forced Landing state.

SW and LOS signals are monitored and derived from the servo lines.

[6] Double check these values again, forgot to take notes on this part

Currently three functionalities has been proposed:

1. Short-distance operation as described above
2. Long-distance operation as described above
3. If a failsafe generator can't be decided upon, the basic functionality of being able to switch between RC and autopilot via the SW signal is all that is needed.

So it seems to be desirable to be able to change functionality via a recompile or a re-configuration of parameters.

Optional:
In long-distance-mode a different transition diagram[7] will be required. We will have to also monitor and do some analysis from the Auto-Pilot line in order to figure out if it is in some 'bad' state or not.

[7] Todo: Add this transition diagram

Hardware Requirements

- The development FPGA[8] provided in the lab will be adequate for now. Probably not that many Configurable Logical Block(CLB) will need to be used which is optimal since we want to be able to perhaps move to a smaller FPGA. Currently we are using a Virtex II pro for development, but we will probably switch to a Spartan 3 starter.

- Servos for testing output. Currently we are using a FUTUBA S3102 given to us by UAARGS for testing.

- An RC receiver for testing input. Currently we use a function generator for testing input but this is somewhat unwieldy.

-To implement this outside of development, we'll need:[9]

- A breakout board for prototyping
- A Spartan 3 series FPGA (cheapest)
- PROM in order to configure FPGA on start up. (Assuming we don't want to have to program the FPGA every time we power it up).

Optional:

- PCB or schematic capture of a working layout
- Since weight is a factor on the plane, using a smaller FPGA would be nice.

[8] Virtex-II Pro FPGA board XUPV2P

[9] Probably can just copy the same setup group 2 is doing on this. We're kind of behind on this since we were under initial impression UAARGS that a prototype on a dev FPGA was okay for them.

Proposed parts, order list, supplier, cost

For now we may use the FPGA (Virtex II Pro[8]) provided by the lab.

Borrowed servo motors from the UAARG group.

RC controller and receiver from UAARG.

Spartan 3 series FPGA.[6], PROM, and breakout board for prototyping. http://www.sparkfun.com/commerce/product_info.php?products_id=8458 has all of this but we can probably just use the same one group 2 is getting. The less interfacing headaches, the better. Cost: 100$

Table of all user IO signals between FPGA and IO and non-digital world

All input/output signals are pulse-width modulated.

Required Power Signals[7] for Spartan 3	Nominal (V)	Min(V)	Max(V)
V_{CCINT}	1.2	1.14	1.26
V_{CCO}	Probably want Max in order to drive servos	1.14	3.465
V_{CCAUX}	2.375	2.5	2.625

The dev board we use takes 5V from the wall.

Signal Name	**Type**	**Range**
Servo Control from RC lines(9)	Input->FPGA	0-2.7V[10]
Servo Control from autopilot lines(9)	Input->FPGA	0-2.7V
Muxed Servo Control Line (9)	FPGA->output	Output signals to the Servos currently operate at 5V.[11] However current testing on the Hitec HS-475HB and the Futuba S3102 indicate that they will operate anywhere from 800mV to 5V[5]. So it seems that the output from FPGA should be enough to drive the servos.
(optional) telemetry/feedback lines	Input->FPGA	

Note: glitch detection and correction should be added for inputs.

[10] From UAARGS. FPGA will accept this voltage.
[11] Might still switch at 3.3V, so might be able to be used straight from FPGA output, double check with UAARGS

Datasheet

Software Implementation:

Design Summary

Number of errors: 0
Number of warnings: 5
Logic Utilization:
Total Number Slice Registers: 3,905 out of 27,392 14%
Number used as Flip Flops: 3,904
Number used as Latches: 1
Number of 4 input LUTs: 5,044 out of 27,392 18%
Logic Distribution:
Number of occupied Slices: 3,958 out of 13,696 28%
Number of Slices containing only related logic: 3,958 out of 3,958 100%
Number of Slices containing unrelated logic: 0 out of 3,958 0%
*See NOTES below for an explanation of the effects of unrelated logic.
Total Number of 4 input LUTs: 5,379 out of 27,392 19%
Number used as logic: 4,612
Number used as a route-thru: 335
Number used for Dual Port RAMs: 384
(Two LUTs used per Dual Port RAM)
Number used as Shift registers: 48
Number of bonded IOBs: 26 out of 556 4%
Number of RAMB16s: 32 out of 136 23%
Number of MULT18X18s: 3 out of 136 2%
Number of BUFGMUXs: 6 out of 16 37%
Number of DCMs: 1 out of 8 12%
Number of BSCANs: 1 out of 1 100%

```
Clock to Setup on destination clock sys_clk_pin
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
sys_clk_pin    |   13.039|         |         |         |
---------------+---------+---------+---------+---------+

Design statistics:
   Minimum period:  13.039ns{1}   (Maximum frequency:  76.693MHz)
```

Note this implies a maximum bus frequency of 76.693Mhz.

From Xilinx ISE Synthesis Report

Hardware Implementation:

Design Summary

Number of errors: 0
Number of warnings: 4
Logic Utilization:
Number of Slice Flip Flops: 214 out of 27,392 1%
Number of 4 input LUTs: 2,270 out of 27,392 8%
Logic Distribution:
Number of occupied Slices: 1,278 out of 13,696 9%
Number of Slices containing only related logic: 1,278 out of 1,278 100%

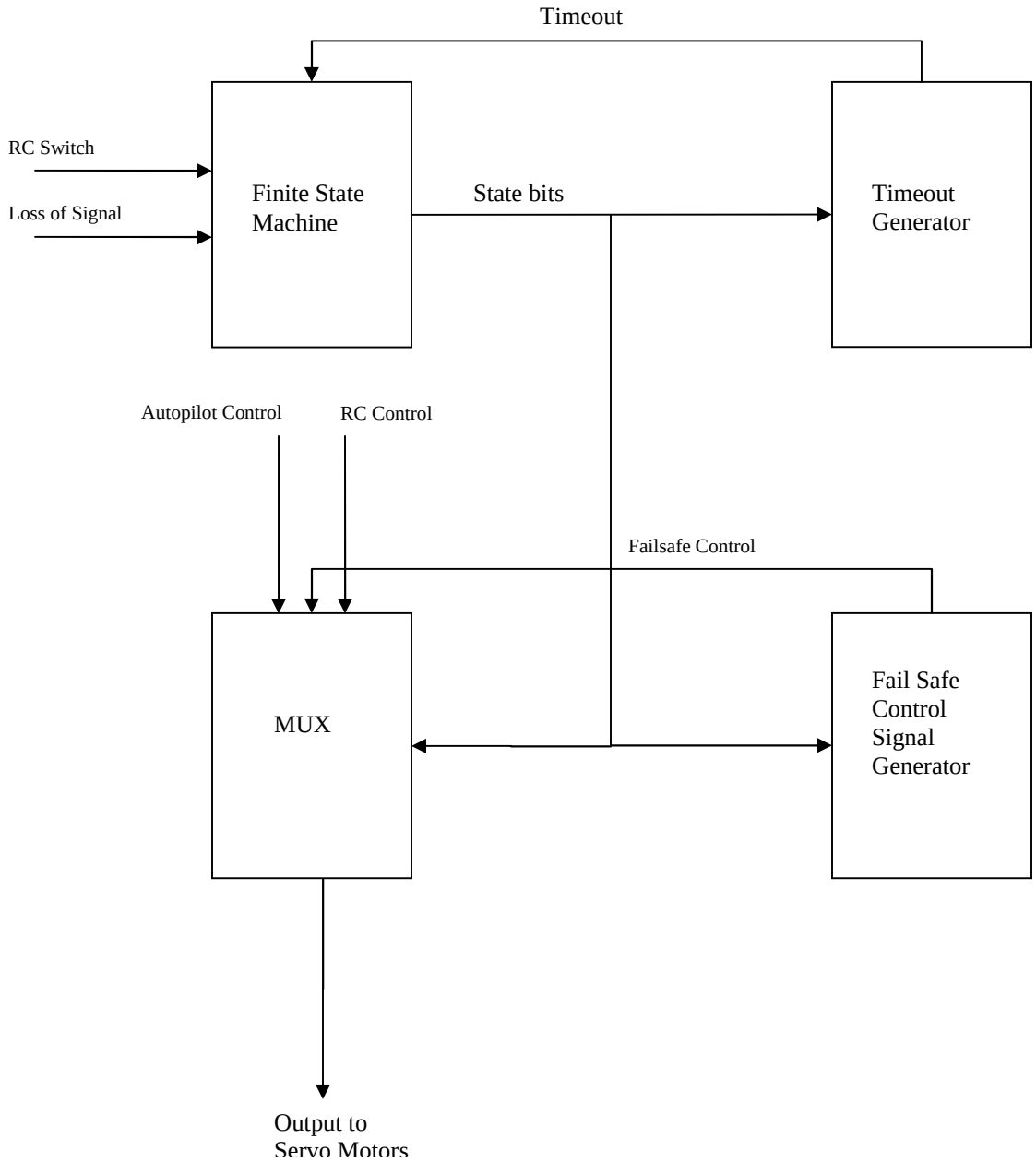
```
   Number of Slices containing unrelated logic:          0 out of   1,278   0%
     *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:       2,433 out of  27,392    8%
    Number used as logic:             2,270
    Number used as a route-thru:        163
  Number of bonded IOBs:                27 out of     556    4%
    IOB Flip Flops:                     1
  Number of BUFGMUXs:                   2 out of      16   12%

Timing Summary:
---------------
Speed Grade: -6

  Minimum period: 4.818ns (Maximum Frequency: 207.576MHz)
  Minimum input arrival time before clock: 3.222ns
  Maximum output required time after clock: 5.197ns
  Maximum combinational path delay: 5.308ns
```

From Xilinx ISE Synthesis Report

Software Design (Implemented only in VHDL so far)

4 VHDL Components:

1. State Machine which changes state by monitoring RC and autopilot lines.
2. Mux which chooses input line depending on state of machine
3. Fail Safe Control generates fail-safe signal depending on state of machine
4. Timer for the timeout used to switch from glide to forced landing



Block Diagram

Require additional pulse-width-checker component to be used to generate LOS/SW signals by measuring pulse widths.
Also require an additional component to clean up glitches from inputs.

See *Design and Description of Operation* for FSM and Mux diagrams.

See *Appendix A* for VHDL code.

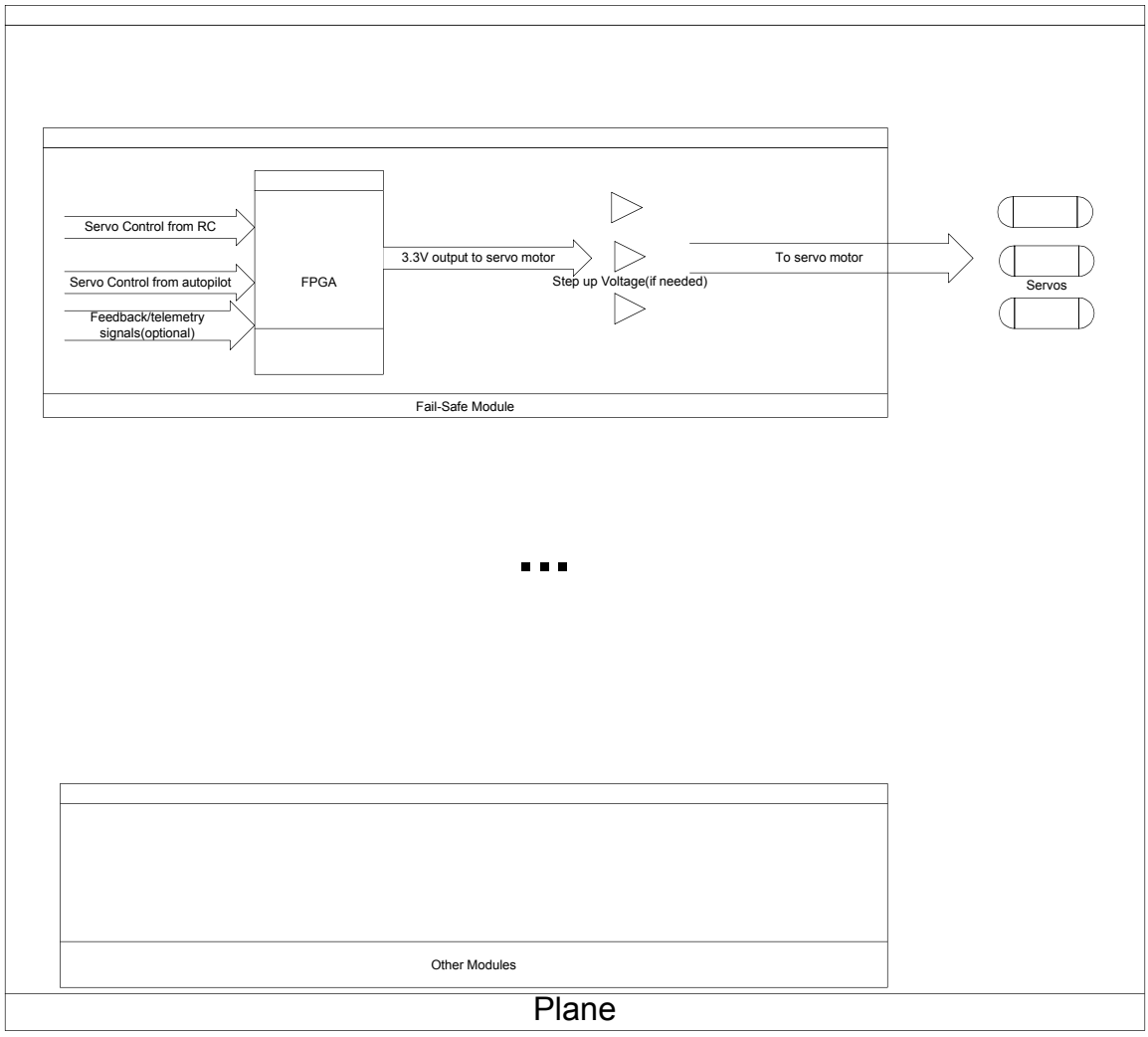
See *Appendix B* for prototype C code.

Autopilot Control and RC Control are inputs, Output to Servo Motors is output. How to get RC switch and LOS signals has been determined but the components have not yet been created in VHDL.

For software (C) implementation, we will have to compile cores using the EDK and provide the ability to access pulse widths from inputs. Probably do this by writing VHDL code which reads in pulse widths from the inputs and writes down the length (in cycles) into a register somewhere accessible by memory map from C. Also allow C code to write an integer to a memory mapped register and have VHDL code output that integer as pulse width modulated signal. Also might want to use interrupts from VHDL to indicate when new input comes in. Figuring out how to do this might take awhile.

We have managed to do everything described in the above paragraph, except for the interrupts. The software architecture remains similar to the hardware architecture as described in the diagram.

Full Schematic showing all components[12]



[12] Todo: Add VHDL component diagrams, and define their interfaces

Test Plan

RC Signals and Autopilot signals are generated by function generator and used as input to the FPGA.
The Control signals SW and LOS will be generated using switches on the FPGA board.
Timeout will be generated when LOS is continuously on for a certain amount of time.
Different LED light will be turned on depending on what state the module is in.
LED1: Auto-Pilot, LED2: Manual RC, LED3: Glide, LED4: Forced Landing
We check the output servo motor behaves according to the corresponding input for each state.

Test Cases	Input to test case	Expected Output
Current State: Autopilot LED1 on	!LOS & SW	Next State: Manual RC. LED1 off, LED2 on.
	!LOS & !SW	Next State: Autopilot. LED1 stays on. AP input get passed on to the output pins
	LOS	Next State: Glide LED1 off, LED3 on.
Current State: Manual RC LED 2 on	!LOS & SW	Next State: Manual RC. LED2 stays on. RC input get passed on to the output pins
	!LOS & !SW	Next State: Autopilot. LED2 off, LED1 on.
	LOS	Next State: Glide LED2 off, LED3 on.
Current State: Glide LED 3 on	!LOS	Next State: Manual RC. LED3 off, LED2 on.
	LOS & Timeout	Next State: Forced Landing LED3 off, LED4 on.
	LOS & !Timeout	Next State: Glide LED3 stays on. The motor go to a predefined glide position
Current State: Crash LED 4 on	!LOS	Next State: Manual RC. LED4 off, LED2 on.
	LOS	Next State: Forced Landing LED4 stays on. Motor switched to crashing pattern

Results of experiments and characterization:

Input servos operate from 800mV to 5V and with a pulse width period of 50ms to 400ms.

Pulse width resolution of 10ns (limited by clock frequency of Virtex 2 FPGA).

Frequency of RC receiver is actually 45mhz (22.2222ms Period).

Only short-distance operation implemented in hardware implementation as design effort is a lot harder on hardware. Spartan 3 chosen to ideally to go on the airplane as it has little cost (~10$).

Servo motor behaving correctly as stated in the expected output.

Integrated Circuit Design:

From Synopsis: with clock period of 6ns.
Using VHDL code for hardware implementation
Require modification to the top2_level.vhd to add pin pads.

```
****************************************
Report : area
Design : fail_top
Version: Y-2006.06-SP4
Date   : Mon Apr 12 18:02:39 2010
****************************************

Library(s) Used:

    GSCLib_2.0 (File: /EDA/kits/gpdk18/GSCLib_3.0/timing/GSCLib_3.0.db)
    GSCLib_IO (File: /EDA/kits/gpdk18/GSCLib_IO_1.4/timing/GSCLib_IO.db)

Number of ports:              27
Number of nets:               67
Number of cells:              41
Number of references:          4

Combinational area:       675000.000000
Noncombinational area:         0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          675000.000000
Total area:                 undefined
Loading db file '/EDA/kits/gpdk18/GSCLib_3.0/timing/GSCLib_3.0.db'
Loading db file '/EDA/kits/gpdk18/GSCLib_IO_1.4/timing/GSCLib_IO.db'
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: Design has unannotated primary inputs. (PWR-414)
Warning: Design has unannotated sequential cell outputs. (PWR-415)

****************************************
Report : power
        -analysis_effort low
Design : fail_top
Version: Y-2006.06-SP4
Date   : Mon Apr 12 18:02:40 2010
****************************************


Library(s) Used:

    GSCLib_2.0 (File: /EDA/kits/gpdk18/GSCLib_3.0/timing/GSCLib_3.0.db)
    GSCLib_IO (File: /EDA/kits/gpdk18/GSCLib_IO_1.4/timing/GSCLib_IO.db)


Operating Conditions: typical   Library: GSCLib_2.0
Wire Load Model Mode: top


Global Operating Voltage = 3
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =  65.6385 mW   (98%)
  Net Switching Power  =   1.4936 mW    (2%)
                         ---------
```

```
Total Dynamic Power    =  67.1321 mW  (100%)

Cell Leakage Power     = 910.6193 uW


****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : fail_top
Version: Y-2006.06-SP4
Date   : Mon Apr 12 18:02:42 2010
****************************************

Operating Conditions: typical   Library: GSCLib_2.0
Wire Load Model Mode: top

  Startpoint: c1/c3/cnt_reg_0_
              (rising edge-triggered flip-flop clocked by padClk)
  Endpoint: c1/c3/cnt_reg_31_
              (rising edge-triggered flip-flop clocked by padClk)
  Path Group: padClk
  Path Type: max

  Point                                       Incr       Path
  ------------------------------------------------------------------------------
  clock padClk (rise edge)                    0.00       0.00
  clock network delay (ideal)                 0.00       0.00
  c1/c3/cnt_reg_0_/CK (SDFFSRX1)                         0.00       0.00 r
  c1/c3/cnt_reg_0_/Q (SDFFSRX1)                          0.30       0.30 f
  c1/c3/add_115/A_0_ (failsafe_signal_gen_n7_DW01_inc_0)
                                              0.00       0.30 f
  c1/c3/add_115/U1_1_1/CO (ADDHX1)                       0.10       0.40 f
  c1/c3/add_115/U1_1_2/CO (ADDHX1)                       0.07       0.47 f
  c1/c3/add_115/U1_1_3/CO (ADDHX1)                       0.07       0.55 f
  c1/c3/add_115/U1_1_4/CO (ADDHX1)                       0.07       0.62 f
  c1/c3/add_115/U1_1_5/CO (ADDHX1)                       0.07       0.70 f
  c1/c3/add_115/U1_1_6/CO (ADDHX1)                       0.07       0.77 f
  c1/c3/add_115/U1_1_7/CO (ADDHX1)                       0.07       0.84 f
  c1/c3/add_115/U1_1_8/CO (ADDHX1)                       0.07       0.92 f
  c1/c3/add_115/U1_1_9/CO (ADDHX1)                       0.07       0.99 f
  c1/c3/add_115/U1_1_10/CO (ADDHX1)                      0.07       1.07 f
  c1/c3/add_115/U1_1_11/CO (ADDHX1)                      0.07       1.14 f
  c1/c3/add_115/U1_1_12/CO (ADDHX1)                      0.07       1.22 f
  c1/c3/add_115/U1_1_13/CO (ADDHX1)                      0.07       1.29 f
  c1/c3/add_115/U1_1_14/CO (ADDHX1)                      0.07       1.36 f
  c1/c3/add_115/U1_1_15/CO (ADDHX1)                      0.07       1.44 f
  c1/c3/add_115/U1_1_16/CO (ADDHX1)                      0.07       1.51 f
  c1/c3/add_115/U1_1_17/CO (ADDHX1)                      0.07       1.59 f
  c1/c3/add_115/U1_1_18/CO (ADDHX1)                      0.07       1.66 f
  c1/c3/add_115/U1_1_19/CO (ADDHX1)                      0.07       1.74 f
  c1/c3/add_115/U1_1_20/CO (ADDHX1)                      0.07       1.81 f
  c1/c3/add_115/U1_1_21/CO (ADDHX1)                      0.07       1.88 f
  c1/c3/add_115/U1_1_22/CO (ADDHX1)                      0.07       1.96 f
  c1/c3/add_115/U1_1_23/CO (ADDHX1)                      0.07       2.03 f
  c1/c3/add_115/U1_1_24/CO (ADDHX1)                      0.07       2.11 f
  c1/c3/add_115/U1_1_25/CO (ADDHX1)                      0.07       2.18 f
  c1/c3/add_115/U1_1_26/CO (ADDHX1)                      0.07       2.26 f
  c1/c3/add_115/U1_1_27/CO (ADDHX1)                      0.07       2.33 f
  c1/c3/add_115/U1_1_28/CO (ADDHX1)                      0.07       2.40 f
  c1/c3/add_115/U1_1_29/CO (ADDHX1)                      0.07       2.48 f
  c1/c3/add_115/U1_1_30/CO (ADDHX1)                      0.07       2.55 f
  c1/c3/add_115/U1/Y (XOR2X1)                            0.08       2.63 f
  c1/c3/add_115/SUM_31_ (failsafe_signal_gen_n7_DW01_inc_0)
                                              0.00       2.63 f
  c1/c3/U43/Y (AND2X1)                        0.05       2.67 f
  c1/c3/cnt_reg_31_/D (SDFFSRX1)              0.00       2.67 f
  data arrival time                                      2.67
```

```
clock padClk (rise edge)                          6.00      6.00
clock network delay (ideal)                       0.00      6.00
c1/c3/cnt_reg_31_/CK (SDFFSRX1)                   0.00      6.00 r
library setup time                               -0.16      5.84
data required time                                          5.84
-------------------------------------------------------------------------
data required time                                          5.84
data arrival time                                          -2.67
-------------------------------------------------------------------------
slack (MET)                                                 3.17
```

From Encounter:

```
# generated on Mon Apr 12 16:49:04
# Top Cell: fail_top

------------------------------------------------------------
        timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|    Setup mode      |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|          WNS (ns):|  2.700  |  2.700  |  4.150  |   N/A   |   N/A   |   N/A   |
|          TNS (ns):|  0.000  |  0.000  |  0.000  |   N/A   |   N/A   |   N/A   |
|   Violating Paths:|    0    |    0    |    0    |   N/A   |   N/A   |   N/A   |
|        All Paths:|   265   |   230   |   97    |   N/A   |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+

Density: 100.000%
Real DRV (fanout, cap, tran): (0, 15, 0)
Total DRV (fanout, cap, tran): (0, 16, 0)
------------------------------------------------------------
# generated on Mon Apr 12 16:49:10
# Top Cell: fail_top

------------------------------------------------------------
        timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+---------+---------+
|    Hold mode       |   all   | reg2reg | in2reg  | reg2out | in2out  | clkgate |
+--------------------+---------+---------+---------+---------+---------+---------+
|          WNS (ns):|  0.134  |  0.368  |  0.134  |   N/A   |   N/A   |   N/A   |
|          TNS (ns):|  0.000  |  0.000  |  0.000  |   N/A   |   N/A   |   N/A   |
|   Violating Paths:|    0    |    0    |    0    |   N/A   |   N/A   |   N/A   |
|        All Paths:|   265   |   230   |   97    |   N/A   |   N/A   |   N/A   |
+--------------------+---------+---------+---------+---------+---------+---------+

Density: 100.000%
------------------------------------------------------------
```

From Cadence: DRC

```
\o *********  Summary of rule violations for cell "fail_safe layout"  *********
\o  # errors  Violated Rules
\o       219  WARNING: Gate used as conductor...
\o       219  Total errors found
```

Please see *Appendix C* for file used and generated in the IC design process.

Appendices

Quick Start Manual

For Software Implementation:

See http://www.ece.ualberta.ca/~elliott/cmpe490/appnotes/2010w/edk_virtex2/Appnotes/appnotes.html for this

For Hardware Implementation:

Simply import all vhdl files and top2_level.ucf into a project. The top level file should be set to top2_level.vhd. Test benches *.tbw are also provided though changing generics defaults might be necessary for them to work as intended.

Files are here:

http://www.ece.ualberta.ca/~elliott/cmpe490/appnotes/2010w/edk_virtex2/Appnotes/Files/Hardware%20Implementation

Graphical Hierarchy of Source Code

See:

http://www.ece.ualberta.ca/~elliott/cmpe490/appnotes/2010w/edk_virtex2/Appnotes/Files/

Files/
|
|-->Hardware Implementation
        |
        --> top2_level.vhd –topmost level to synthesize with
        --> *.vhd
        --> *.tbw –testbenches
        --> top2_leve.ucf – user constraint file
|-->Software Implementation
        |
        --> C code
                |
                --> *.c files
        --> VHDL
                --> system.ucf –user constraint
                --> *.vhd

Citations

[1] http://www.ece.ualberta.ca/~uaarg/UAARGV3/index2.html
[2] http://www.markschulze.net/java/meanmed.html
[3] http://www.xilinx.com/support/documentation/application_notes/xapp097.pdf - Xilinx FPGAs: A Technical Overview for first-time users
[4] http://www.datasheetarchive.com/datasheet-pdf/040/DSA00102215.html
[5] http://www.ece.ualberta.ca/~elliott/cmpe490/appnotes/2010w/Hobby_Servo_Sig_and_controller/
[6] http://www.xilinx.com/support/documentation/spartan-3_data_sheets.htm
[7] http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf
[8] http://www.xilinx.com/support/documentation/virtex-ii_pro_data_sheets.htm

Appendix A*: Hardware Implementation*

See:
http://www.ece.ualberta.ca/~elliott/cmpe490/appnotes/2010w/edk_virtex2/Appnotes/Files/Hardware%20Implementation/

Appendix B: Software Implementation

See
http://www.ece.ualberta.ca/~elliott/cmpe490/appnotes/2010w/edk_virtex2/Appnotes/Files/Software%20Implementation/

Appendix C*: IC design Files and Output*

Code used and generated by the IC design process can be found from :
http://www.ece.ualberta.ca/~elliott/cmpe490/projects/2010w/g8_failsafe/

Include codes:

1. VHDL codes for Synopsys:	**fail_safe_VHDL.tar**
2. Script for running Synopsys:	**compile_fail.tcl**
3. Synthesized files from Synopsys:	**synopsys_output.tar**
4. Script for running Encounter:	**encounter_script**
5. DEF file from Encounter:	**fail_top.def**