Star Award

Artificially Intelligent Autonomous Aircraft Navigation System Using a Distance Transform on an FPGA

Institution: **Monash University**

Participants: **Nathan William Smith**

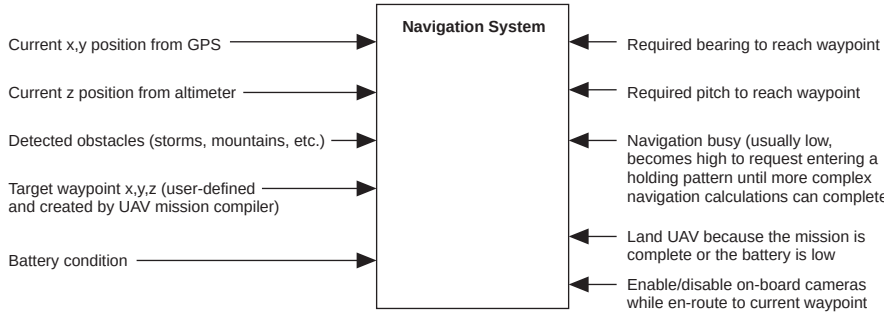
Instructor: **Dr. Andrew Price**

Design Introduction

This project is an artificially intelligent navigation system for use on board unmanned aerial vehicles (UAVs). The system takes the available input data and a user-defined mission statement, integrates the data, and produces navigation reference signals that are used by other systems to control the plane. Figure 1 shows the UAV navigation system block diagram, which is implemented in an Altera® Cyclone® FPGA. The output is continually updated based on the dynamic input from changes in the UAV or obstacle positions. The navigation system produces output signals to other UAV systems, e.g., a signal indicates that the UAV should land because the mission is complete or because an emergency landing is required.

The navigation system also produces a busy signal, which is usually inactive. This signal indicates that the navigation system is busy recalculating several parameters due to complex obstacles in the flight path, and the system will not produce the required pitch and heading for some time. In this case, the busy signal causes the flight control system to enter a holding pattern (i.e., causes the UAV to travel in a circle) until the navigation calculations are complete.

Figure 1. UAV Navigation System Block Diagram



The navigation system uses a novel two-level hierarchical distance transform (DT) algorithm in three dimensions to control the aircraft's longitude, latitude, and altitude. The DT technique is commonly used in signal processing, specifically image processing. However, DT has been used more recently for the robotic navigation of land-based vehicles over low ranges. This project applies the DT for aerospace vehicles in three dimensions over long-range missions.

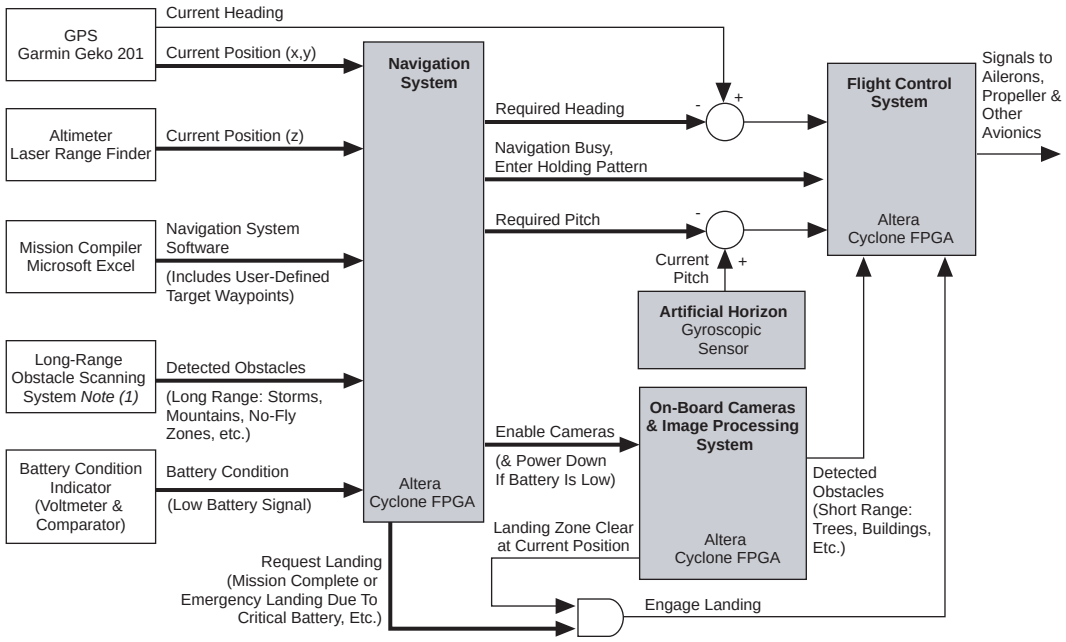
The project also involves significant systems integration and systems engineering. The navigation system must interface and communicate with other hardware in the overall UAV system. For example, a simple interface includes voltage-level signaling for one-bit signals, such as when to land the UAV. More complex interfacing occurs for the global positioning system (GPS) where a National Marine Electronics Association (NMEA) 183 communications standard applies, and for the RS-232 serial communications required for specific navigation signals (pitch and bearing).

I chose to implement the Nios® II system on a Cyclone FPGA for the following reasons:

- The application must perform computationally complex algorithms quickly.
- The application receives inputs from a variety of communications media, such as from the GPS.
- The application has to debug each part of the design in a user-friendly environment.

Figure 2 describes the required systems engineering and interfaces from the navigation system to other UAV systems. Grey blocks show the systems that are within the project scope. The navigation system produces or communicates with the interfaces indicated by bold arrows.

Figure 2. UAV System and Interfaces Block Diagram



Note to Figure 2:

(1) Specific hardware is unknown; possible wireless TCP/IP to meterology and topographical site.

Function Description

The heart of the navigation system hardware is an Altera Cyclone FPGA, which is incorporated into Monash University's miniboard (called the Monash miniboard). Figure 3 shows the board and describes its main components. The FPGA is used as a powerful microcontroller that can implement custom logic or microprocessor systems by configuring the device using software. To configure this device on this board requires the Altera Quartus® II software version 5.1.

Figure 3. Monash Miniboard

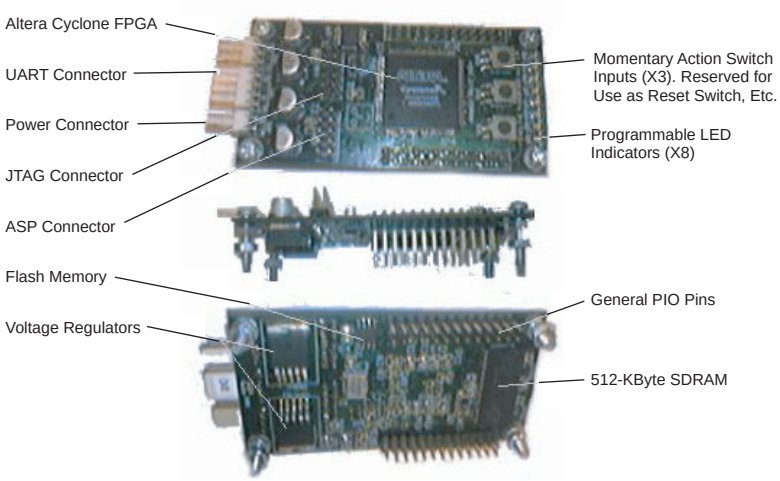


Figure 4 shows the Quartus II project implementing the navigation system. To create the project, I performed the following general steps:

1. Drew a block diagram of the desired system in the Quartus II software, including a block (NAV_Processor) for a custom microprocessor that I developed with SOPC Builder.

2. Added pin assignments to the block diagram (described later in Table 1), according to the physical pins that were available on the FPGA. Pins include I/O pin blocks that communicate to specific pins on the FPGA, such as pins specific to the navigation system I/O as defined in Figure 1.

3. Used the Quartus II software to analyze and compile the design.

4. Used the Quartus II software to configure the FPGA with the configuration information.

After the configuration information was loaded into the FPGA, I could upload the specific C/C++ navigation software (provided from the mission complier) to the FPGA's microprocessor. I could then navigate the UAV for the desired mission.

Figure 4. Quartus II Project

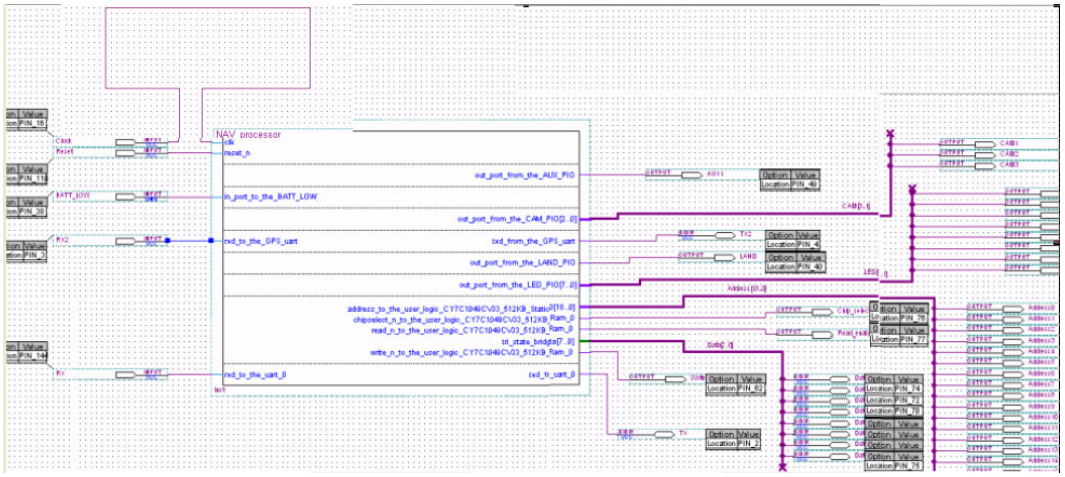
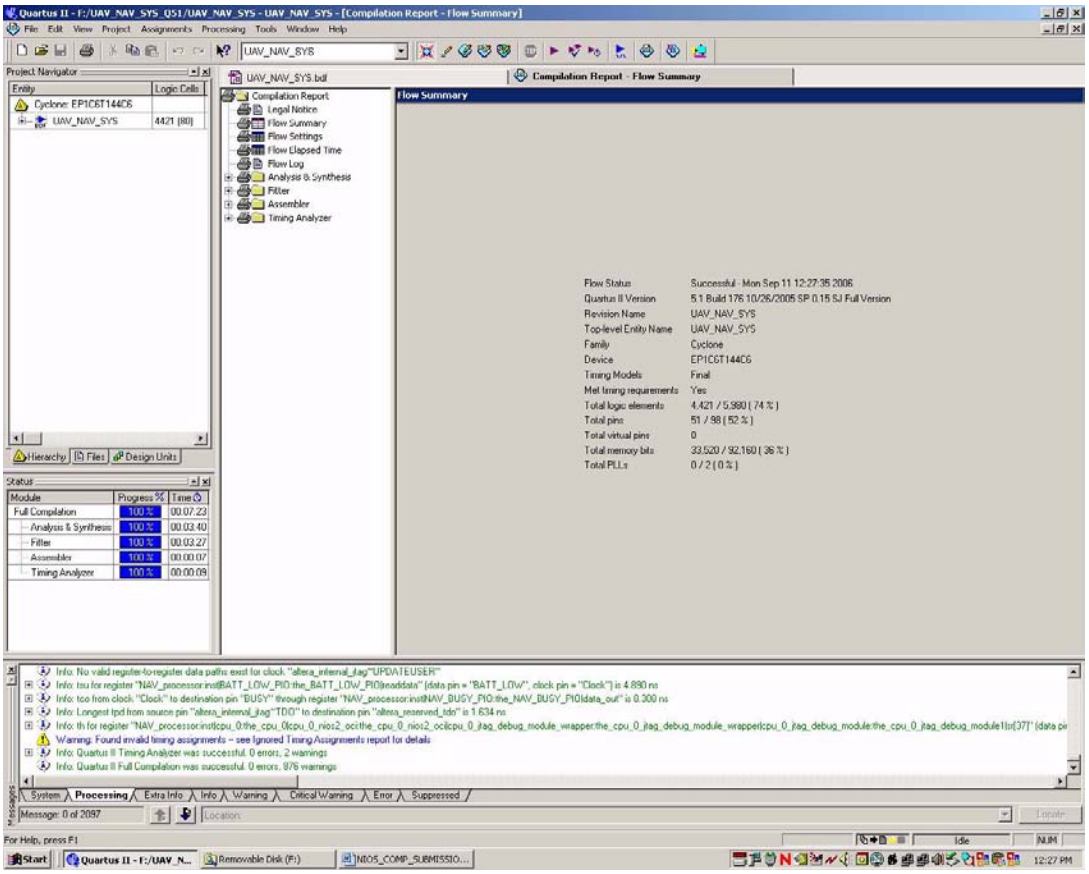


Figure 5 shows the Quartus II compilation report, which describes the system usage. This project uses more than 70% of the logic elements (LEs) available on the FPGA.

Figure 5. Quartus II Compilation Report



A critical part of the system is the microprocessor, which executes the navigation system software to compute a DT and generate navigation control signals. The SOPC Builder graphical interface made it easy to select a Nios II processor and connect it to the navigation system hardware. SOPC Builder provides three processor variants, and lets you customize the selected processor. Depending on the microprocessor system being developed, SOPC Builder allows the the designer to include UART outputs, Ethernet interfaces, VGA displays, etc., provided that the selected components fit into the memory/LEs available on the target FPGA.

Figure 6 shows the navigation system's microprocessor configuration in SOPC Builder, including all of the required hardware. The project uses the Nios II/f variant because of its speed and data cache. I selected the cache size and pipeline performance by trial and error; I wanted to use the largest possible cache configuration while still fitting the software and configuration data into the Cyclone device's memory.

Figure 6. Navigation System's Processor Configuration in SOPC Builder

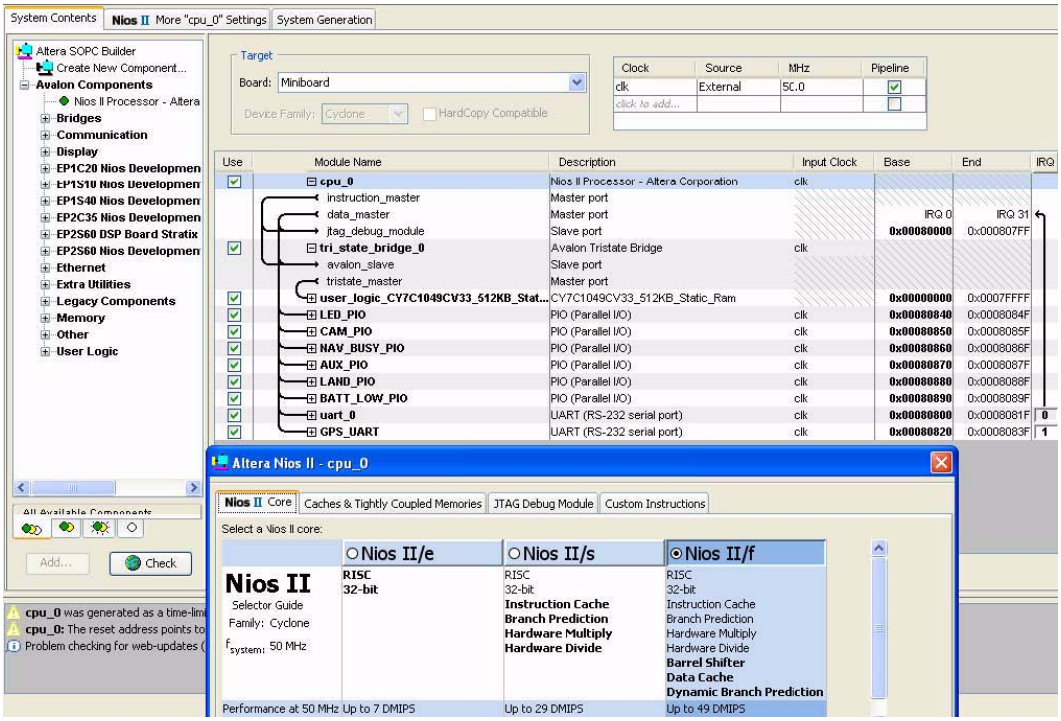


Table 1 shows the pin configuration for the navigation system's test hardware, which was the Monash miniboard.

Table 1. Navigation System Pin Configuration (Part 1 of 2)

Pin Number	Type	Function
16	Input	Clock (CLK)
110	Input	Reset (Reset)
3, 4	Input Bidirectional	UART 0, *Note (1)* (RX, TX)
N/A	JTAG UART	GPS_UART, *Note (2)* (RX, TX)
108, 100, 104, 106, 107, 109, 103,1 05	Output	LED PIO (LED 1-8)
143	Output	Battery Low Signal PIO (BATT)
141	Output	Landing Signal PIO (LAND)
96, 84, 82, 78, 60, 58, 56, 52, 51, 53, 57, 59, 61, 67, 79, 83, 85, 97	Output	Address Bus to 512-Kbyte SRAM (ADR 0 - 18)
74, 72, 70, 68, 69, 71, 73, 75	Bidirectional	Data Bus to 512-Kbyte SRAM (DAT 0 - 7)
76	Output	Chip Select to 512-Kbyte SRAM
77, 62	Output	Read, Write Enable to 512-Kbyte SRAM
134, 133, 132	Output	Camera Control PIO (CAM1 - CAM3)

Table 1. Navigation System Pin Configuration (Part 2 of 2)

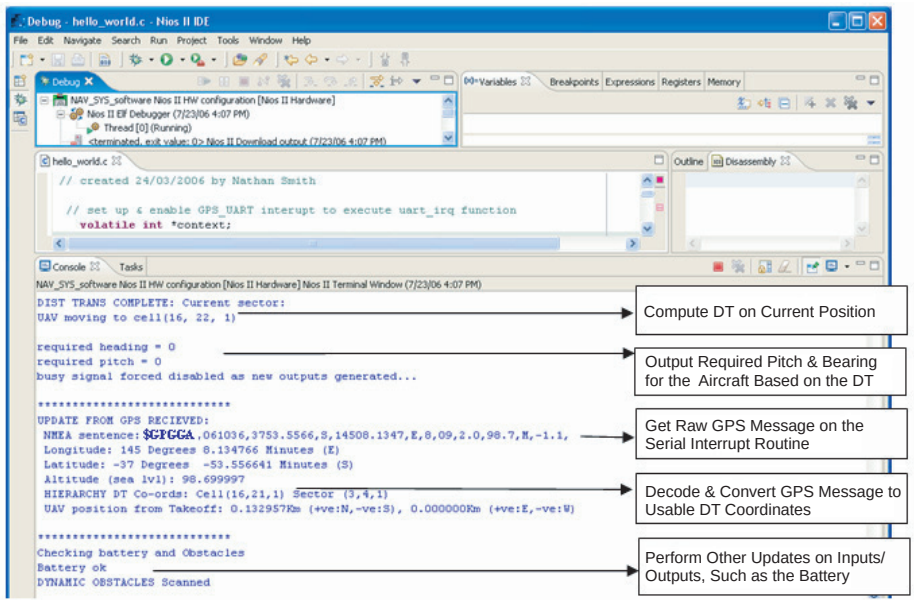
Pin Number	Type	Function
142	Output	Navigation Busy PIO (PUSY)
140	Output	Ancillary UAV Output PIO (AUX1)

Notes to Table 1:

(1) UART 0 is the connection from the miniboard to the PC or to the flight control board. It outputs the required pitch and bearing control signals as a text stream for decoding by other hardware.
(2) GPS_UART is the connection from the miniboard to the GPS. It uses the JTAG UART interface in the miniboard and has no pins specified in the Quartus II software.

With the microprocessor design uploaded to the FPGA, I loaded the navigation system software, which I developed using the Nios II Integrated Development Environment (IDE), into the microprocessor. Figure 7 shows the navigation system's output stream, which is displayed by the Nios II terminal. The system outputs more information than that required by the DT algorithm (i.e., the pitch and heading data). The extra data, such as the UAV longitude and latitude, is useful for debugging and data recoding of the actual UAV flight path.

Figure 7. Navigation System Output Stream



Performance Parameters

The most important performance parameter for the navigation system is that it performs a DT computation over a large three-dimensional (3-D) mission area (requiring two 3-D arrays in software to perform the computation) in the time that the GPS receives and outputs the next positional fix, which is roughly 2 to 3 seconds.

SOPC Builder allows the user to configure additional aspects of the microprocessor to improve computation speed, at the expense of using more system memory and LEs. Specifically, the user can control the core type (Nios II/s, Nios II/f, etc.), pipelining, hardware multiply and divide, and cache allocation. Pipelining allows multiple instructions to be fed into each stage of the microprocessor execution cycle in parallel, enabling maximum execution performance of the navigation system software. Larger caches provide more memory data storage, which makes code execute faster. A large

cache is particularly useful for the navigation system software, which uses an incremental iterative process (i.e., values in a DT matrix are updated in a scanned incremental manner) to determine the DT. However, larger caches also use more FPGA LEs and memory, and the designer can inadvertently create a system that does not fit into the target Cyclone device.

Ultimately, I selected the cache size and pipelining based on trial and error, with the goal of maximizing the cache size while still fitting the design into the Cyclone FPGA. The variant I chose met the design's timing requirements. Table 2 compares this variant and other hardware options.

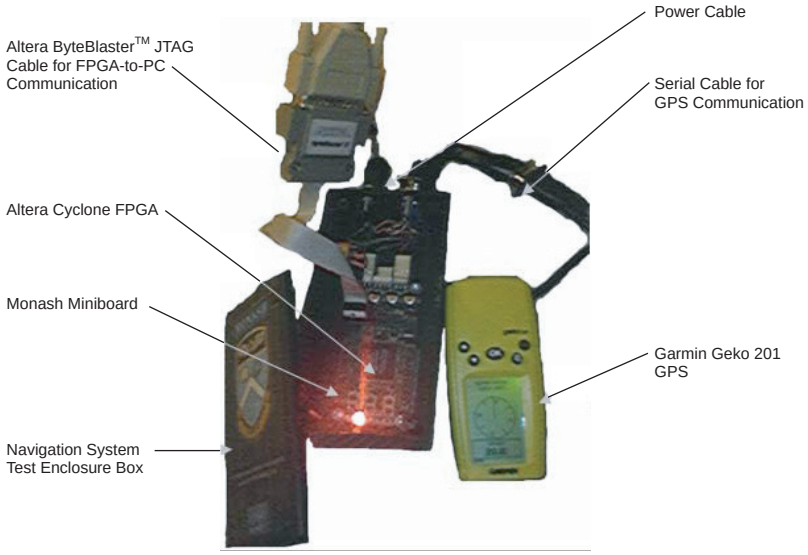
Table 2. Hardware Configuration and Performance Characteristics

Nios II Processor	Instruction Cache Size (Kbytes)	Data Cache (Kbytes)	Clock Pipeline	Design Fits in Cyclone Device	Time to Perform DT Algorithm in 3-D Array (25 x 25 x 25)
Nios II/s	2	N/A	Yes	Yes	8 seconds
Nios II/f	2	N/A	Yes	Yes	4 seconds
Nios II/f	2	1	Yes	No	N/A, no fit
Nios II/f	1	1	Yes	No	N/A, no fit
Nios II/f	1	512	Yes	Yes	2 seconds (meets GPS timing requirements)

Design Architecture

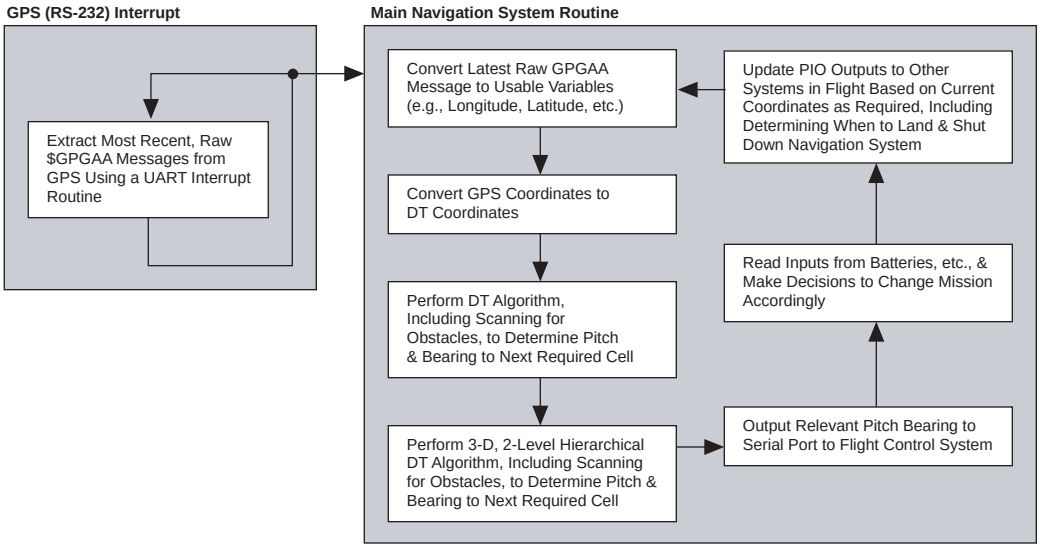
Figure 1, shown previously, provides the hardware design, and Figure 2 shows the systems integration of the navigation system and other UAV hardware. I implented this design in the Monash miniboard and Garmin Geko 201 GPS hardware (see Figure 8).

Figure 8. Hardware Construction



The navigation system software flow chart in Figure 9 provides information about the software building blocks used to develop the project.

Figure 9. Navigation System Software Flow Chart



Design Description

The design's implementation steps are as follows:

- Research and determine the validity of the DT technique for navigation, and develop software algorithms to support it. Research the use of Altera FPGAs using the Quartus II software, SOPC Builder, and Nios II IDE. Use the web editions of the software and documentation available from the Altera web site (**www.altera.com**).

- Create a Quartus II project applicable to the device on the Monash miniboard, selecting an appropriate Nios II processor in SOPC Builder, as shown in Figures 4 and 6. Compile and debug the Quartus II design and review the compilation report shown in Figure 5. Test this project, including testing the microprocessor response, and miniboard hardware (UART communications, etc.), with the simple hello world Nios II program.

- Create a 3-D, two-level hierarchical, DT algorithm implemented in the Nios II processor. Test the performance in the Cyclone device with the appropriate test input.

- Update the SOPC Builder processor configuration to determine the fastest possible configuration that fits in the device's memory and available LEs (as outlined in Table 2). Optimize the processor until the desired performance requirement is met.

- Create interrupt-based interfaces using the Nios II IDE to control the appropriate input and output to integrate the navigation system with other systems. Test these I/O interfaces.

- Test the complete navigation system performance using available land-based vehicles and altitude information relative to sea level provided from the GPS. A UAV was not available at this time because other systems on board the UAV were not complete.

Future work will involve completing other UAV systems as shown in Figure 2, interfacing them with the navigation system, and installing the navigation system on board a UAV. Once installed, I can perform live testing in an airborne (rather than land-based) environment.

Design Features

This design features an Altera Cyclone FPGA. While Altera also offers some higher-performance device families (e.g., Cyclone II, Stratix® and Stratix II devices), the Cyclone device used in this project meets the performance requirements of the complicated navigation system, which uses somewhat computationally and memory-intensive DT algorithms. The Cyclone device is built onto the Monash miniboard, which includes 512-Kbyte on-board static RAM for storing instruction and other program memory from the Nios II processor, as well as other electronic interfaces that support the Cyclone device (power regulators, I/O connectors, communications controllers, etc.). The design hardware also uses a Garmin Geko 201 GPS system, which interfaces with the Cyclone device. SOPC Builder makes it easy to create interfaces between hardware and the Cyclone device.

Additionally, there are several features of the software programmed onto the device, as follows:

- A two-level, 3-D sweeping DT algorithm gives the UAV associated intrinsic artificial intelligence (AI) to plan and dynamically follow a flight path to predefined waypoints, and provides compensating vectors when the UAV flies off course.
- A DT orbiting routine uses the DT coordinate system to orbit a waypoint once it is reached. This routine supports missions in which the UAV may take aerial surveillance photography over waypoints, or gather meteorological data at the waypoint with appropriate sensors.
- A NMEA 183 GPS serial string decoder algorithm decodes GPS data and converts it into X/Y cell and sector equivalents for use by the DT algorithm.
- The system provides software control of all other required I/O signals, as shown in Figure 2.
- The system provides communications monitoring of the GPS interface, and can determine whether the GPS has updated data within the last 6 or more seconds (most likely due to a temporary loss of GPS satellite feed in flight). Should this situation occur, the navigation system signals that it is busy. It then requests a holding pattern for the flight control system until GPS communications are restored, at which point, the navigation system resumes its mission.

Conclusion

Using the DT, I developed a reliable, artificially intelligent navigation system for use on a UAV's on-board FPGA hardware. The system can plan a flight path dynamically, based on data obtained by other hardware, such as a GPS, and the user-defined mission generated by the mission compiler. The system also makes various other AI decisions, such as when to attempt landing the UAV.

This design involves several systems integration issues, as well as a variety of communication with other systems, such as JTAG, RS-232 UART, and programmable I/O (PIO). The Quartus II software, SOPC Builder, and Nios II IDE allow the user to implement a wide variety of I/O, and program and test it quickly. With such simple, powerful systems design and integration packages, the Altera development environment, along with the FPGAs capable of supporting such designs, is the best choice to satisfy the navigation system's requirements.

Additionally, SOPC Builder's ability to customize the Nios II processor configuration, including cache, pipelining, hardware multipliers, devices, etc., let me develop a highly efficient custom microprocessor that is capable of computing the DT algorithm over a large mission area for UAV systems. Without this control, I would not have been able to implement the project's navigation system with the required performance parameters.

References

Grevera, G J, “The ‘dead reckoning’ signed distance transform,” *Computer Vision and Image Understanding* 95 (2004): 317-333.

Qin-Zhong Ye, “The signed Euclidean Distance Transform and its Applications” *IEEE* (1988): 495-499.

Jarvis.R, “An all-terrain intelligent autonomous vehicle with sensor-fusion-based navigation capabilities” *Control Engineering Practice*, vol. 4, no. 4, (1996): 481-486.

Wong. E, Jarvis.R, “Real Time Path Planning and Navigation for a Humanoid Robot in and Indoor Environment” *Proceedings of the 2004 IEEE Conference on Robotics, Automation and Mechatronics* (2004): 693-698.

Lim Chee Wang, Lim Ser Yong, Marcelo H, “Hybrid of Global Path Planning and Local Navigation implemental on a Mobile Robot in Indoor Environment” *Proceedings of the 2002 IEEE International Symposium on Intelligent Control* (2004): 821-826.