University of St. Thomas
Graduate Program in Software

SEIS-740, Spring 2014
Class Project

Real-Time Systems Demonstration in a Pinewood Derby Race Track:

Utilizing FreeRTOS, IR-Sensors, ADXL345 MEMS Accelerometer, UART Serial Port, and 7-Segment Display Modules in an ARM Cortex-M3 Microcontroller Based System

Table of Contents

Project Goal

The goal of this project is to demonstrate the use of the FreeRTOS operating system on a Cub-Scout style pinewood derby racetrack. The tasks scheduled would be responsible for real-time data collection, processing, and display of timing events located at known "gates" along the track. Eventually, parameters such as velocity & acceleration could be calculated from this data.

The goal is realized by a system that provides race times of derby cars at evenly spaced intervals along the pinewood derby track. This will be accomplished by having infrared sensors in each racing lane distributed along the track at even intervals to record the times from the start to the position of the IR sensor. Each time measurement will be displayed on a corresponding 7-segment digital display. At the end of the race, all times and calculations are saved to file (to allow for future analysis) and displayed on PC. Also, calculations & graphing/trending of metrics such as velocity and acceleration between the points on the track can be performed.

A critical portion of this project was the real-time processing & display of the captured timing events. The FreeRTOS tasks had to be scheduled/prioritized to ensure that each gate's time would be captured and displayed real-time as the race occurred. Also, running at a lower priority is a UART task that transmits the timing data over the RS232 serial port to a PC-based application (intended for further data processing & display).

Other components also used in the design were components such as the Analog Devices Inc. ADXL345 MEMS accelerometer used as the "starting gate" measurement device (triggers an interrupt that records the timer's value as the reference starting value), infrared (IR) sensors used to detect the presence of the car at each "gate" along the race track, and the UART RS232 serial port.

Real-Time Considerations

The challenge was to prioritize interrupts and tasks in such a way that *all* times are promptly displayed and accurate to the millisecond. Some of the critical timing constraints of this system are:

1. Accurately capturing all IR-sensor triggers real-time. It is very critical that NO car times (IR-sensor triggers) are missed!
2. The race times for each "gate" interval (for each IR-sensor measurement) must be displayed real-time and in the *proper order*.
   a. In other words, the times for the 1st gate must be displayed before the 2nd gate times.
   b. Task priorities must also be scheduled such that the 1st car to cross a gate is the 1st car to have its time displayed.
   c. NO gate times should be missed.
3. All timing measurements must be synchronized with the "starting gate" which will employ an *accelerometer* to detect the opening of the gate to start the race.
4. All final times must be accurate to 1msec. Thus, all interrupts must be handled and tasks scheduled & executed in time to record & display the results accurately and in the correct order.

Project Motivation

This project allowed the team members to learn to design a real-time system utilizing an RTOS running numerous tasks with specific scheduling requirements, as well as requiring numerous interrupts servicing multiple different devices. These devices include infrared (IR) sensors, an accelerometer, and digital displays. Also, the team members could learn much from each other as they each bring differing levels of experience & skills in various areas.

Theory of IR-Sensor Measurement

The infrared (IR) sensor used in this application was the Vishay TCND5000 reflective optical sensor. A picture and schematic of the device is given below.

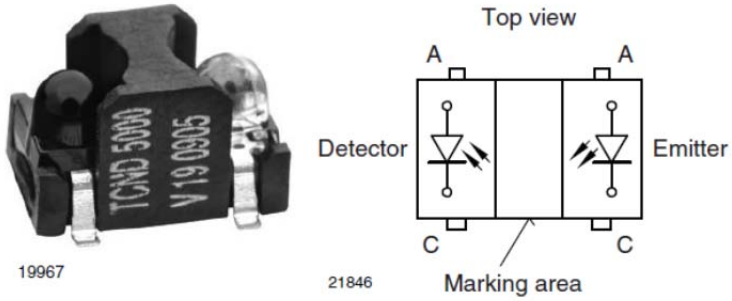


Figure 1: Vishay TCND5000 Reflective Optical Sensor

There are two (2) sub-devices in the TCND5000: the IR emitting LED and the IR Photodiode receiver. The emitter transmits the IR frequency light and the detector receives the reflected IR light.

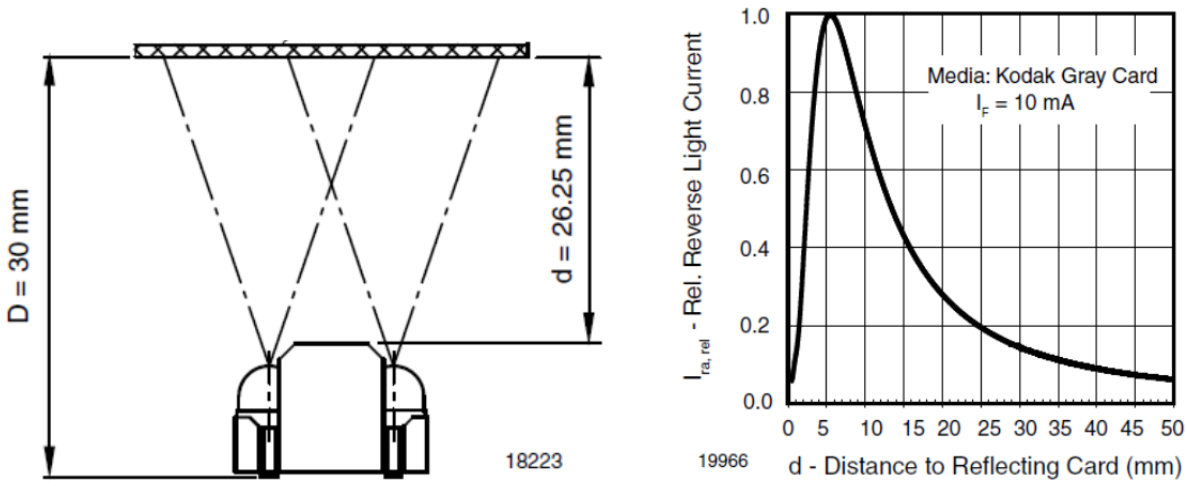


Figure 2: Test Setup of the TCND5000, and The Relative Reverse Light Current vs. Distance

The above figure shows that the peak distance for object detection is at approximately 6mm above the surface of the TCND5000. Also, it should be noted that the emissivity of the object being detected has a strong influence on the performance of the IR sensor.

System Design Overview

Refer to figure-3 for a basic *physical system diagram* and figure-4 for a *block diagram of the implemented µC system*.

Pinewood Derby Track Overview

This application was designed to have two (2) racing lanes in the pinewood derby track, where each lane has four (4) "gates" located along the track as shown in figure 3. At each gate there is an IR-sensor detecting the event of the car passing through the gate, as well as a 7-segment-display module to display the time of the event at that gate.

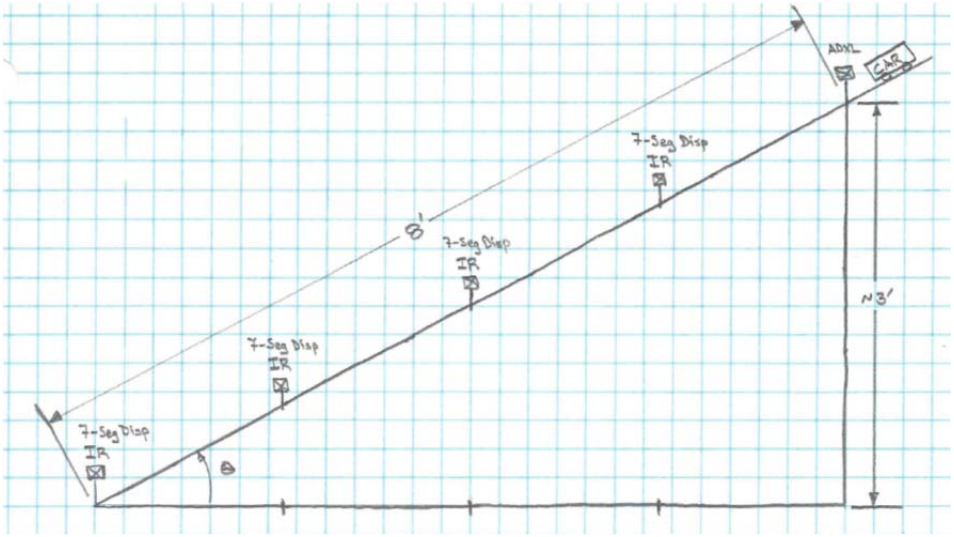


Figure 3: Main Physical System Diagram & Components

The physical system for this project:

1. Has two racing lanes on an angled track that is approximately 8-feet in length. Also note that the height of the track is adjustable from 3-feet up to 4½-feet in ½-foot increments.
2. There is a total of 8 IR-sensors (4 on each lane) that are evenly distributed at "gates" along the length of the track to capture timing intervals during the race.
3. There is an interrupt handler to capture the IR-sensor triggering events.
4. A MEMs accelerometer is used at the "starting gate" to capture the start of the race by sensing the opening of the gate.
5. There are RTOS tasks that calculate and display the times accompanying each "gate". The calculated times are output on both the 7-segment digital displays as well as the UART.
6. There are tasks to manage the LED's that will light up to indicate the start of the race as well as the winner's lane.
7. An end-of-race task will tabulate all time, calculate all velocities, accelerations, and store all results to file. This file can then be used to analyze the results of each run to determine areas of improvement for each car's design/performance.

Block Diagram

The high-level system design is shown in the following diagram.

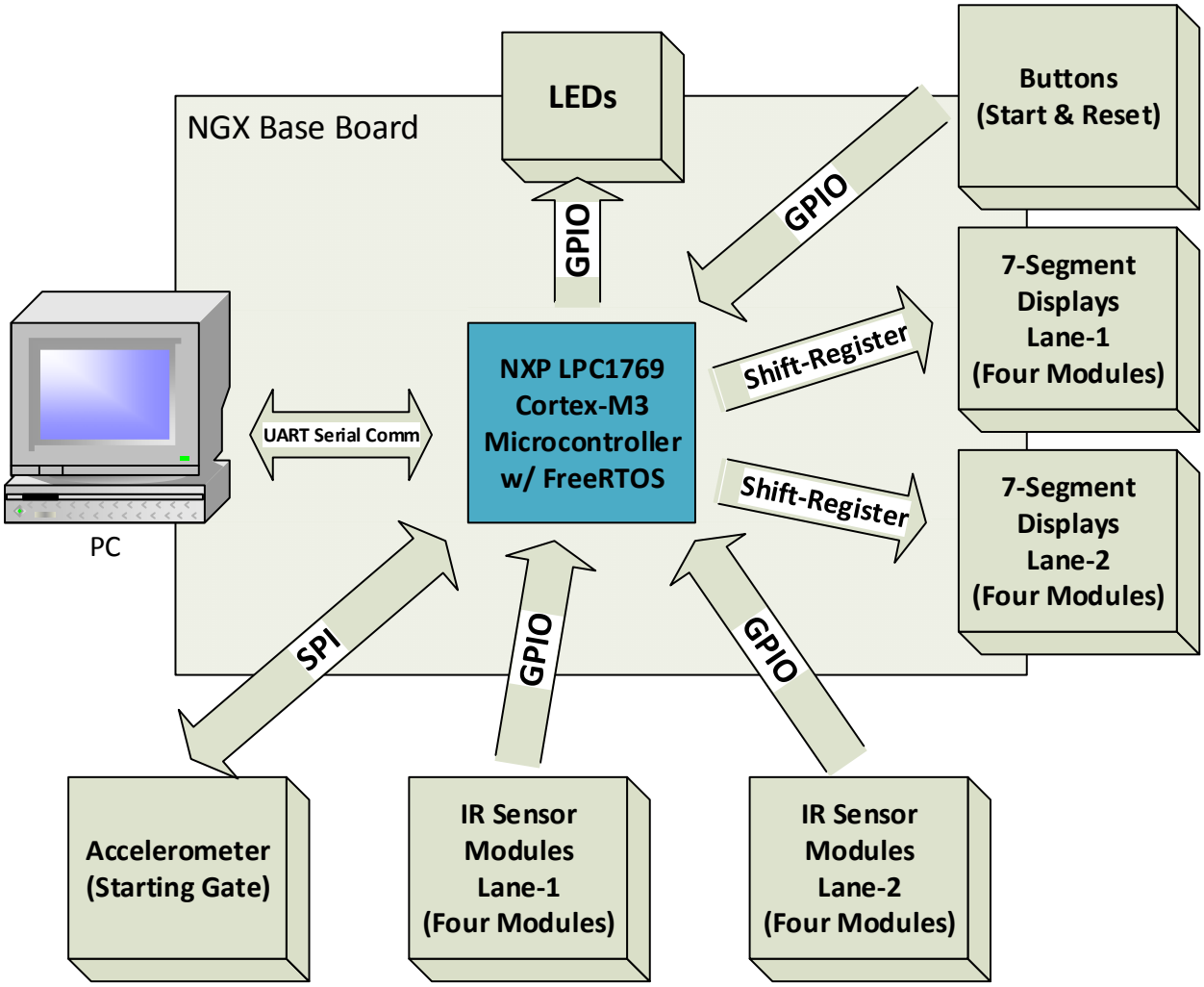


Figure 4: Block Diagram for Pinewood Derby Rack Track System

A summary of the peripherals of the µC system that were used:

Table 1: Summary of µC peripheral usage

µC component
Timer Interrupts (2)
General-Purpose I/O
External Interrupt Pins (3)
SPI bus module
UART module

System Software & Hardware Configuration

The configuration & details of the basic system components (shown in the block diagram) is given in this section.

InfraRed (IR) Sensor Circuit

The signal from the Vishay TCND5000 IR sensor had to be conditioned for input to the µC. The intent was to use the IR-sensor signal to trigger an external interrupt on the µC. Thus, the following circuit below was designed to compare the IR-sensor signal with a known reference voltage (set by a potentiometer). The comparator IC would toggle its output when the IR-sensor detected the presence of an object and its signal exceeded the threshold reference value. For this application, it was determined that a threshold value of 0.8V gave good IR-sensor event detection without too many false detection events.

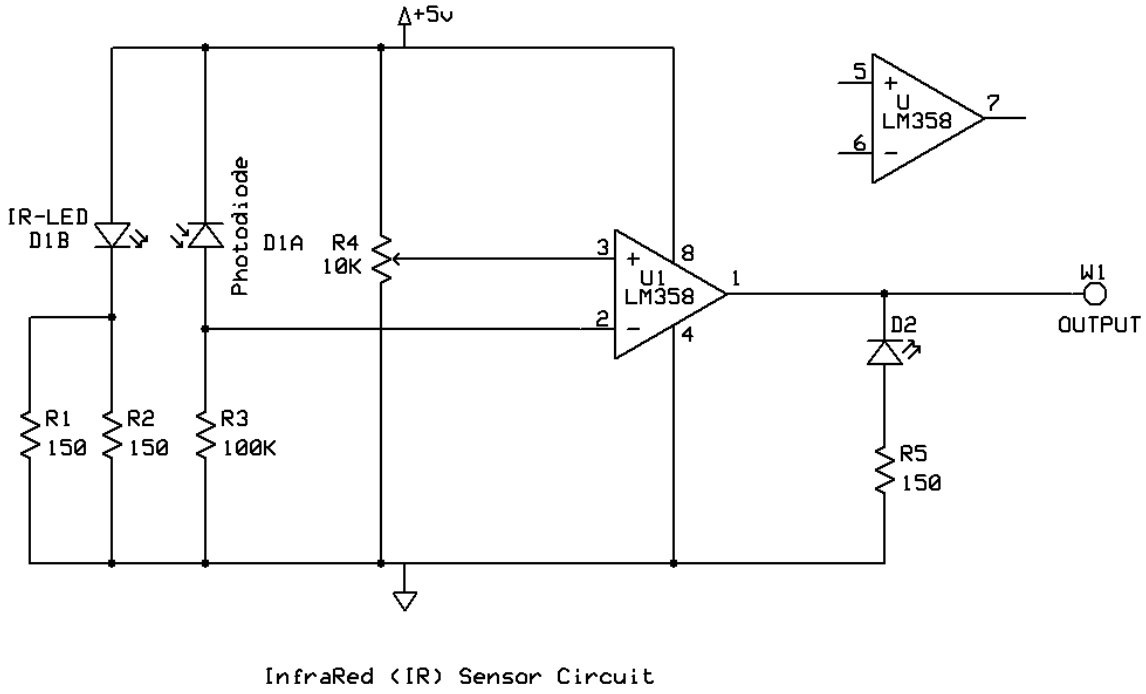


Figure 5: IR-Sensor Schematic

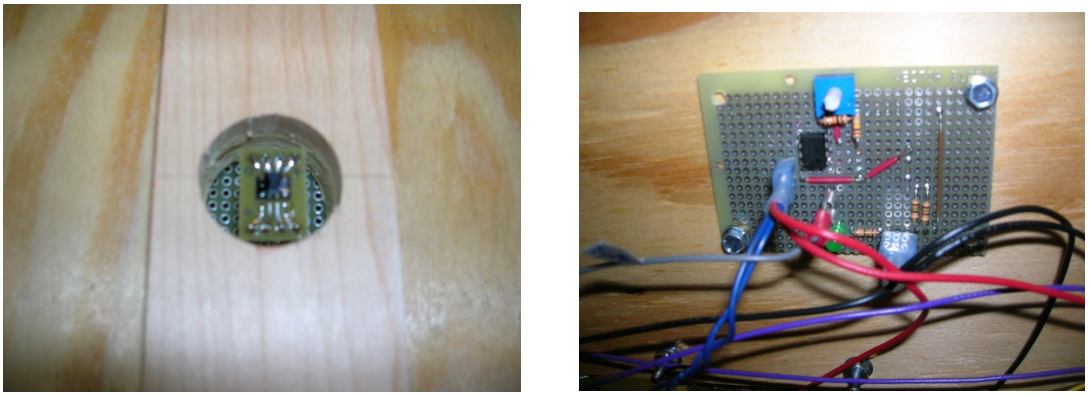


Figure 6: IR-Sensor Sensing Element and Circuit Board

Multiplexer (MUX) Circuit

Since the LPC1769 µC used in the implementation effectively only had three (3) external interrupt inputs, and the pinewood derby track system had nine (9) devices generating external interrupts, some multiplexers (MUX) had to be implemented. For this application, each race-track lane has four (4) IR sensor modules, so a 4-to-1 MUX (TI p/n SN74S153N) was used on each lane, where all four (4) IR sensor outputs are channeled through the MUX as shown in the MUX schematic.

This design also allowed for a method to eliminate IR-sensor "bounce". This was accomplished in the interrupt handler which first captured the timer value and then advanced the MUX to the next gate. By advancing the MUX in the interrupt handler, any more spurious IR-sensor events at that particular gate were effectively ignored.

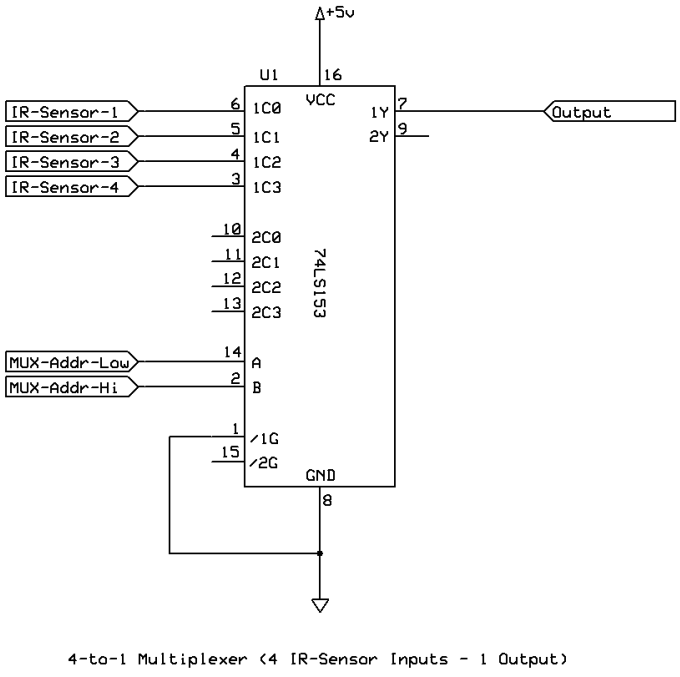


Figure 7: MUX Schematic

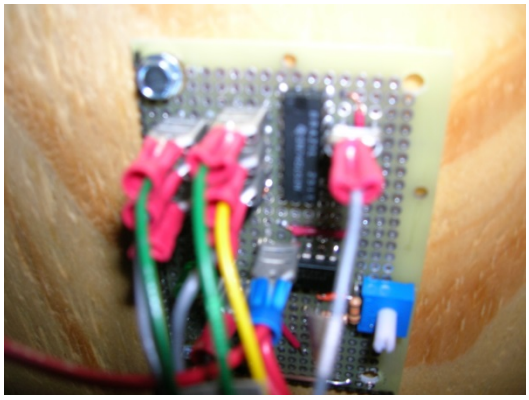


Figure 8: MUX Circuit Built Into One IR-Sensor Module

7-Segment Display Circuit

The 7-Segement-4-Digit digital display used was the LITE-ON p/n LTM-8328PKR-04. This display module (schematic shown below), had a shift-register interface. Thus, a custom firmware device driver was developed to communicate to this display.

The time displayed at each gate of the race track had a resolution of 1mSec.

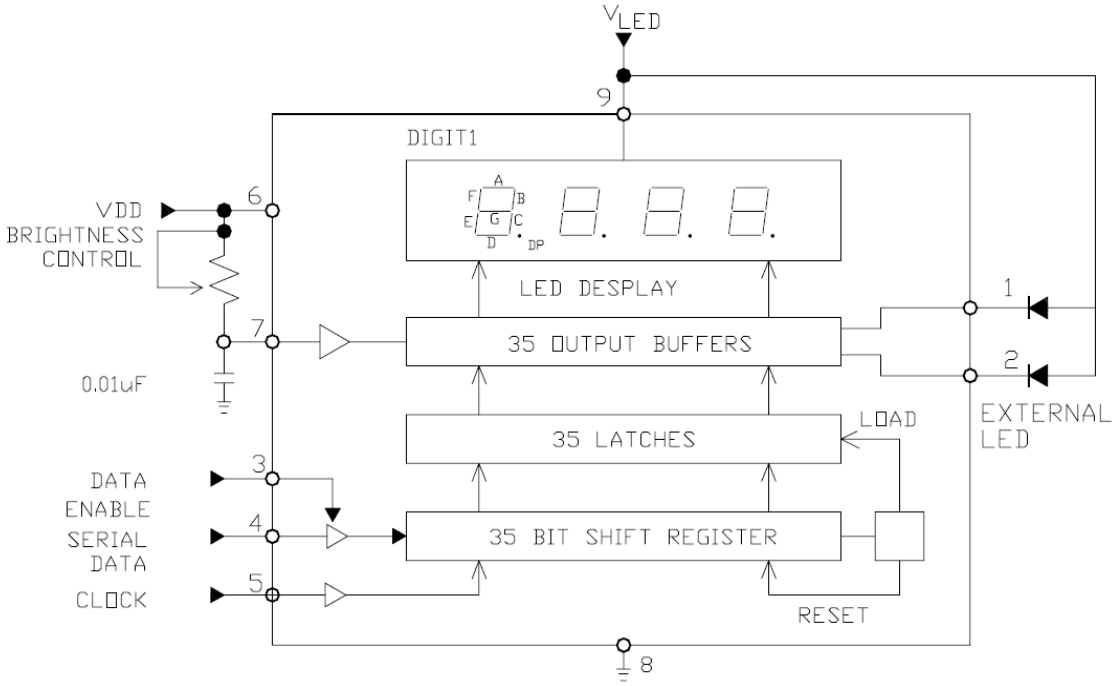
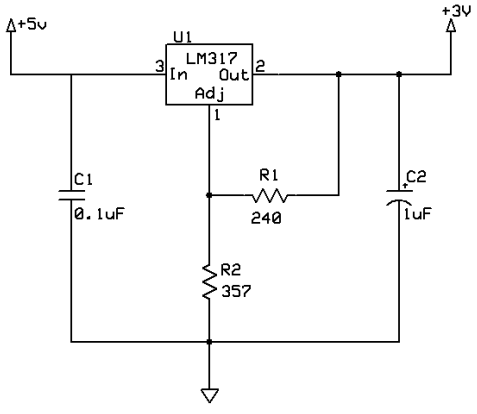


Figure 9: 7-Segment-Display Schematic

+3V Regulator

Since the main system power supply was +5VDC, and the LITE-ON digital display required both +5V for its logic and +3V for the LED drive, a custom +3VDC regulator circuit had to be designed into each display module. The LM317T adjustable regulator was used in the design, and the schematic for this +3V regulator is shown below.



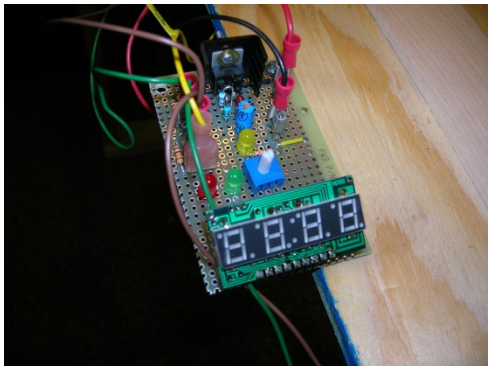


Figure 10: Display Module with Built In +3VDC Regulator

ADXL345 Configuration

In order to detect the "starting gate" event, the ADXL345 was configured to use its "Activity" mode. The configuration is summarized below.

1. The 4-wire SPI mode was used for communication with the microcontroller.
2. Activity mode was used to detect an acceleration event greater than the threshold of 4*g* on either the X or Y axes.
3. The Activity interrupt was used on pin INT2 to signal the microcontroller that the ADXL345 had detected a +4*g* acceleration event.
4. All tap & free-fall detection features were disabled.

Details showing the complete configuration are listed below in the Register Map & initialization values.

```
/* ********************************************
 * ADLX345 Register Map
 * ********************************************/

 #define REG_THRESH_TAP          0x1D   // Tap Threshold register
 #define REG_OFSX                0x1E   // X-axis offset
 #define REG_OFSY                0x1F   // Y-axis offset
 #define REG_OFSZ                0x20   // Z-axis offset
 #define REG_DUR                 0x21   // Tap Duration
 #define REG_LATENT              0x22   // Tap Latency
 #define REG_WINDOW              0x23   // Tap Window
 #define REG_THRESH_ACT          0x24   // Activity Threshold
 #define REG_THRESH_INACT        0x25   // Inactivity Threshold
 #define REG_TIME_INACT          0x26   // Inactivity Time
 #define REG_ACT_INACT_CTL       0x27   // Axis enable control for activity and
                                        //  inactivity detection
 #define REG_THRESH_FF           0x28   // Free-Fall threshold
 #define REG_TIME_FF             0x29   // Free-Fall time
 #define REG_TAP_AXES            0x2A   // Axis control for single tap/double tap
 #define REG_ACT_TAP_STATUS      0x2B   // Source of single tap/double tap. Read-Only
 #define REG_BW_RATE             0x2C   // Data Rate and power mode control
 #define REG_POWER_CTL           0x2D   // Power-saving features control
 #define REG_INT_ENABLE          0x2E   // Interrupt enable control
 #define REG_INT_MAP             0x2F   // Interrupt mapping control
 #define REG_INT_SOURCE          0x30   // Source of interrupts.  Read-Only
 #define REG_DATA_FORMAT         0x31   // Data format control
 #define REG_DATAX0              0x32   // X-Axis Data 0.
 #define REG_DATAX1              0x33   // X-Axis Data 1.
 #define REG_DATAY0              0x34   // Y-Axis Data 0.
 #define REG_DATAY1              0x35   // Y-Axis Data 1.
 #define REG_DATAZ0              0x36   // Z-Axis Data 0.
 #define REG_DATAZ1              0x37   // Z-Axis Data 1.
 #define REG_FIFO_CTL            0x38   // FIFO control
 #define REG_FIFO_STATUS         0x39   // FIFO status
```

```
/* ********************************************
 * ADXL345 Register Initialization Values
 * ********************************************/

 #define INIT_THRESH_TAP         0xFF   // Single/double tap disabled
 #define INIT_OFSX               0x00   // X-axis offset TBD
 #define INIT_OFSY               0x00   // Y-axis offset TBD
 #define INIT_OFSZ               0x00   // Z-axis offset TBD
 #define INIT_DUR                0x00   // Tap Duration (disabled)
 #define INIT_LATENT             0x00   // Tap Latency (disabled)
 #define INIT_WINDOW             0x00   // Tap Window (disabled)
 #define INIT_THRESH_ACT         0x40   // Activity Threshold 4g
 #define INIT_THRESH_INACT       0x7F   // Inactivity Threshold - N/A
 #define INIT_TIME_INACT         0x01   // Inactivity Time 1s
 #define INIT_ACT_INACT_CTL      0x00   // Disable activity and inactivity detection
 #define INIT_THRESH_FF          0x05   // Free-Fall threshold - desirable behavior
 #define INIT_TIME_FF            0x46   // Free-Fall time - desirable behavior
 #define INIT_TAP_AXES           0x00   // Single tap/double tap disabled
 #define INIT_ACT_TAP_STATUS     0x00   // Source of single tap/double tap. Read-Only
 #define INIT_BW_RATE            0x09   // Data Rate 50Hz - Sample rate 25Hz
 #define INIT_POWER_CTL          0x08   // Power-saving features control - init last?
 #define INIT_INT_ENABLE         0x10   // Interrupt enable Activity
 #define INIT_INT_MAP            0x10   // Interrupt mapping Activity -> interrupt
                                        //  pin INT2; all others -> pin INT1
 #define INIT_INT_SOURCE         0x00   // Source of interrupts.  Read-Only
 #define INIT_DATA_FORMAT        0x0C   // Data format control, SPI, Active High,
                                        //  Full Res, Justify, +/- 2g
 #define INIT_DATAX0             0x00   // X-Axis Data 0.  Read-Only
 #define INIT_DATAX1             0x00   // X-Axis Data 1.  Read-Only
 #define INIT_DATAY0             0x00   // Y-Axis Data 0.  Read-Only
 #define INIT_DATAY1             0x00   // Y-Axis Data 1.  Read-Only
 #define INIT_DATAZ0             0x00   // Z-Axis Data 0.  Read-Only
 #define INIT_DATAZ1             0x00   // Z-Axis Data 1.  Read-Only
 #define INIT_FIFO_CTL           0x88   // FIFO control. Stream mode & 8 samples
                                        //  Watermark
 #define INIT_FIFO_STATUS        0x00   // FIFO status  Read-Only
 #define INIT_TAP_SIGN           0x00   // Sign and source for single tap/double tap
```

The orientation of the ADXL345 as mounted on the end of the starting gate handle, and the ADXL345 was configured to detect an “Activity” event on either the X-axis or the Y-axis. The Z-axis was not utilized in this particular application. Since the X-axis and Y-axis for the ADXL345 have better sensitivity over the Z-axis, they were the two axes of the ADXL345 chosen for use in this application.

Microcontroller Configuration

The NXP LPC1769 microcontroller was utilized for this project. This device is an ARM 32-bit Cortex-M3 device. The choice of microcontroller was driven by two (2) main factors:

1. The project team desired a project that would utilize an ARM Cortex-M device (for the express purpose of learning more about this technology).
2. The availability of the LPC1769 evaluation board that was given to the project team in the Spring 2014 course by Chris Guarneri of NXP.

The LPC1769 was configured to run on the external 12MHz oscillator. From this main CLK signal, two (2) timer interrupts were utilized: one set to trigger every 0.1mSec (this was used by the IR-sensor interrupt handlers to capture the timing event of the cars), and a second timer interrupt set to trigger every 20µSec (used to drive the shift-registers of the 7-Segment-Display module boards).

The UART was configured for serial communication with the LPC1769 evaluation board through the NGX base board. The UART was configured for 57600 baud rate, 8 bit, 1 stop bit, no parity. During normal operation, the system output the measured & calculated gate times over the UART to a PC-based application program which in turn reads and displays these time values to a console.

The SPI port was utilized as the method of communication to the ADXL345. If was configured for 4-wire mode with a 100kHz clock rate and 8-bit data frame.

Power Supply

An external +5VDC power supply was used to supply the NGX base board as well as the logic side of the IR-sensor and 7-Segment-Display modules. In turn, the NGX base board has a +3.3VDC regulated supply to power all its digital circuitry such as the LPC1769 board and the ADXL345 accelerometer.

Software Architecture

FreeRTOS was utilized as the operating system for the system. It implemented three (3) tasks that scheduled the major tasks of:

1. Display Task – displays a given decimal value to the correct display
2. Reset Task – handles button presses and resets the system for another race
3. UART Serial Communications – sends output over the RS232 port to the PC-based application

Also critical to this application were the multiple interrupt handlers:

1. IR-Sensor trigger interrupt handler
2. ADXL345 interrupt handler
3. Timer interrupt handlers (2)
4. UART interrupt handler

Main Flow Chart

The overall main flowchart for the software architecture is shown in the figure below. The main() is responsible for most of the initialization of the peripherals, creating queues and semaphores, creating the tasks, and starting the FreeRTOS scheduler.

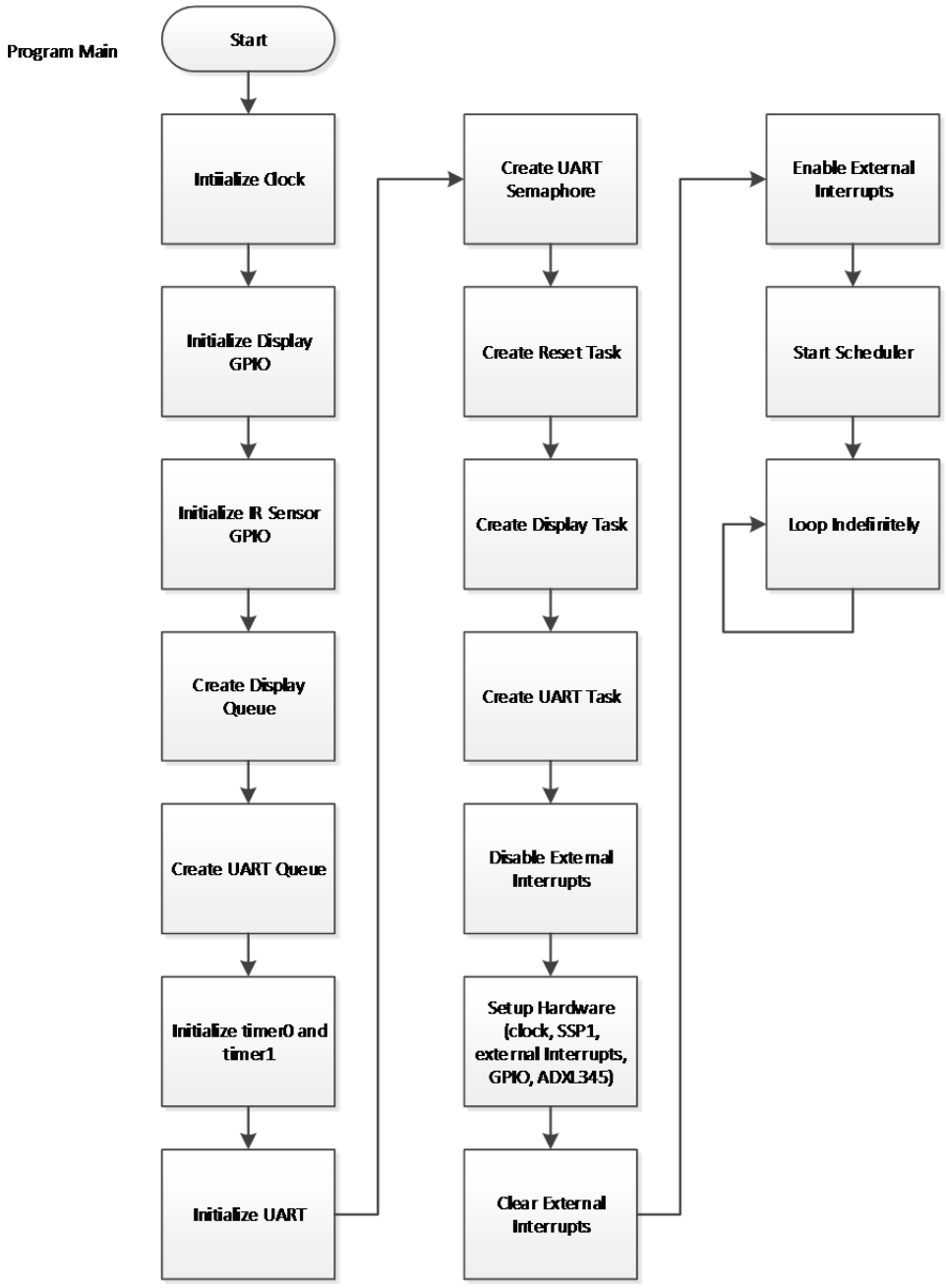


Figure 11: Software Architecture Main Flowchart

Display Task

The Display task is responsible for taking the raw timer values generated by the IR-sensor interrupt handlers, converting them to a time, and then displaying that time on both the 7-segment-displays and output on the UART serial communication port.

A FreeRTOS queue is used for process communication between the IR-sensor ISR and the Display task. In summary, the IR-sensor ISR's responsibility is to capture the raw timer value and IR-sensor ID# into a struct, and then place that struct onto the Display task's queue for later processing by the Display task itself.

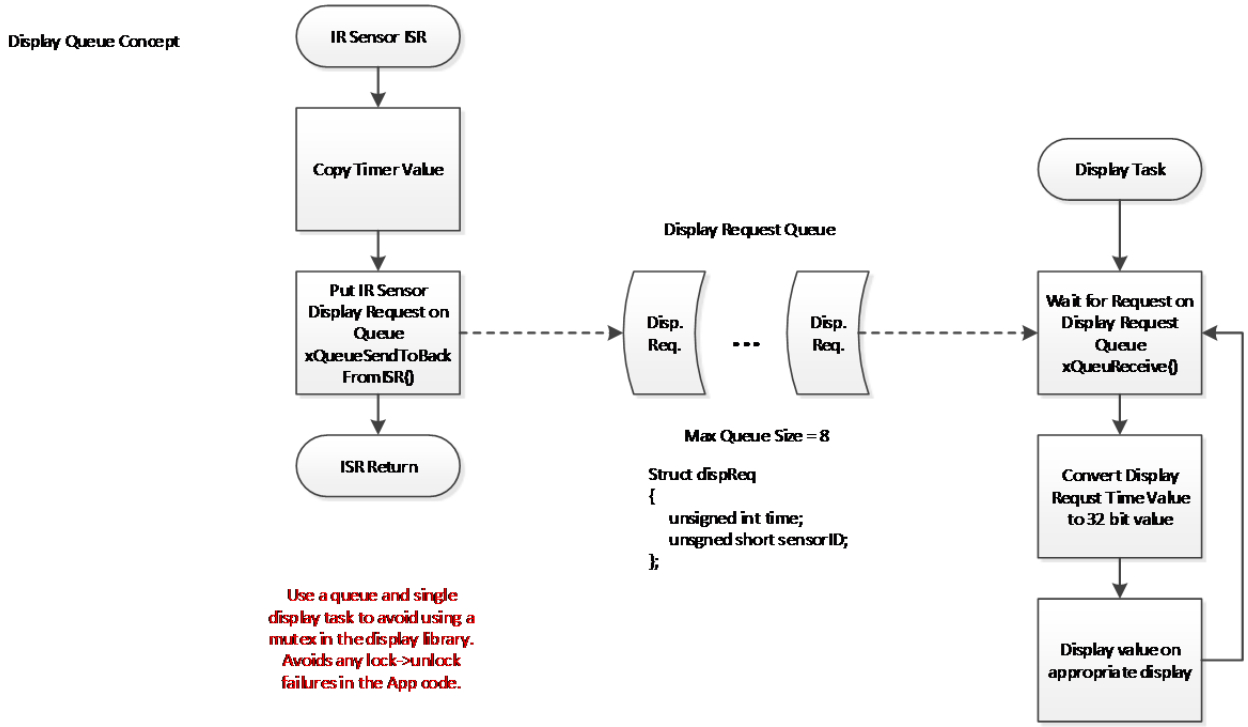


Figure 12: Software Architecture of Display Queue

The Display task then blocks until there is a value in the Display task queue, at which time it then takes the raw timer value and converts it to a time down to the 0.1msec resolution. Lastly, it displays that time on the 7-segment-display module that matches the ID# of the IR-sensor. The Display task priority was set to value 1 (lower the value, higher the priority).

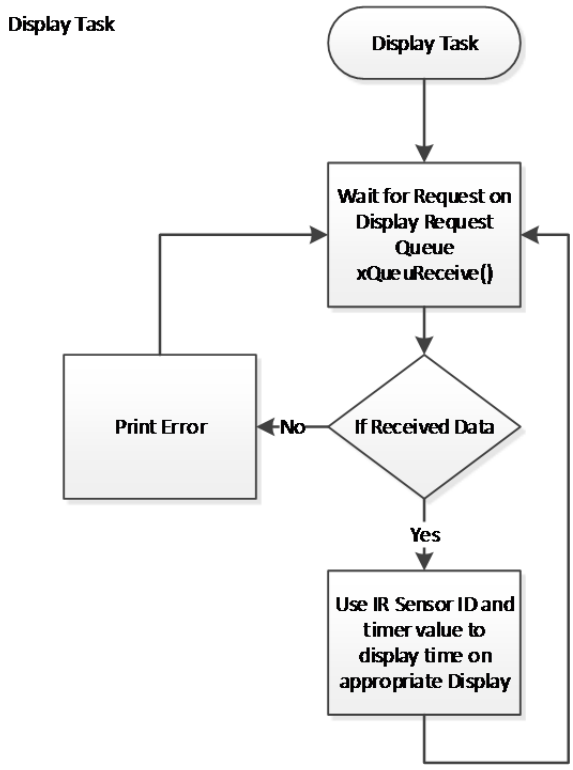


Figure 13: Software Architecture of Display Task

Reset Task

The Reset task is responsible for resetting the entire system to prepare it for another race. It monitors the external "reset" button input, and resets all appropriate system parameters upon detecting a button press state. The Reset task priority was set to value 1 (lower the value, higher the priority).

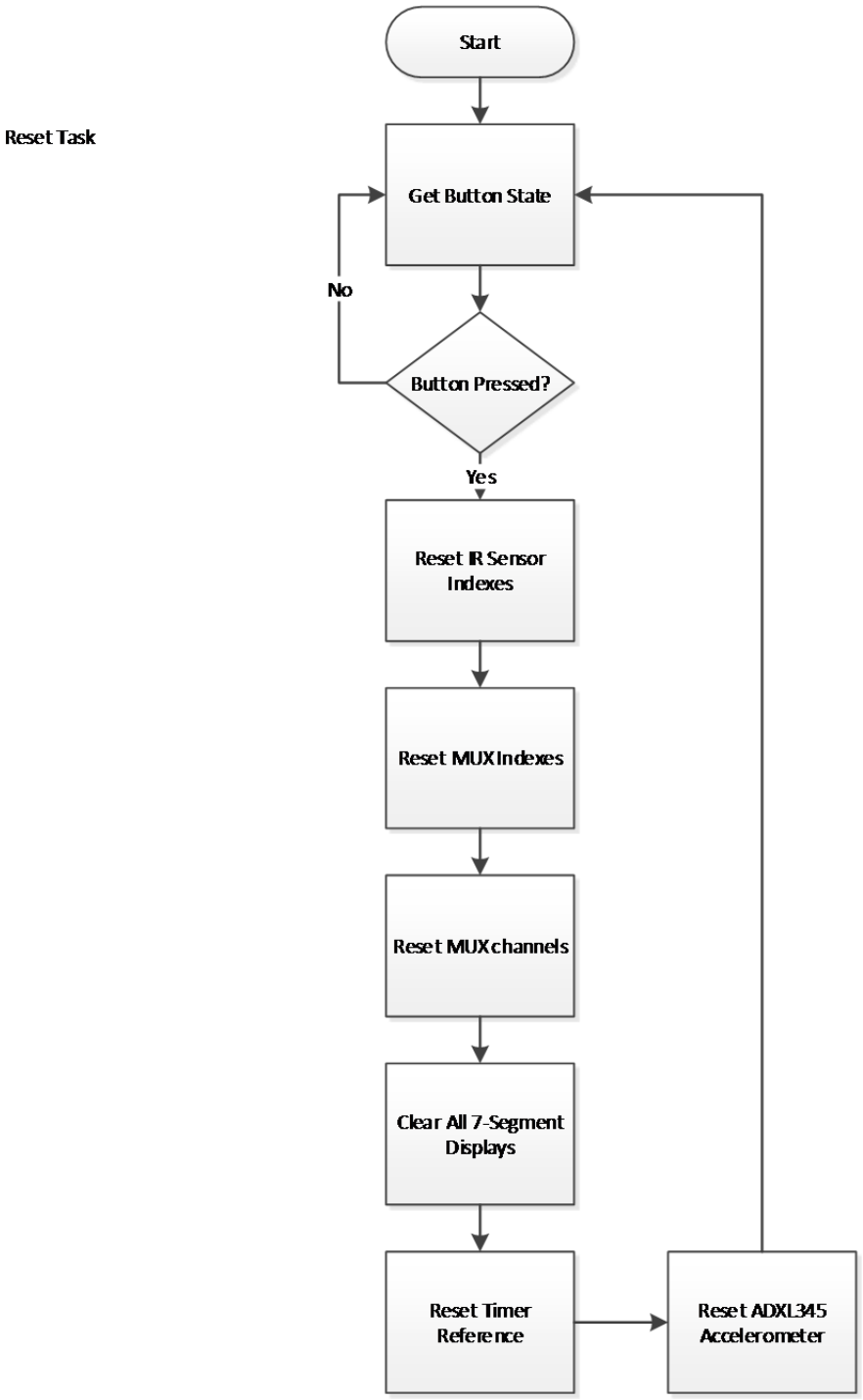


Figure 14: Software Architecture of Reset Task

UART Serial Communications Task

The UART task is responsible for the RS-232 serial communication between the system and a PC-based software application. The serial communication is mainly dedicated to outputting the measured race track timing values in tenths-of-milliseconds to the PC-based software application (which is displayed in run time). The UART task blocks until its queue has an item to output. Once the UART queue has an item, the task fires and outputs both the gate ID# and time value out the RS-232 communication port to the PC-based application.

The UART task priority was set to value 2 (lower the value, higher the priority), which is lower in priority than both the Display and Reset tasks. This is because the UART transmissions are not as timing critical as the Display and Reset tasks. Also, the UART task only uses the COM port resource, which is not used by the other tasks, so no timing anomalies should occur.

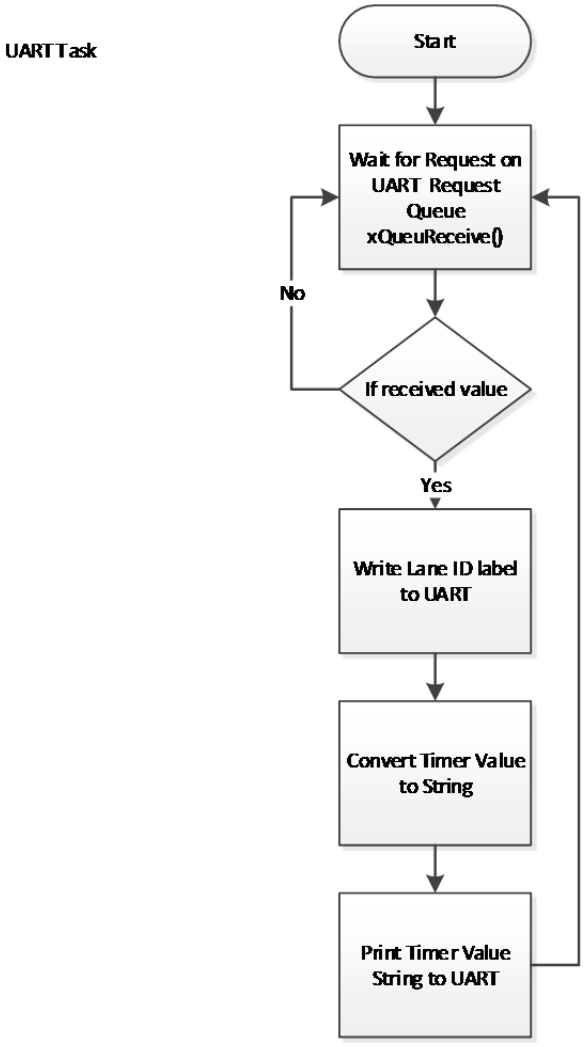


Figure 15: UART Serial Communications Task

The µC UART design can also accept commands from the PC-based application; however, for the existing design, the µC currently only reads, processes, and accepts any commands transmitted to it, but there is no functionality associated with the processing of the commands.

IR-Sensor Trigger Interrupt Handler

The IR-Sensor external interrupt handler captures an event from the IR-sensor module and then pushes both the gate ID# and the IR-Sensor Timer counter value to the Display task queue as well as advancing IR-Sensor and MUX indexes to the next gate. Later, when the Display task fires, it will pull these values from the queue and calculate a time (resolution of milliseconds) to be output on both the 7-segment-display modules and the UART.

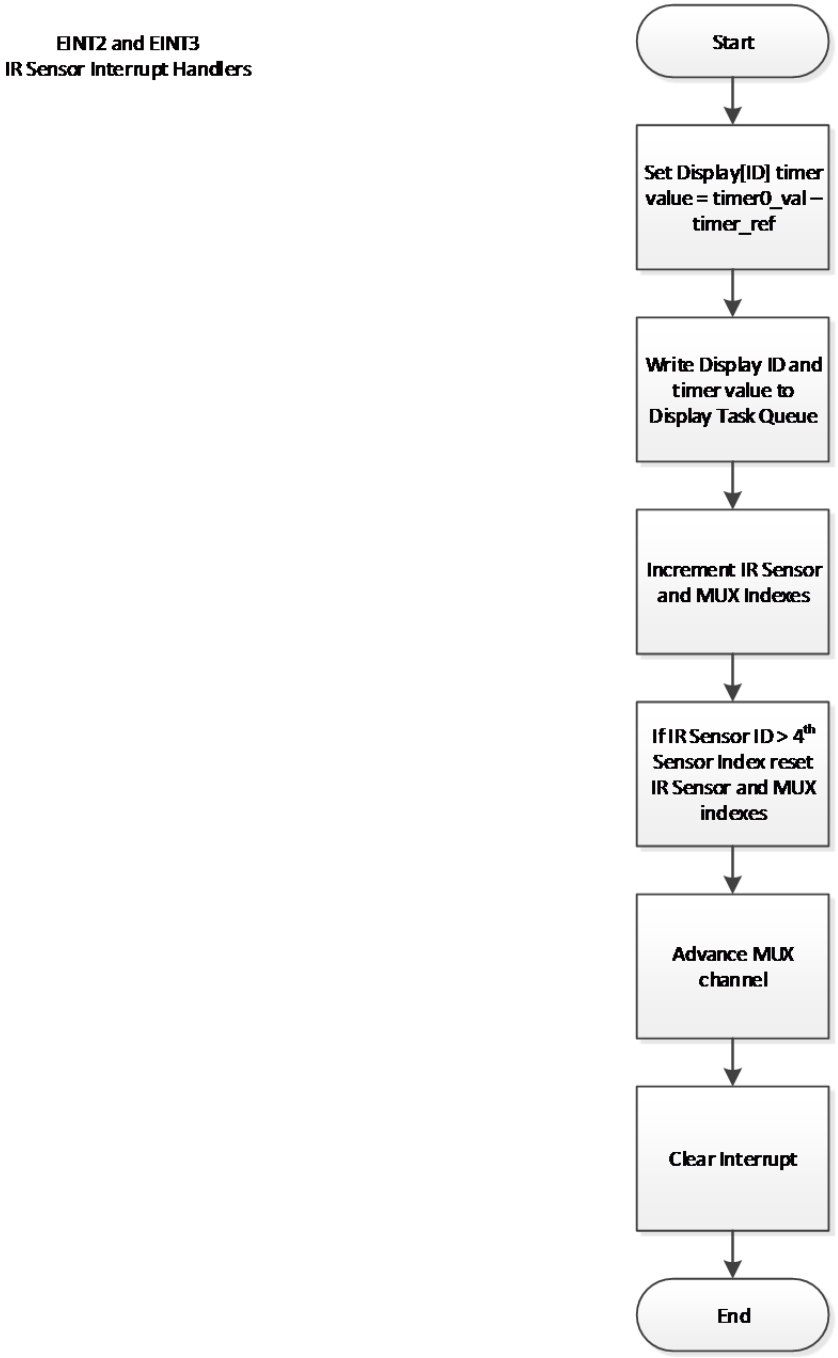


Figure 16: IR-Sensor Interrupt Handler

ADXL345 Activity Event Interrupt Handler

The ADXL345 "Activity" event interrupt handler triggers when the ADXL345 experiences an acceleration event greater than 4*g* on either the X-axis or the Y-axis. The interrupt handler sets a global timer reference value, which is later used as the starting time when calculating the times at each race track gate.

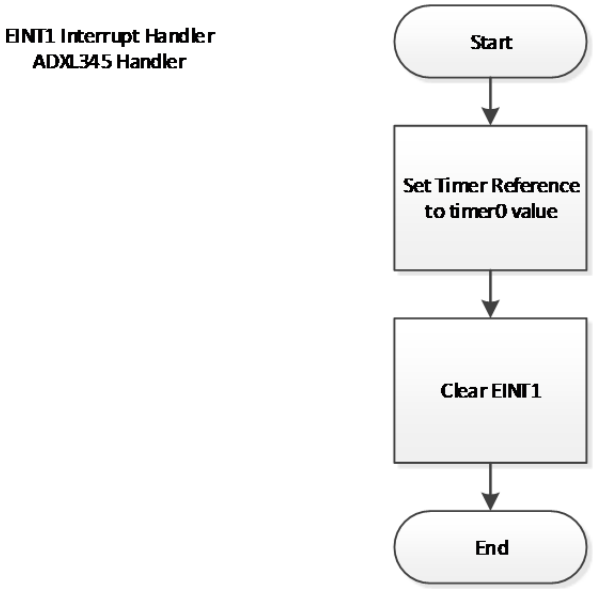


Figure 17: ADXL345 Interrupt Handler

IR-Sensor Timer Interrupt Handler

The timer used by the IR-sensor modules (for capturing the time of a race track gate event) is configured to a 0.1mSec interrupt rate. This timer interrupt handler simply increments a global timer value that is later used by the Display Task to calculate & display the time of the gate event (the race car passing the given gate).

7-Segment-Display Timer Interrupt Handler

The timer used by the 7-segment-display driver (for setting bit rate to the 7-segment shift-register) is configured to a 20µSec interrupt rate. This timer interrupt handler simply sets a global flag that flags the 7-segment-display driver that it can send another bit to the shift-register (if a bit is available).

Hardware/Software Constraints & System Analysis

LPC1769 Memory Map

Address range	General Use	Address range	Details and Description
0x0000 0000 to 0x1FFF FFFF	On-chip non-volatile memory	0x0000 0000 - 0x0007 FFFF	For devices with 512 kB of flash memory.
	On-chip SRAM	0x1000 0000 - 0x1000 7FFF	For devices with 32 kB of local SRAM.
	Boot ROM	0x1FFF 0000 - 0x1FFF 1FFF	8 kB Boot ROM with flash services.
0x2000 0000 to 0x3FFF FFFF	On-chip SRAM (typically used for peripheral data)	0x2007 C000 - 0x2007 FFFF	AHB SRAM - bank 0 (16 kB), present on devices with 32 kB or 64 kB of total SRAM.
		0x2008 0000 - 0x2008 3FFF	AHB SRAM - bank 1 (16 kB), present on devices with 64 kB of total SRAM.
	GPIO	0x2009 C000 - 0x2009 FFFF	GPIO.
0x4000 0000 to 0x5FFF FFFF	APB Peripherals	0x4000 0000 - 0x4007 FFFF	APB0 Peripherals, up to 32 peripheral blocks, 16 kB each.
		0x4008 0000 - 0x400F FFFF	APB1 Peripherals, up to 32 peripheral blocks, 16 kB each
	AHB peripherals	0x5000 0000 - 0x501F FFFF	DMA Controller, Ethernet interface, and USB interface.
0xE000 0000 to 0xE00F FFFF	Cortex-M3 Private Peripheral Bus	0xE000 0000 - 0xE00F FFFF	Cortex-M3 related functions, includes the NVIC and System Tick Timer.

For this project, the program will reside in FLASH, non-volatile memory will reside in EEPROM and volatile memory will reside in SRAM.

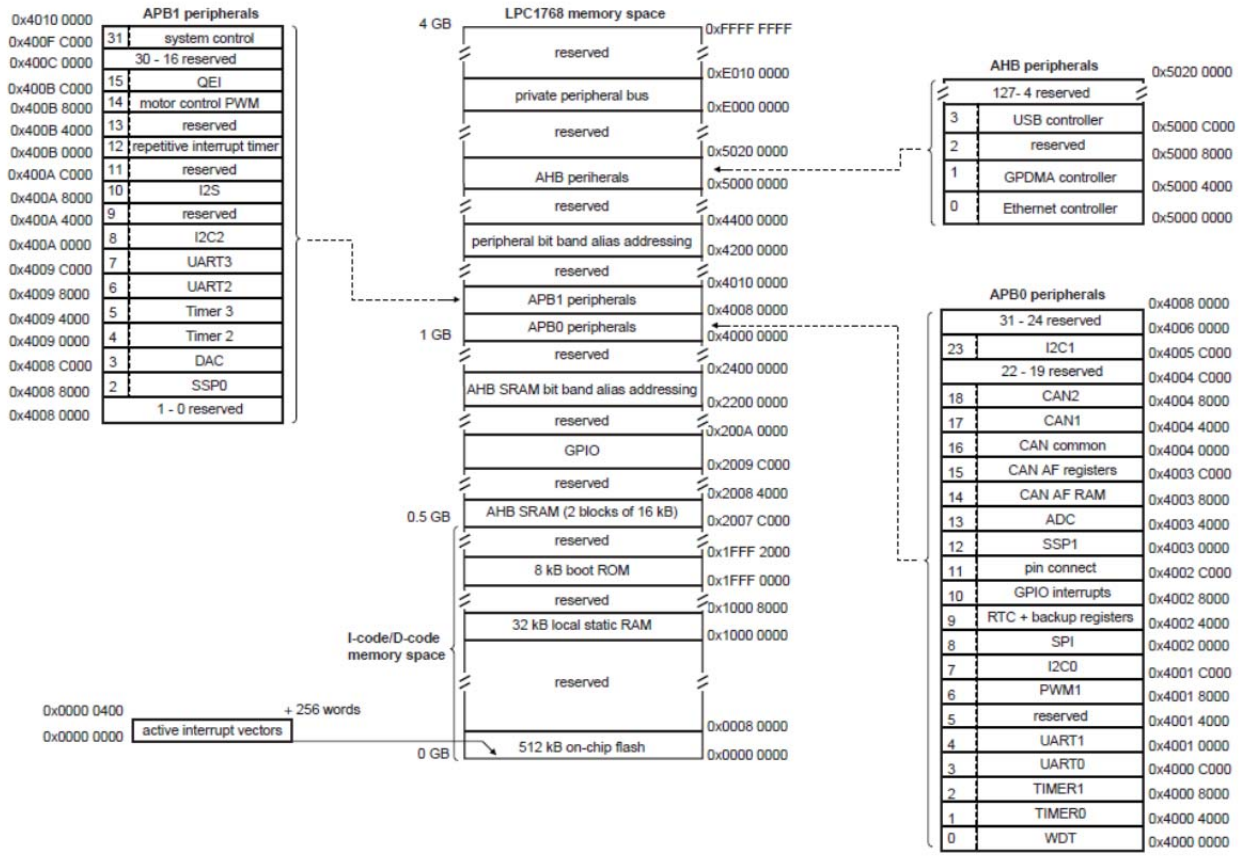


Figure 18: LPC1769 Memory Map

Processor Bandwidth

Definition: Bandwidth – the rate at which data is moved [Wolf 3rd-Ed p. 189].

In this application, there are two (2) timers, one running at 0.1msec and the other running at 20µsec. These are used by the IR-sensor event handlers and 7-segment-display drivers respectively. Outside of these timers are some asynchronous external interrupts (mainly handling the IR-sensor events), RTOS task & queue handling, and external data transfers over the UART bus. Thus, the µC bandwidth calculation can be limited to the timer interrupts, estimates of RTOS task handlers and UART communication.

The 7-segment-4-digit digital displays operate on a 20µSec (50kHz) timer interrupt. This timer routine can be assumed to be 10 instructions. Assuming a 4MHz bus frequency (with a given 12MHz clock), 1-instruction/cycle execution rate (for the Cortex-M3 3-stage pipeline) and a conservative estimate of 10 instructions to handle the interrupt, the total execution time would be:

$$(10 \text{ instructions})(1/(4\text{MHz})) = 2.5\mu\text{Sec}$$

Thus, its bandwidth is 2.5µSec executing every 20µSec (50kHz). This gives a 7-segment-4-digit digital display timer bandwidth estimate of 2.5µSec/20µSec = 12.5% of the time between IRQs.

Also, the IR-sensor timer operates on a 0.1mSec (10kHz) timer interrupt. This timer routine can also be assumed to be 10 instructions. Thus, its bandwidth is 2.5µSec executing every 0.1mSec. This gives an IR-sensor timer bandwidth estimate of 2.5µSec/0.1mSec = 2.5% of the time between IRQs.

Due to the short distances between the race track gates, the times measured between any two gates could be as low as 3msec. This equates to an approximate 333Hz IR-sensor interrupt rate. (Remember, the system was designed with a 0.1msec timer to be able to effectively capture the IR-sensor event for both tracks during this time.) Assuming the RTOS task & queue handlers conservatively have 2000 instructions,

(2000 instructions)(1/(4MHz)) = 500µSec => 2kHz rate

Thus, the RTOS task & queue bandwidth estimates to be 500µSec/3mSec = 16.7% µC utilization.

During UART serial communication, there is a mode of regular streaming of sensor data to a PC-based application. New sensor data is ready approximately every 3mSec. The routine to transmit sensor data during every Tx IRQ can be assumed to be 200 instructions. Thus, its bandwidth is 50µSec executing every 3mSec. This gives a UART bandwidth estimate of 1.7% of the time between IRQs.

The additional various asynchronous inputs (such as button presses) are very intermittent by nature, and will add negligible bandwidth.

Summing all of these calculations give a total peak µC bandwidth of:

12.5% + 2.5% + 16.7% + 1.7% = **33.4%**

Memory Usage

Utilizing the LCPXpresso IDE, the following memory usage estimates were made:

Program memory = ~27K bytes
Volatile memory = ~800-1200 bytes

Software Constraints by Hardware

1. The 7-segment-display modules have a shift-register interface, so a device driver was developed utilizing a 20µSec timer to control the timing & sequencing of the communication to the device.
2. Communication to the external ADXL345 is made over the SPI bus, so the software must have a driver and be configured to handle SPI communication to the external device.
3. The ADXL345 requires an external interrupt to trigger an interrupt to the microcontroller; thus, the µC must be able to be configured to accept an external interrupt via a pin.
4. Since there were effectively only three (3) external interrupt handlers available in the LPC1769, external multiplexers (MUXs) were required to MUX all the IR-sensor modules.

Hardware Constraints by Software

1. Software debug access must be presented over the JTAG interface.

Suggestions for Future Improvements

The first pass at this system had much success in many areas, but there's definitely room for future improvements / features to be added. Here's a brief summary of some of the major improvements that would add greatly to the project:

1. Additional tasks to manage the LED's that will light up to indicate the start of the race as well as the winner's lane. The LEDs are present in the hardware, just the RTOS tasks would need to be constructed.
2. An end-of-race task will tabulate all time, calculate all velocities, accelerations, and store all results to file. This file can then be used to analyze the results of each run to determine areas of improvement for each car's design/performance.
3. The physical system track is very crude. A much better, longer and smoother track could be built.
4. The mechanics of the starting gate for the current design is very crude. A more precise and equal mechanism would greatly benefit this system.
5. The ADXL345 itself may or may not be the best device for detecting the starting gate event. Depending on the upgrades to the mechanical features of the starting gate, a better starting detection device may be appropriate.
6. Calibration of the system:
    a. The accuracy of the timing values of the current system was not exactly measured. Obviously, for improved performance & accuracy, a system-wide calibration of the timing measurements would be necessary.
    b. Placement of the IR-sensor modules would be machined into exact locations. Currently, the IR-sensor modules were hand placed, and the tolerances of the placements could be easily detected with the 0.1msec timer resolution.
    c. In order to store any calibration coefficients, the on-board non-volatile data memory (emulated EEPROM) will need to be utilized.
7. The reflective IR-sensor chosen for this project had a severe limitation of being susceptible to the emissivity of the material it is trying to detect. In order to eliminate this vulnerability, a better IR sensor could be utilized. Perhaps one where the emitter and detector are separated into two components and detect on a line-of-sight instead of relying on reflection.
8. An expanded serial command set could be created to work with the PC-based program to give better functionality. Examples are:
    a. Commands to activate the starting gate
    b. Commands to reset the controller before a new race (instead of relying only on the external hardwired switch).

Resources

1. "Using the FreeRTOS Real Time Kernel: NXP LPC17xx Edition", 3rd Edition, by Richard Barry, Real Time Engineers, 2011.
2. Vishay IR Sensor: http://www.vishay.com/docs/83795/tcnd5000.pdf
3. LITE-ON 7-Segment-4-Digit Digital Display: http://www.digikey.com/product-search/en?x=0&y=0&lang=en&site=us&KeyWords=LTM-8328PKR-04
4. Analog Devices ADXL345 website: http://www.analog.com/en/mems-sensors/mems-inertial-sensors/adxl345/products/product.html
5. Analog Devices ADXL345 Break-Out Board website: http://www.analog.com/en/mems-sensors/mems-accelerometers/adxl345/products/eval-adxl345z/eb.html
6. NXP LPC1769 website: http://www.nxp.com/products/microcontrollers/cortex_m3/LPC1769FBD100.html
7. Analog Device App-Note AN-1057, "Using an Accelerometer for Inclination Sensing" by Christopher J. Fisher.
8. Analog Device App-Note AN-1077, "ADXL345 Quick Start Guide" by Tomoaki Tusuzki.
9. Analog Device App-Note AN-1025, "Utilization of the First-In-First-Out (FIFO) Buffer in Analog Devices Inc. Digital Accelerometers" by Christopher J. Fisher, Tomoaki Tsuzuki, and James Lee.
10. "Microsoft Visual C# 2012: Step by Step", by John Sharp