Wattbot:nt

**Current state of project**

**Jim Herd**

**March 2020**

Contents

[1 Introduction 3](#_Toc34209322)

[1.1 Background 3](#_Toc34209323)

[2 Wattbot 1 3](#_Toc34209324)

[2.1 Existing system 3](#_Toc34209325)

[3 Considerations 4](#_Toc34209326)

[3.1 General considerations 4](#_Toc34209327)

[3.2 Mechanics 4](#_Toc34209328)

[3.3 Power supplies 4](#_Toc34209329)

[3.4 Motion 5](#_Toc34209330)

[3.4.1 Brushed DC motors 5](#_Toc34209331)

[3.4.2 Low level motor control 5](#_Toc34209332)

[4 System architecture 7](#_Toc34209333)

[4.1 System overview 7](#_Toc34209334)

[4.2 Overview of FPGA subsystem 8](#_Toc34209335)

[4.2.1 FPGA device selection 9](#_Toc34209336)

[4.3 Overview of low-level microprocessor subsystem 10](#_Toc34209337)

[5 Interfaces 10](#_Toc34209338)

[5.1 I2C 10](#_Toc34209339)

[5.2 Serial/UART 11](#_Toc34209340)

[5.3 CANbus 11](#_Toc34209341)

[5.4 SPI 11](#_Toc34209342)

[5.5 Digital I/O 11](#_Toc34209343)

[5.6 Analogue I/O 11](#_Toc34209344)

[6 Progress (March 2020) 12](#_Toc34209345)

[6.1 FPGA 12](#_Toc34209346)

[6.2 Low-level microprocessor 13](#_Toc34209347)

[6.3 High-level processor 13](#_Toc34209348)

[6.4 Vehicle design 13](#_Toc34209349)

[7 Proposal 14](#_Toc34209350)

[Appendix A: 15](#_Toc34209351)

Figures

[Figure 1 System structure 7](file:///C:\jth\HW_new_robot\Wattbot_nt_design_review.docx#_Toc33737347)

[Figure 2 General FPGA structure 9](#_Toc33737348)

# 1 Introduction

## 1.1 Background

The original Wattbot I undergraduate robot is now over 10 years old. The robust nature of its construction has resulted in a reliable small robot vehicle platform that has stood the test of time and students. However, the underlying technology, based on 8-bit PIC processors is now dated and lacks development potential as a platform for robot experimentation as well as an introduction to programming. This document will outline a replacement system using up-to-date technology to allow the unit to be used in a wider range of applications. This new robot will be designated – Wattbot:*nt*.

# 2 Wattbot 1

## 2.1 Existing system

Wattbot I was designed as an interesting and engaging way to teach C programming and, as an extra, to allow students to experiment with simple robot vehicle concepts. The aspects that made it successful are

* Software
  + Simple processor with free C compiler
  + IDE to develop and download code to robot
  + Supplied basic C software that forms the basis of extensions. These are developed through a series of lab experiments.
* Mechanics
  + Aluminum chassis that could withstand student miss-handling
  + Robust access to processor I/O through high quality screw terminals
* Electronics
  + Cheap, easily replaceable DIL processor chips
  + Robust motors and H-bridge controllers
  + On-board NiMh battery and associated charging circuitry
  + Low complexity 2-line display
* Usage
  + Free development environment available allowing students to use their own computers
  + Sufficient number of units to allow one robot to be used by two students

The aim of this development is to keep the positive aspects of Wattbot 1 but implement them with current technology.

# 3 Considerations

## 3.1 General considerations

The original Wattbot 1 specification called for the design of a small mobile robot vehicle to support practical work for undergraduate students on a joint Electronic/Mechanical Engineering course. In time, its use was expanded to support the teaching of C programming to Electronic Engineering undergraduates. Although the design of a new robot is primarily seen as a replacement for Wattbot 1, it would be useful to see the core motion/sensor package as being able to be applied to a range of robot vehicles, particularly in the research environment. For example, a vehicle that could be manufactured in volume and at low cost could form the basis of research into swarm structures.

|  |  |
| --- | --- |
| Proposal | Details |
| P\_A1 | Replacement for Wattbot 1 to be refered to as Wattbot:*nt.* |
| P\_A2 | Design the motion and sensor package with application in a range of vehicles from small to large. |
| P\_A3 | Design an instance (Wattbot:*nt* ) of the motion/sensor platform to provide a replacement for Wattbot 1 |

## 3.2 Mechanics

The mechanical chassis of Wattbot 1 (170mm \* 120mm \* 38mm) seems to be a good compromise in size and form factor. It would seem sensible to leave this unchanged. The M3 slot pattern could be changed and we could make it mandatory that the chassis should not be bent by the user and that plastic M3 screws and nuts should be used to prevent damage to the aluminum. The Wattbot 1 electronic was based on a stack system. This proved to be convenient and the idea would be retained for Wattbot:*nt*, but would not retain backward compatibility.

|  |  |
| --- | --- |
| Proposal | Details |
| P\_B1 | Retain Wattbot I mechanical form factor for Wattbot:*nt*. |
| P\_B2 | Electronic system to be built as a stack of interconnected boards. |

## 3.3 Power supplies

The original Wattbot I used NiMH batteries that were regulated to both 5v and 6v. The 5v supplied the system electronics and the 6v for the servo motors. The electronics industry has moved to 3.3v, therefore ALL electronics should work at 3.3v. Working with a mixed 5v/3.3v system is likely to give issues. However, it may be necessary to retain a small 5v supply, e.g. the RS485 bus for Dynamixel servos requires 5v.

Battery technology has advanced therefore it would be sensible to move from NiMH to Lithium-ion. This will probably require enhanced monitoring of the charge/use cycle to prevent battery damage. However, there are packaged solutions that could take care of these issues. USB powerbank units could provide a possible solution.

|  |  |
| --- | --- |
| Proposal | Details |
| P\_C1 | Plan use of Lithium-ion battery technology for Wattbot:*nt* with integrated charging and protection features. |
| P\_C2 | Primary electronics package should use 3.3v supply. Default to using 3.3v devices. |
| P\_C3 | Provide 5v low current secondary supply |
| P\_C4 | Provide regulated power supply (5v or 6v) for motors for Wattbot:*nt*. |

## 3.4 Motion

The motion system is a critical part of the robot. The Wattbot I system used modified servos in an open-loop mode. The system was based on individual motors driving the two rear wheels. A single front ball bearing unit provided the required stable point. There was no measurement of shaft angular position or rotational velocity. Apart from the lack of feedback the arrangement was reliable and compact. The original design used a direct connection between the motors and the battery. This made open-loop correction very difficult as the battery voltage changed over time. A mid-life update was made to supply the motors with a regulated 6v which made open-loop correction experiments much more reliable.

### 3.4.1 Brushed DC motors

The continued use of brushed DC motors for the new design seems a reasonable option. Given our experience of Wattbot I, the use of modified RC servo units would be a good starting point for the new design for the undergraduate version of the new robot. However, it would seem sensible to allow the system to be used with a wider range of motor/encoder/gearbox options.

To provide position/speed feedback it is proposed to customize the standard RC motor servos with non-contact quadrature encoders based on 2-D Hall-effect sensors. This technology is quite mature but some mechanical work will be needed to design the device into the RC servo housing.

|  |  |
| --- | --- |
| Proposal | Details |
| P\_D1 | Continue to use modified RC servos for basic undergraduate Wattbot:*nt* vehicle. |
| P\_D2 | Allow system to interface to higher power H-bridge chips/boards for larger vehicles. |
| P\_D3 | Electronic system to be built as a stack of interconnected boards. No requirement for backward compatibility. Basic board would include FPGA and managing microprocessor. |
| P\_D4 | Add a quadrature sensor to an RC servo. Unit to be based on a modern non-contact Hall-effect sensor. |

### 3.4.2 Low level motor control

Wattbot 1 had open-loop control of two modified RC servo motors driven by a PWM signal from a PIC microcontroller. The microcontroller generated the PWM signals that were sent to two automotive grade H-bridge devices. Although this setup proved very reliable, it had a number of disadvantages

* No feedback of motor position and speed
* limited to two motor control channels
* Even if feedback had been available, the compute power of the microcontroller would have been insufficient to run reasonable quality loop control algorithms.

To provide flexible motion control features for more than 2 motor channels it is proposed to split the PWM bit generation and encoder decoding into its own FPGA and have a more powerful microcontroller handle loop control and interfacing to a higher level computer such as a Raspberry Pi.

|  |  |
| --- | --- |
| Proposal | Details |
| P\_E1 | Motion control system to have   * motor PWM channels   + Minimum 4 channels   + configurable to differing types of H-bridge chips. * quadrature encoder channels   + Minimum 4 channels   + measurement of position and velocity.   + simulated encoder to verify channel * RC servo channels   + Minimum 8 channels   + allow full range of possible waveforms |
| P\_E2 | Implement the following bit manipulation activities in an FPGA   * PWM signal generation * angular encoder decoding * angular position and velocity measurement * RC servo signal generation |
| P\_E3 | Implement the following activities in a low level microprocessor   * High speed Interface to FPGA   + High speed parallel link     - > 1 Mbyte/second * Interface to higher level processor   + Basic interface through high speed UART link,     - > 0.5Mbits/second   + CAN interface     - 1Mbit/second * PID type control of each motor   + > 100Hz on all control channels * monitoring of motion control system * manage I2C sensors   + 5v and 3.3v busses * manage Dynamixel servo bus   + 3-pin and 4-pin busses * Real-time clock   + battery backup |
| P\_E4 | Implement high-level features in attached processor   * e.g. Raspberry Pi (3B, W, etc) * C and Python access libraries * ROS access to vehicle facilities * System test software |

# 4 System architecture

## 4.1 System overview

Based on the proposals outlined in the section 3 the following diagram gives an overview of the system and the general connections between the various sub-systems.

CAN bus

FPGA

H-bridge

(opt)

H-bridge

(opt)

H-bridge

(opt)

H-bridge

(opt)

Encoders

RC servo

RC servo

RC servo

RC servo

RC servo

RC servo

RC servo

RC servo

Low level

Microprocessor

High speed bus

Serial comms channel

High level

Control computer

e.g. Raspberry Pi

Real-time

clock

Dynamixel

servo bus

I2C

sensors

Motion

Control

System

H-bridge

(opt)

H-bridge

(opt)

H-bridge

(opt)

H-bridge

(opt)

External

H-bridge

Internal

H-bridge

DC motors

Figure 1 System structure

The general allocation of tasks is

|  |  |
| --- | --- |
| FPGA | * Activities requiring timed bit manipulation of multiple external input/output signals. |
| Low level microprocessor | * Interpretation of high level commands. * Executing feedback control loops * Manage CAN bus. * Manage I2C devices * Access battery backup real time clock (RTC) |
| High level computer | * Implementation of task level robotic actions.   + e.g.Raw Python/C++, ROS * wireless connection to local/wide area networks |

## 4.2 Overview of FPGA subsystem

The FPGA is essentially an interface subsystem for the low level microprocessor. Basing the FPGA on an register addressable system would seem an appropriate strategy which would allow new features to be added without significant disruption to existing features. Allocating a block of registers to each feature would allow the interaction between the low level microprocessor and the FPGA to be conducted by reading and writing specific registers.

The following decisions were made

1. Subsystem registers in the FPGA would be 32-bits.
   * implies an internal 32-bit bus
2. Register address would be 8 bits
   * Giving a block of 256 configuration and data registers
   * 256 seems a reasonable number with room for expansion. Current Systemverilog software uses 34 addresses for 2 PWM channels, 2 QE channels, and 8 RC servos.
3. High speed bus between FPGA and low level microprocessor is limited by the number of pins on the microprocessor.
   * Use 8-bit bidirectional bus and send/receive data in byte packets.

The general structure within the FPGA

Figure 2 General FPGA structure

µP

FPGA

other functions

using 32-bit bus

8-bit bus

32-bit bus

Low level

microprocessor

Digital I/O

FSM = Finite State Machine

### 4.2.1 FPGA device selection

There are a few of manufacturers that target low-end, low-cost FPGA devices that have less than 50,000 logic elements (LEs). This application is unlikely to need more than 10,000 LEs (current design uses 4000 LEs). With the authors experience with Altera/Intel devices and design software, the relatively modern MAX 10 series was selected. Decisions are as follows

|  |  |  |
| --- | --- | --- |
| D\_A1 | FPGA family | Intel/Altera MAX 10 |
| D\_A2 | Device | MAX10M08SAE144C8GES   * 8k LEs * 144 pin QFP (quad flat pack : 20mm\*20mm * 3.3v power supply * £16.88 (Digikey 4/3/20) |
| D\_A3 | Design Environment | Quartus Prime 18.x   * Free version |
| D\_A4 | Development software | SystemVerilog   * integrated into Quartus Prime * Is an update to Verilog that adds features that make code more readable (eg for definition of Finite State Machines) |
| D\_A5 | Simulation software | ModelSim - INTEL FPGA STARTER EDITION 10.5b   * Separate program * Free program |

## 4.3 Overview of low-level microprocessor subsystem

There are some features that would be difficult to implement in the FPGA but need to be hidden from the general user. A good example would be the implementation of feedback control of motor position and speed. A floating point microprocessor able to run a small real-time operating system should provide the necessary compute capability for this application.

Design decisions as follows

|  |  |  |
| --- | --- | --- |
| D\_B1 | Family | ST Microelectronics STM32 |
| D\_B2 | Device | STM32F303RET6   * 3.3v 72MHz Arm Cortex M4 * FPU processor * 512Kb Flash + 80Kb RAM * 64-pin flatpack - LQFP64 * Real-Time clock * CANbus, I2C, SPI, serial interfaces * ADC + DAC * Longevity guarantee * £5.39 (Digikey 4/3/20) |
| D\_B3 | Development environment | * Mbed OS 5 (online or offline) |
| D\_B4 | Software | * C/C++ * define a simple packet API for connections to FPGA and high-level computer |
| D\_B5 | RTOS | * Mbed RTOS (based on RTX) |

# 5 Interfaces

A standard set of interfaces will be provided by the motion control system. These include – I2C, UART, CAN and SPI. Although some of these could be provided by the FPGA it would give a simpler and more flexible system if these were handled by the low-level microprocessor (STM32 device). If absolutely needed, then some SPI and UART interfaces could be built into the FPGA. The structure of the FPGA system would make this doable. This could be addressed once a development system had been tested and evaluated.

## 5.1 I2C

I2C is a common low data rate bus for many useful sensors. the STM32 processor has enough capability to manage these devices and to do some initial processing before sending the data to the high level computer. For example, filtering could be applied to raw data.

An issue that needs to be considered is that the industry is currently in a transition between 5V and 3.3V as the primary system supply voltage. It is envisaged that a development board would have both a 5V and 3.3V based I2C busses. Devices such as the PCA9306 can provide a bridge between a 3.3v bus and a 5V bus.

## 5.2 Serial/UART

Less and less sensor devices use a serial UART interface, therefore there is little need for multiple UART channels. However, it is proposed that the communication between the low-level processor and the high-level processor (e.g. RaspPi) would use a UART channel running at as high a rate as possible (e.g. 1Mbit/sec).

Another use of a UART channel would be for a Bluetooth connection. However, this would only connect into the high-level processor and not be part of the motion control board.

## 5.3 CANbus

The CAN bus is a high reliability bus targeted at noisy environments such as automotive applications running at up to 1Mbit per second. Although this may have limited use in the undergraduate robot, it could be useful when the board is applied to larger vehicles where a network of processors may be required to run the low level functions. Higher level networking would not use CAN but would use standard TCP/UDP networking that would be managed by the higher level processor/s.

## 5.4 SPI

Some higher rate sensors use the SPI bus which uses a simple bit protocol and would be easy to implement on an FPGA. SPI is also available on the STM32 low-level processor. For the initial development board, a SPI bus from the STM32 will be brought to header pins and will not be implemented in the FPGA.

## 5.5 Digital I/O

Spare pins on both the FPGA and the STM32 processor will be bought to header pins for possible use.

## 5.6 Analogue I/O

The following analogue I/O facilities are available

|  |  |  |
| --- | --- | --- |
| STM32F303RE | Input | * 4-input analogue multiplexor feeding a 12-bit SAR A/D converter |
| Output | * 2 channel 12-bit D/A converter. |
| Max10 FPGA | Input | * 8-input analogue multiplexor feeding a 12-bit SAR A/D converter |

# 6 Progress (March 2020)

Notes

* Snapshot of system as of March 2020
* Testing levels
  + A = Basic testing
  + B = Exercised through usage
  + C = Soak testing of all features
* FSM = Finite State Machine
* QE = Quadrature Encoder

## 6.1 FPGA

|  |  |  |  |
| --- | --- | --- | --- |
| **Topic** | **Features** | **Coded** | **Tested** |
| Device selection | Intel/Altera MAX10M08SAE144   * 3.3v, 50MHz clock (50nS period) * 8000 Logic Elements (LEs) * 144 pin quad flat pack |  |  |
|  |  |  |  |
| System info subsystem | Return useful system data | ✓ | A |
|  |  |  |  |
| FPGA 🡨🡪uP interface | 8-bit Bidirectional bus driven by appropriate FSMs | ✓ | A |
|  |  |  |  |
| FPGA internal 32-bit bus | 32-bit internal bus driven by FSM | ✓ | A |
| Modular format to allow duplication of capability for multiple subsystems | ✓ | A+ |
|  |  |  |  |
| PWM Subsystem | Compile time configuration of number of PWM channels | ✓ | A |
| Run-time programming of period, pulse width, and outputs   * timing resolution = 50nS * 32-bit counters for period + pulse width. | ✓ | A |
| Configurable logic to interface to different types of H-bridge chips and systems | ✓ | A- |
|  |  |  |  |
| Encoder subsystem | Compile time configuration of number of quadrature encoder (QE) channels | ✓ | A |
|  | Swap control of QE inputs | ✓ | A |
|  | QE simulator can be used as input | ✓ | A |
|  | Speed calculation based on counting 50nS clocks during QE pulse | ✓ | A- |
|  | Averaging filter available   * 2,4,8,16, and 32 values | ✓ | A- |
|  |  |  |  |
| RC servo subsystem | Compile time configuration of number of quadrature RC servo channels | ✓ | A |
| Run-time programming of period and pulse widths to suit all formats of RC servo   * timing resolution = 50nS * 32-bit counters for period + pulse width. | ✓ | A |

## 6.2 Low-level microprocessor

|  |  |  |  |
| --- | --- | --- | --- |
| **Topic** | **Features** | **Coded** | **Tested** |
| Device selection | ST Microelectronics STM32-F303RE   * 3.3v, 72MHz clock, FPU * 512Kb flash, 16Kb RAM * 64 pin quad flat pack |  |  |
|  |  |  |  |
| uP 🡨🡪 FPGA | 8-bit Bidirectional bus driven by appropriate FSMs | ✓ | A |
| MBED library to encapsulate comms protocol | ✓- | A- |
| Exercising of basic FPGA features | ✓ | A |
|  |  |  |  |
|  |  |  |  |
| uP 🡨🡪 High-level processor | API undefined at this time | X | X |
|  |  |  |  |
| RTOS structure of uP | Tentative design at this time | X | X |
|  |  |  |  |

## 6.3 High-level processor

No work done at this time.

## 6.4 Vehicle design

|  |  |  |  |
| --- | --- | --- | --- |
| **Topic** | **Features** | **Coded** | **Tested** |
| Mechanical design | No detailed design. Likely to have a similar form factor to Wattbot I |  |  |
|  |  |  |  |
| DC motors | Modified RC servos have worked well for Wattbot I and will be used in Wattbot:*nt*. |  |  |
|  |  |  |  |
| Wheel Encoders | Proposal to retrofit RC servo with a non-contact hall-effect quadrature sensor. Device has been selected but design work yet to be done. |  |  |
|  |  |  |  |
|  |  |  |  |

# 7 Proposal

This is a suitable point in the development of the project to consider its future, i.e. do we continue the development to build a prototype vehicle. The tasks required to build a prototype are

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Notes |  | Cost | Timescale  (days) |
| Development/test board  (not final format) | Order components | AHo/JTH | £100.00 | 1 |
| Layout PCB | AHo | 2 days tech effort | 2 |
| Order PCB | AHo | £50.00 | 14(1) |
| Build PCB | AHo | 2 days tech effort | 2 |
| Basic PCB test(2) | JTH |  | 5 |
| Design and layout production board | AHo/JTH | 2 days tech effort | 2 |
|  |  |  |  |  |
| System software development | Soak-test board with current software | JTH |  | 5 |
| Design STM32 uP system code(3) | JTH |  | 20 |
| Develop local feedback control of velocity of two motors | JTH |  | ?? |
| Develop software library for Raspberry Pi(4) | JTH |  | ?? |
|  |  |  |  |  |
|  |  |  |  |  |
| Small test vehicle | TWO DC motors with encoders | AHo/JTH | £60.00 |  |
| Build robot with test/development board | JTH |  | 2 |
|  |  |  |  |  |
|  |  |  |  |  |

Notes

1. Delivery time
2. Run current software to verify integrity of board and its programming capabilities.
3. Basic system with comms to higher level processor (e.g. Raspberry Pi)
4. C/C++ and/or Python

The aim would be to have a first production board by the start of the new academic year and to have a replacement by Christmas 2020.

Appendix A: Other information

* Technical annex (In preparation)
* SystemVerilog code on GITHUB
  + https://github.com/jimherd/motion\_1