DISEÑO DE UN SISTEMA DE MONITORIZACION PARA LA INCUBADORA ARTIFICIAL DE ALEVINOS DE MOJARRA ROJA UBICADA EN LA PLANTA DE AGROAVICOLA SAN MARINO EN EL MUNICIPIO DE FLANDES - TOLIMA

DORA ALEJANDRA PARRA HERNANDEZ

MIGUEL SUAREZ SIERRA

INSTITTUTO DE EDUCACIÓN SUPERIOR “ITFIP”

FACULTAD DE INGENIERÍA Y CIENCIAS AGROINDUSTRIALES

PROGRAMA DE INGENIERIA ELECTRONICA

EL ESPINAL – TOLIMA - COLOMBIA

2016

DISEÑO DE UN SISTEMA DE MONITORIZACION PARA LA INCUBADORA ARTIFICIAL DE ALEVINOS DE MOJARRA ROJA UBICADA EN LA PLANTA DE AGROAVICOLA SAN MARINO EN EL MUNICIPIO DE FLANDES - TOLIMA

DORA ALEJANDRA PARRA HERNANDEZ

MIGUEL SUAREZ SIERRA

Trabajo de grado para optar al título de:

Ingeniero en Electrónica

DIRECTOR

AMILCAR VILLANUEVA, ING. EN ELECTRONICA

INSTITTUTO DE EDUCACIÓN SUPERIOR “ITFIP”

FACULTAD DE INGENIERÍA Y CIENCIAS AGROINDUSTRIALES

PROGRAMA DE INGENIERIA ELECTRONICA

EL ESPINAL – TOLIMA - COLOMBIA

2016

NOTA DE ACEPTACION

_____

_____

_____

_____

PRESIDENTE DEL JURADO

_____

JURADO

_____

JURADO

El Espinal, Enero 20 de 2016

Agradecimientos

A Dios, por ser fuente suprema de toda sabiduría y ser la luz y guía de nuestras metas así como la fuerza que inspiró nuestro camino.

A nuestras familias, por ser el pilar fundamental para alcanzar tan anhelado triunfo, que representa el final de una de las etapas más importantes de nuestras vida y el inicio de otras que serán aún más enriquecedoras.

Al cuerpo docente del ITFIP y con mayor gratitud al Ing. Amílcar Villanueva por la ayuda prestada, al Ing. Oscar Bernate por el apoyo y orientación brindada para la culminación del presente proyecto y al Ing. Diego Parra por incentivar la búsqueda de alternativas en la consecución de procesos automatizados.

CONTENIDO

pág.

INTRODUCCIÓN 10

2. DEFINICIÓN DEL PROBLEMA 11
2.1 FORMULACIÓN DEL PROBLEMA 11
2.2 DESCRIPCIÓN DEL PROBLEMA 11

3. JUSTIFICACIÓN 13

4. LINEA DE INVESTIGACION 14

5. OBJETIVOS 15
5.1 OBJETIVO GENERAL 15
5.2 OBJETIVOS ESPECÍFICOS 15

6. MARCOS DE REFERENCIA 16
6.1 MARCO LEGAL 16
6.1.1 Normatividad sobre uso del agua 16
6.1.2 Normatividad sobre cultivo de peces 19
6.1.3 Normatividad sobre equipos electrónicos 19
6.1.4 Normatividad cableado eléctrico 20
6.1.5 Otras disposiciones legales 23
6.1.6 Resolución permiso ICA 23
6.2 MARCO CONCEPTUAL 23
6.3 MARCO TEORICO 26
6.3.1 Producción tradicional de alevinos de tilapia roja 26
6.3.2 Ventajas de la incubación artificial 27

6.3.3 Desventajas de la incubación artificial....28
6.3.4 Obtención de huevos para incubación artificial....28
6.3.5 Incubación....29
6.3.6 Desarrollo....30
6.3.7 Larvicultura....31
6.3.8 Sistemas de recirculación para acuicultura....32
6.3.9 Protocolo MODBUS....34
6.4 MARCO INSTITUCIONAL....45
6.4.1 Misión de la empresa....46
6.4.2 Visión al año 2020....46

7. METODOLOGIA....47
7.1 PARTICIPANTES....47
7.2 MATERIALES Y DISPOSITIVOS....47
7.3 PROCEDIMIENTO....47
7.3.1 Realización de mediciones....48
7.4 ANALISIS ESTADO DEL ARTE....50
7.4.1 Proyectos universitarios....50
7.4.2 Empresa Innovaqua....52
7.4.3 Empresa Pentair....56
7.5 MONTAJES LABORATORIOS DE INCUBACIÓN EN COLOMBIA....58
7.5.1 Empresa Omegafish Acuicultura SAS....58
7.5.2 Estación piscícola de Gigante....59
7.6 DISEÑO DE LA SOLUCIÓN PROPUESTA....60
7.6.1 Unidad Terminal Remota (RTU)....60
7.6.2 Unidad Central Maestra (MTU)....62
7.6.3 Sensores....62
7.6.4 Pantalla HMI....62
7.7 DESCRIPCIÓN DE LOS SENSORES Y CÓDIGOS DE LECTURA....63
7.7.1 Sensor de temperatura LM35DZ....63
7.7.2 Sensor de Potencial de Hidrógeno (pH)....65
7.7.3 Sensor de oxígeno disuelto (OD)....68

7.7.4 Sensor de distancia para medir nivel de estanque 72
7.8 CREACIÓN DE LA ESTRUCTURA SCADA 76
7.8.1 Propiedades asignadas a la variable temperatura 84
7.8.2 Propiedades asignadas a la variable pH 86
7.8.3 Propiedades asignadas a la variable de oxígeno disuelto 88
7.8.4 Propiedades asignadas a la variable nivel del estanque 90
7.8.5 Propiedades asignadas a la variable indicador de encendido 91
7.8.6 Propiedades asignadas a la variable indicadora de alarma 92
7.8.7 Creación de los formularios del SCADA 93
7.9 HMI PARA ARDUINO 102
7.10 Configuración correo electrónico envío notificación de alarma 107

8. CRONOGRAMA 110

9. RESULTADOS 111
9.1 INSTALACIÓN DE LOS SENSORES EN LA ESTRUCTURA DE LA INCUBADORA 112
9.2 COMPROBACION DE LA CALIBRACION DE LOS SENSORES 113
9.3 PUESTA EN MARCHA DEL SCADA Y EL HMI 114

10. CONCLUSIONES 119

11. PRESUPUESTO DEL PROTOTIPO 121

BIBLIOGRAFIA 122

Anexo A 125
Anexo B 135
Anexo C 145
Anexo D 147
Anexo E 150

LISTA DE TABLAS

Pág.

Tabla 1. Contenidos máximos permitidos de sustancias 20

Tabla 2. Código de colores conductores eléctricos.....22

Tabla 3. Modos de transmisión MODBUS 35

Tabla 4. Consulta del maestro con formatos ASCII y RTU. 43

Tabla 5. Respuesta de Esclavo con formatos ASCII y RTU 44

Tabla 6. Características sensor de pH.....65

Tabla 7. Características circuito EZO pH.....66

Tabla 8. Características sensor de OD. 69

Tabla 9.Características circuito EZO OD 70

Tabla 10. Características sensor de ultrasonido.....73

Tabla 11. Tabla de la base de datos INCUBAPECES 97

Tabla 12. Consulta de variables en base de datos 117

Tabla 13. Consulta de alarmas en base de datos.....118

LISTA DE FIGURAS

Pág.

Figura 1. Etapas para la incubación artificial 30

Figura 2 Sistema de recirculación de agua.....33

Figura 3. Trama en el modo de transmisión ASCII 38

Figura 4. Trama en el modo de transmisión RTU 40

Figura 5. Situaciones posibles en el intercambio de mensajes entre dispositivos conectados en red en el modo de transmisión RTU. 41

Figura 6. Elementos de medición.....47

Figura 7. Gráfica registrador temperatura.....48

Figura 8. Toma de pH.....49

Figura 9. Toma de OD 50

Figura 10. Vista general del sistema.....51

Figura 11. Esquema funcionamiento 52

Figura 12. Sistema de control Miranda 53

Figura 13. Controladores MIRANDA.....53

Figura 14. Miranda ACB 54

Figura 15. Equipo Miranda.....55

Figura 16. Miranda SDA 55

Figura 17. PLC Pentair 56

Figura 18. Sistema G-Hab Pentair.....57

Figura 19. Laboratorio de incubación artificial de Omegafish 58
Figura 20. Laboratorio de incubación artificial de Gigante 59
Figura 21. Arquitectura del prototipo 61
Figura 22. Sonda de temperatura 63
Figura 23. Conexión sonda a la placa Arduino Mega 64
Figura 24. Sensor de pH 65
Figura 25. Circuito embebido EZO pH 66
Figura 26. Conexión del circuito EZO pH a la placa Arduino 67
Figura 27. Estructura interna y externa de la sonda de Oxígeno disuelto 68
Figura 28. Sensor de OD 69
Figura 29. Circuito embebido EZO OD 70
Figura 30. Conexión del circuito EZO OD a la placa Arduino Mega 71
Figura 31. Sensor de ultrasonidos HC-SR04 72
Figura 32. Sensor con cable de conexión 73
Figura 33. Señales en el sensor HC-SR04 74
Figura 34. Medición del nivel del agua 74
Figura 35. Conexión de HC-SR04 a la placa Arduino Mega 76
Figura 36. Definición de los servidores 78
Figura 37. Parámetros de comunicación con el esclavo Modbus 79
Figura 38. Configuración del Editor de grupos del proyecto 80
Figura 39. Propiedades del editor de variables 80
Figura 40. Ejemplo de escalado 81
Figura 41. Propiedades del editor de alarmas 82

Figura 42. Visor de alarmas 83
Figura 43. Propiedades de la variable temperatura 85
Figura 44. Alarma para alta temperatura 85
Figura 45. Alarma para baja temperatura 86
Figura 46. Propiedades de la variable pH 86
Figura 47. Alarma para alto nivel de pH 87
Figura 48. Alarma para bajo nivel de pH 87
Figura 49. Propiedades de la variable oxígeno disuelto 88
Figura 50. Alarma para alto nivel de OD 89
Figura 51. Alarma para bajo nivel de OD 89
Figura 52. Propiedades de la variable nivel del estanque 90
Figura 53. Alarma para alto nivel del estanque 90
Figura 54. Alarma para bajo nivel del estanque 91
Figura 55. Propiedades de la variable indicador de encendido 91
Figura 56. Propiedades de la variable indicadora de alarma 92
Figura 57. Relé de estado sólido empleado para actuadores y alarma 92
Figura 58. Base de 8 canales para montaje de los relevos 93
Figura 59. Formulario inicial del SCADA 94
Figura 60. Ejemplo de propiedad para la variable temperatura 94
Figura 61. Formulario para visualizar los valores actuales de las variables 95
Figura 62. Matrices de validación para cada variable 95
Figura 63. Configuración editor de serie para cada variable 96
Figura 64. Ejemplo gráfica de tendencia para temperatura 96

Figura 65. Configuración de almacenamiento en base de datos 97
Figura 66. Editor de acción de guardado de los datos. 98
Figura 67. Ejemplo de configuración para la variable de temperatura 98
Figura 68. Establecimiento de las propiedades de la conexión 99
Figura 69. Formulario representación gráfica sistema apagado 100
Figura 70. Formulario representación gráfica sistema encendido.....100
Figura 71. Ventana que muestra el histórico de alarmas.....101
Figura 72. Ventana que muestra el histórico de datos.....101
Figura 73. Módulo Bluetooth HC-05 y tableta 102
Figura 74. Conexión de la aplicación HMI al Arduino Mega 104
Figura 75. Configuración componente para mostrar temperatura.....105
Figura 76. Configuración componente para mostrar nivel 105
Figura 77.Configuración componente para mostrar pH 105
Figura 78. Configuración componente para mostrar OD 106
Figura 79. Configuración componente para pulsador encendido.....106
Figura 80. Presentación final del HMI 107
Figura 81. Cuenta en Gmail para enviar notificaciones de alarma.....107
Figura 82. Establecer evento de correo electrónico.....108
Figura 83. Prototipo terminado.....111
Figura 84. Ubicación de los sensores de temperatura y OD.....112
Figura 85. Ubicación sensores de pH y nivel 112
Figura 86. Lectura de la temperatura.....113
Figura 87. Comprobación calibración pH.....113

Figura 88. Comprobación calibración OD 114
Figura 89. Conexión establecida del SCADA con Arduino Mega 114
Figura 90. Estado actual de las variables 115
Figura 91. Visualización de los valores de las variables y alarmas.....115
Figura 92. Gráficas del comportamiento de las variables 116
Figura 93. Visualización valores variables en Android.....116
Figura 94. Notificación por correo electrónico.....118
Figura 95. Arduino Mega 2560.....125
Figura 96. Creación de la bases de datos en SQLEXPRESS 145
Figura 97. Conexión modulo a Arduino Mega.....147
Figura 98. Monitor serial Arduino para enviar códigos AT 149

LISTA DE ANEXOS

Pág.

Anexo A 125

Anexo B 135

Anexo C 145

Anexo D 150

INTRODUCCIÓN

En la actualidad, las áreas de producción se ven influenciadas por la presencia de instrumentos tecnológicos y técnicas innovadoras que permiten optimizar los procesos y actividades que allí se desarrollan. Uno de estos campos de aplicación es la piscicultura, una actividad que a pesar de ser practicada hace ya bastante tiempo, solo en los últimos años ha cobrado mayor popularidad e importancia.

En Colombia por su diversidad climática y topográfica, la piscicultura se ha convertido en una excelente alternativa de producción e inversión, sin embargo, por diferentes motivos, ésta en la mayoría de los casos carece de herramientas que le ayuden a optimizar y potenciar su trabajo y márgenes productivos.

La automatización industrial es una disciplina que involucra principalmente a las especialidades de la ingeniería electrónica, industrial, química, mecánica, mecatrónica y de sistemas, que va desde un sencillo sistema de control; hasta la instrumentación industrial, sistemas de control y supervisión, los sistemas de transmisión y recolección de datos y las aplicaciones de software en tiempo real para vigilar y controlar las operaciones de procesos y plantas industriales.

El progreso tecnológico en el área de automatización industrial supone una profunda transformación del sistema de producción de una empresa y provoca que la innovación sea un imperativo para el crecimiento y supervivencia de la misma.

Partiendo de lo anterior y estableciendo que las variables fisicoquímicas más representativas en la piscicultura son la temperatura del agua, el pH y el oxígeno disuelto; lograr su medición de forma confiable en una *incubadora artificial de alevinos de tilapia roja,* constituye un paso importante para la automatización por medio de un sistema de monitorización de la misma, que tiene un alcance que va más allá de la simple mecanización del proceso ya que provee a quienes operan un sistema componentes para asistirlos en los esfuerzos físicos y mentales, a través de sensores y transmisores, sistemas de control y supervisión de las operaciones, que permite un incremento tanto de los resultados como en la calidad y reducción de los costos del producto final.

2. DEFINICIÓN DEL PROBLEMA

2.1 FORMULACIÓN DEL PROBLEMA

¿La propuesta de monitorización de la incubación artificial de alevinos de tilapia roja mejora su producción y control?

2.2 DESCRIPCIÓN DEL PROBLEMA

Las tendencias globales que apuntan a la tecnificación de los espacios donde el ser humano realiza sus actividades han llevado a generar una serie de aplicaciones tendientes a solucionar problemas que se presentan en un renglón de la economía tan importante como el agropecuario, donde gran parte de su productividad se ve afectada por factores externos como el clima.

En el caso de la piscicultura el proceso de reproducción de alevinos de manera incontrolada conduce a la sobrepoblación de los estanques donde los reproductores llevan su desarrollo, dada por la imposibilidad de recolectar todas las larvas liberadas, lo cual lleva al aumento en la competencia por el alimento y el espacio, que redunda en una disminución en la producción de alevinos y su calidad.

Estos factores permiten elegir la incubación artificial como método apropiado por el control individual que se tiene sobre los lotes de huevos recolectados de cada hembra. Es decir, cada ovoposición de una hembra puede ser incubada separadamente del resto de los huevos.

El sistema de incubación artificial de huevos de mojarra roja es muy efectivo para producir una alta calidad de alevinos con un mínimo grado de manipulación, control sobre las condiciones físico-químicas del agua de incubación, mejor monitoreo de los reproductores en términos de producción de huevos y alevinos, así como el aprovechamiento del 100% de las larvas sexualmente indiferenciadas para someter a tratamientos hormonales de reversión sexual, con resultados por encima del 99%.

Sin embargo, las variaciones de la condiciones tanto climáticas del entorno como físico-químicas del agua, ocasionan perturbaciones en el proceso de incubación artificial de los alevinos y éste es una parte decisiva en el futuro desarrollo de los mismos, además, se evidencia un mínimo monitoreo manual de las variables por parte de las personas encargadas de la piscícola ocasionado por las ocupaciones

en otra labores, esto desencadena a que no haya lugar a correctivos inmediatos al proceso lo cual origina una elevada tasa de mortalidad.

Por tal motivo se hace necesaria la implementación de un sistema de monitoreo de la incubación artificial con el objeto de lograr un manejo más adecuado de todas las condiciones físico-químicas que vayan en la obtención de un mejor resultado productivo.

3. JUSTIFICACIÓN

En la actualidad el desarrollo tecnológico brinda la oportunidad de encontrar diferentes soluciones a diversos problemas en el ámbito industrial. Uno de los más importantes es el sensado de variables físicas, proceso en el que se han venido presentando grandes mejoras, gracias a la aparición de sensores con características superiores.

De ahí que el campo de la automatización electrónica de equipos se ha venido difundiendo ampliamente en los últimos años en sectores donde no se había tenido en cuenta, logrando con esto una mayor y más controlada ejecución de procesos.

Teniendo en cuenta lo anterior esta propuesta representa un gran aporte a la acuicultura ya que no sólo busca mejorar la producción sino también se presenta como una posibilidad de incrementar la competencia en el plano económico, al tiempo que, respondiendo a su naturaleza académica desde la perspectiva tecnológica, el ITFIP debe desarrollar procesos de investigación que le permitan responder a las problemáticas detectadas en el entorno, especialmente cuando ellas propician espacios de mejoramiento en los ciclos productivos que son base de la economía regional. La incorporación de herramientas tecnológicas en la producción acuícola conlleva a mejorar las condiciones de desarrollo académico, profesional, laboral y socioeconómico de estudiantes y población en general.

4. LINEA DE INVESTIGACION

La línea de investigación a la cual se inscribe la presente propuesta es:
Al área institucional del Desarrollo de las Ingenierías en lo referente a la Electrónica Industrial.

5. OBJETIVOS

5.1 OBJETIVO GENERAL

- Diseñar un sistema de monitoreo para la incubadora artificial de alevinos de mojarra roja existente en la planta de Agroavícola San Marino ubicada en Flandes - Tolima.

5.2 OBJETIVOS ESPECÍFICOS

- Desarrollar el estado del arte en lo referente al funcionamiento de las incubadoras de alevinos
- Evaluar el comportamiento de las variables con el fin de determinar los parámetros del sistema de monitoreo a desarrollar.
- Desarrollar el marco teórico concerniente a las formas de supervisión automatizada más adecuada para la incubadora.
- Establecer los elementos y dispositivos necesarios para la construcción del sistema de monitoreo haciendo énfasis en la utilización de hardware y software libre a fin de diseñar la arquitectura del prototipo.
- Realizar la programación de las partes constitutivas del hardware y el software del sistema de monitoreo que garantice obtener el registro de las variables fisicoquímicas necesarias adecuadas para el óptimo desarrollo de los alevinos y su posterior análisis.
- Comprobar el correcto funcionamiento del prototipo en cuanto a su calibración y desempeño en campo que permita tener confiabilidad en su manejo por parte de quienes lo utilizan.
- Presentar informe de resultados obtenidos encaminados a oportunidades de mejora.

6. MARCOS DE REFERENCIA

6.1 MARCO LEGAL

6.1.1 Normatividad sobre uso del agua

Dado que la propuesta está enfocado hacia la automatización de una incubadora artificial de alevinos de tilapia roja, donde el principal recurso a manejar es el agua, es importante conocer la normatividad que rige en su utilización y manejo adecuado con el objeto de no incurrir en posibles sanciones al desacatar la ley.

"El agua es el recurso más abundante de la Tierra", esta es una afirmación muy conocida por todos; ya que se dice que el agua ocupa dos de las terceras partes de la superficie del planeta que habitamos; además el agua es el mayor componente del cuerpo de todos los seres vivos. Pero cada día aumenta el número de investigaciones en los diferentes medios de comunicación sobre su escasez, el agua que existe en la naturaleza cada vez se hace más insuficiente, o sea, la cantidad de agua disponible para uso humano, pierde las condiciones para tal finalidad, debido al alto consumo e inadecuado manejo que de ella hacemos.

El tema del agua es de gran importancia en estos momentos, puesto que las reservas de agua dulce han ido disminuyendo a nivel mundial, lo que obliga a que se generen reglamentos para su uso, distribución, control y manejo. Es así como en Colombia no se ha desconocido este tema por completo, y los gobernantes han tenido que implementar leyes, normas, tratados y estatutos para controlar el abastecimiento y el aprovechamiento de este importante recurso.

He aquí algunos de los tratados que ha firmado Colombia con respecto al adecuado uso de éste:

- Declaración de Estocolmo de 1972, la cual se refiere al desarrollo sostenible mediante la preservación del medio ambiente. *"Los recursos naturales de la tierra incluidos el aire, el agua, la tierra, la flora y la fauna y especialmente muestras representativas de los ecosistemas naturales, deben preservarse en beneficio de las generaciones presentes y futuras, mediante una cuidadosa planificación u ordenación, según convenga".*[1]

[1] Principio 2 - Declaración de Estocolmo / 1972
http://www.juridicas.unam.mx/publica/librev/rev/derhum/cont/66/pr/pr27.pdf

Análisis Mundial de Cooperación de los Estados (Río de Janeiro), el cual entra en vigor el 21 de marzo de 1994 y en Colombia lo recoge la Ley 99 de 1993 que crea el SINA[2] y el Ministerio del Medio Ambiente, que trata, entre otros, la utilización de los recursos hídricos.

- Johannesburgo en el 2002: "*Ha conducido a casi todos los Estados Latinoamericanos a desmontar paulatinamente la institucionalidad ambiental creada a partir de la Cumbre de Río de Janeiro en 1992. Se establece así la prevalencia de los acuerdos comerciales sobre todos los demás aspectos"*
- Protocolo de Kioto[3], en diciembre de 1997, que en Colombia lo ratifica la Ley 629 del 27 de diciembre de 2000. *(…) promoción de modalidades agrícolas sostenibles a la luz de las consideraciones del cambio climático (…)*[4]

Por otro lado se han implementado también, leyes para la protección y conservación de este recurso como son:

- Código de Recursos Naturales, decreto 2811 del 18 de diciembre de 1974. *"De acuerdo con los objetivos enunciados, el presente Código regula (…) a. El manejo de los recursos naturales renovables, a saber: (…) 1.Las aguas en cualquiera de sus estados."*

- Ley 99 de 1993; la que señala las guías de acción del Ministerio del Medio Ambiente. *(…) Ejercer las funciones de evaluación, control y seguimiento ambiental de los usos del agua, el suelo, el aire y los demás recursos naturales renovables (…)*[5]

- Proyecto de Ley 365 – Cámara de representantes (22 de abril de 2005): *"Por la cual se establecen medidas para orientar la planificación y administración del recurso hídrico en el territorio nacional".*

Sin embargo, pese a los intentos del gobierno por regular el aprovechamiento del recurso, muchos son los vacíos que ha tenido al implementar las leyes que limitan el manejo, el uso, la planificación, el control de la contaminación, la estimación del caudal etc., todo esto ocurre puesto que al tratar de elaborar dichas leyes se ha

---

2 La ley 99 de 1993 creó el Sistema Nacional Ambiental (SINA), que se define como el conjunto de orientaciones, normas, actividades, recursos, programas e instituciones que permiten la puesta en marcha de los principios generales ambientales contenidos en la Constitución Política de Colombia de 1991 y la ley 99 de 1993. El SINA está integrado por el Ministerio del Medio Ambiente, las Corporaciones Autónomas Regionales, las Entidades Territoriales y los Institutos de Investigación adscritos y vinculados al Ministerio

3 Tratado internacional de la Convención Marco de las Naciones Unidas Sobre Cambio Climático, 36 países industrializados firmaron en diciembre de 1997 el Protocolo de Kioto.

4 Artículo 2 – Punto III – Protocolo de Kioto

5 Ley 99 de 1993 – Art. 31 – Punto 12

hecho caso omiso a las cifras reales que establecen los organismos de control tanto nacionales e internacionales, tales como la FAO[6], la Conferencia de las

Naciones Unidas sobre Medio Ambiente y el Desarrollo (CNUMAD) y el IDEAM.

Hay que establecer controles sobre los principales factores que degradan el recurso hídrico como son el sector industria y de servicios, el sector doméstico, y por último el sector con mayor influencia sobre el recurso: el agrícola y pecuario. Teniendo en cuenta el desarrollo de lo señalado en la norma que establece las fases para la ordenación y manejo de cuencas hidrográficas en el país (Decreto 1729 de 2004), donde además de las fases establecidas en el mencionado Decreto, se incluye una fase que es fundamental para el éxito en la ordenación de cuencas, lo cual da las pautas para un mejor control de éste recurso en los sectores anteriormente mencionados.

Como el recurso hídrico se ha visto maltratado se han implementado ciertas medidas como son las tasas por uso y las tasas retributivas, según el Artículo 42 del proyecto de Ley 365:

"La utilización del recurso hídrico por cualquier persona natural o jurídica, ya sea para aprovecharlo o para introducir o arrojar directamente en él aguas residuales o servidas de cualquier origen, estará sujeta al pago de tasas por uso y tasas retributivas, respectivamente, cuyos valores serán determinados y recaudados por la respectiva autoridad ambiental competente, entendida esta última como aquella con la facultad de otorgar la concesión o el permiso de vertimiento correspondiente. Todo lo anterior, de acuerdo con la reglamentación que para el efecto expida el Gobierno Nacional".

El propósito de estas tasas al servir como "instrumentos económicos" es tratar de que las personas usen este recurso de una forma racional de acuerdo con la calidad y con la cantidad del líquido; las tasas por uso son empleadas primero que todo por la utilización del recurso por cualquiera, tal y como lo establece el Artículo 43 de la Ley 99 de 1993:

"La utilización de aguas por personas naturales o jurídicas, públicas o privadas, dará lugar al cobro de tasas fijadas por el Gobierno Nacional que se destinarán al pago de los gastos de protección y renovación de los recursos hídricos"; Con estas tasas es posible controlar parcialmente los problemas de la calidad del líquido parcialmente, ya que no deben ser consideradas como las únicas maneras de generar un control, por ello se deben establecer otras medidas que no solo midan la cantidad de contaminantes que son vertidos en las cuencas sino también se deben recurrir a métodos que creen una conciencia para que en los sectores industriales se usen materiales que no sean tan dañinos para el medio ambiente.

[6] La FAO ayuda a los países en desarrollo y a los países en transición a modernizar y mejorar sus actividades agrícolas, forestales y pesqueras, con el fin de asegurar una buena nutrición para todos.

Partiendo de lo planteado en las normas es necesario establecer los elementos adecuados a implementar en la realización de la automatización que no originen desaprovechamiento del agua ni residuos contaminantes que vayan en detrimento tanto del recurso hídrico como de los resultados en la reproducción de los alevinos, por lo tanto, deben usarse materiales a base de PVC, en el caso de las carcasas de los equipos a instalar, y acero inoxidable en todos aquellos elementos que se utilicen para adquirir mediciones o hagan contacto con el agua como en los sensores que toman la temperatura, el pH o el caudal.

6.1.2 Normatividad sobre cultivo de peces

Colombia cuenta con excelentes condiciones climáticas, topográficas, hidrológicas y edafológicas para desarrollar la acuicultura, entre ellas se destaca su localización geográfica en la zona tropical lo que provee un régimen de temperaturas estables durante el año, lo cual permite considerar al país en el mundo como una potencia en biodiversidad.
Para el desarrollo de la acuicultura el gobierno ha designado un organismo denominado AUNAP (INPA) "*para señalar los requisitos y condiciones al establecimiento de las actividades acuícolas"*[7] asimismo "*se promocionará el fomento y desarrollo de la acuicultura… y abastecimiento de semillas*…"[8]

6.1.3 Normatividad sobre equipos electrónicos

Los componentes electrónicos deben cumplir la directiva RoHS que establece las "Restricciones en el uso de ciertas sustancias peligrosas en equipo electrónico y eléctrico", del inglés "restriction on the use of certain hazardous substances in electrical and electronic equipment". La norma restringe el uso de seis sustancias consideradas como peligrosas y dañinas al medio ambientes, estas son: Plomo, Mercurio, Cadmio, Cromo VI, PBB (PoliBromoBifenilos) y PBDE (PoliBromoDifenil Eter). Las dos últimas son retardantes de llamas. Estas sustancias no necesariamente son anuladas en los componentes y materiales de los equipos electrónicos y eléctricos sino que la norma RoHS establece los porcentajes máximos de esas sustancias que pueden ser usados en la fabricación de los mismos. Ver Tabla 1.

Se debe exigir a los fabricantes la documentación necesaria para demostrar que sus productos cumplen la norma antes de comprarlos.

---

[7] Artículo 42 Ley 13 de 1990

[8] Artículo 43 Ley 13 de 1990

Tabla 1. Contenidos máximos permitidos de sustancias

Sustancia	Contenido (mg/kg)
Plomo	< 1000
Cadmio	< 100
Mercurio	< 1000
Cromo hexavalente	< 1000
PBBs	< 1000
PBDEs	< 1000
Fuente NTC 5720. Requisitos para materias primas, componentes e insumos	

Fuente: Norma RoHS

6.1.4 Normatividad cableado eléctrico

Los cables y conductores eléctricos son el medio para el transporte de la energía eléctrica y constituyen una parte trascendental en las instalaciones que inciden sobre la seguridad de los operadores de la automatización del sistema de incubación artificial , por lo tanto, es de vital importancia tener claridad sobre las reglas establecidas al respecto y aplicarlas al proyecto ya que las Entidades Certificadoras y los Inspectores pondrán especial énfasis en la verificación de los requisitos establecidos por el Reglamento (RETIE) y el Código Eléctrico Colombiano (NTC 2050).

- **Cables de control**. Se usan para llevar señales entre aparatos en interface directa con el sistema eléctrico de potencia, tales como transformadores de corriente, transformadores de potencia, relés interruptores y equipos de medición.
  Los cables de control son cables multiconductores que llevan señales eléctricas usadas para monitorear o controlar sistemas eléctricos de potencia y sus procesos asociados. La tensión de operación de estos cables es de 600 V.

El aislamiento usado para los cables de control es PVC retardante a la llama para una temperatura de operación de 90°C. La chaqueta también es PVC, resistente a la abrasión y a la manipulación durante la instalación y operación.[9]

- **Cables de instrumentación**. Son usados para llevar señales desde procesos de monitoreo a procesos de analizadores, usualmente equipo electrónico, y de los analizadores al equipo de control en el sistema eléctrico de potencia. Los cables de instrumentación son cables multiconductores que transportan señales eléctricas de baja potencia (los circuitos son inherentemente de potencia limitada) usadas para monitorear o controlar sistemas eléctricos de potencia y sus procesos asociados. La tensión de operación de estos cables es de 300 V y también son aptos para usos en 600 V en circuitos de potencia limitada. El aislamiento usado para los C cables de instrumentación es PVC retardante a la llama, para una temperatura de operación de 105°C. La chaqueta también es PVC, resistente a la abrasión y a la manipulación durante la instalación y operación.[10]

- **Cables de baja tensión**. En general, se usan en el proceso de utilización y van desde la salida de los transformadores de distribución hasta la conexión con los equipos. Se consideran cables de baja tensión aquellos cuyo voltaje de operación es como máximo de 1000 V entre fases. Dentro de esta familia se encuentran principalmente cables para 600 V. De forma básica un cable de baja tensión está compuesto por uno o varios conductores de cobre y materiales que componen el aislamiento o la chaqueta, que generalmente son plásticos. Opcionalmente se construyen con pantalla electrostática y en algunas aplicaciones específicas con armaduras para protección mecánica.
  Los materiales de aislamiento más usados son el PVC, el Polietileno Termoplástico (PE) y el Polietileno Reticulado (XLPE). Dentro de estos tipos, se encuentran compuestos con características especiales como retardancia a la llama, compuestos no halogenados, baja emisión de humos, resistencia a los rayos solares, entre otros.

  La chaqueta proporciona resistencia mecánica a la abrasión y a posibles daños ocasionados durante la instalación y/o manipulación en operación. Para algunas aplicaciones a la intemperie o en instalación subterránea se usa el PE que posee una mejor impermeabilidad al agua y buena resistencia a los rayos solares.[11]

- **Puesta a tierra**. Partiendo de que la propuesta está constituido por un conjunto de equipos electrónicos, interconectados con cables y que requieren energía y

[9] Articulo 17 Reglamento Técnico de Instalaciones Eléctricas (RETIE) y Sección 340 NTC 2050
[10] ibíd.
[11] Artículo 17 RETIE

protección eléctrica para su operación normal se debe tener en cuenta en la instalación del sistema de alimentación eléctrica la puesta a tierra necesaria para la operación normal de los equipos con lo cual se "*garantiza condiciones de seguridad a los seres vivos* (operarios del sistema y alevinos), *permitir a las protecciones despejar rápidamente las fallas, evitar ruidos eléctricos…"*[12]

- **Método de identificación de conductores**. *"Con el objeto de evitar accidentes por mala interpretación de los niveles de tensión y unificar los criterios para instalaciones eléctricas, se debe cumplir el código de colores para conductores establecido en la Tabla 13 (Tabla 2 de este documento). Se tomará como válida para determinar este requisito el color propio del acabado exterior del conductor o en su defecto, su marcación debe hacerse en las partes visibles con pintura, con cinta o rótulos adhesivos del color respectivo."*[13]

Tabla 2. Código de colores conductores eléctricos

SISTEMA	MONOFÁSICO		TRIFÁSICO				
			(Y) ESTRELLA		(Δ-) DELTA	(Δ) DELTA	
Tensión (V)	120	120/240	208/120	480/277	240/208/120	240	480
Fases	1	2	3	3	3	3	3
Neutro	1	1	1	1	1	N/A	N/A
Fases	Negro	Negro	Amarillo	Amarillo	Negro	Negro	Amarillo
		Rojo	Azul	Naranja	Naranja	Azul	Naranja
			Rojo	Café	Azul	Rojo	Café
Neutro	Blanco	Blanco	Blanco	Gris	Blanco	N/A	N/A
Tierra de Protección	Desnudo o Verde	Desnudo o Verde	Desnudo o Verde	Desnudo o Verde	Desnudo o Verde	Desnudo o Verde	Desnudo o Verde
Tierra Aislada	Verde amarillo	Verde amarillo	Verde amarillo	N/A	Verde amarillo	N/A	N/A

Fuente: RETIE

12 Sección 250 NTC 2050

13 Numeral 4 Artículo 11 del RETIE

6.1.5 Otras disposiciones legales

En el artículo 50 de la ley 1152 de 2007 del Estatuto de Desarrollo Rural el gobierno establece que:

"El Ministerio de Agricultura y Desarrollo Rural, en coordinación con las entidades del Sistema Nacional de Ciencia y Tecnología Agroindustrial y teniendo en cuenta la agenda de competitividad, definirá una política de generación y transferencia de tecnología para la estrategia de desarrollo rural, orientada a mejorar la productividad y la competitividad, optimizar el uso sostenible de los factores productivos, facilitar los procesos de comercialización y de transformación, y generar valor agregado, que garantice a largo plazo la sostenibilidad ambiental, económica y social de las actividades productivas, y que contribuya a elevar la calidad de vida, la rentabilidad y los ingresos de los productores rurales".

Lo cual favorece el desarrollo de la propuesta que se presenta brindando facilidades para su ejecución y puesta en marcha.

6.1.6 Resolución permiso ICA

La resolución 002470 del 30 de junio de 2009 del ICA otorga un permiso de cultivo y comercialización de tilapia roja y plateada en estado de alevinaje a la sociedad Agroavícola San Marino Ltda., por diez (10) años, igualmente, autoriza el engorde y comercialización de reproductores y carne.

6.2 MARCO CONCEPTUAL

- **Alevino**: Embrión que nace del huevo de los peces luego del período de fecundación.

- **Automatización:** uso de sistemas o elementos computarizados y electromecánicos para controlar maquinarias y/o procesos industriales sustituyendo a operadores humanos.

- **Base de Datos:** Conjunto de datos que pertenecen al mismo contexto, almacenados sistemáticamente para su posterior uso; mientras que un sistema administrador de base de datos (DBMS) es una herramienta de software que

permiten crear estructuras, almacenar datos y posteriormente acceder a ellos de forma rápida, segura y eficiente.

- **Condiciones físico-químicas:** las físicas son aquellas que se pueden medir, sin que se afecte la composición o identidad de la sustancia (temperatura) y las Químicas, son las que se observan cuando una sustancia sufre un cambio químico, es decir, en su estructura interna, transformándose en otra sustancia.

- **Controlador electrónico:** es un medidor al que se le agrega la posibilidad de fijar un punto de fijación y un circuito que compara la diferencia entre el valor real de la variable medida y la deseada, actuando en consecuencia para habilitar o no el sistema controlado que llevará la variable hasta niveles iguales al deseado de tal forma que, al hacerse cero la diferencia entre ambas la acción sobre el sistema cese.

- **Incubación artificial:** Consiste en la cría de algún ser vivo sin la intervención de los progenitores reemplazando a éstos por condiciones similares creadas por el hombre.

- **Incubadora:** dispositivos de diferente tipo que tienen la función común de crear un ambiente adecuada para el crecimiento o reproducción de seres vivos.

- **Instrumentación Industrial:** Conocimiento de la correcta aplicación de los equipos encaminados a apoyar al operario/analista de procesos en la medición, regulación, observación, seguridad y almacenamiento del valor de una variable dentro de un proceso productivo. Como parte de la instrumentación industrial podemos mencionar a los sensores, actuadores, controladores, tuberías, mandos eléctricos, suministros de energía entre otros.

- **Intogresión genética:** se refiere al hecho de introducir en una población genes de otra diferente.

- **OD:** es la cantidad de oxígeno que está disuelto en el agua. Es un indicador de cómo de contaminada está el agua o de lo bien que puede dar soporte esta agua a la vida vegetal y animal. La cantidad de oxígeno que puede disolverse en el agua (OD) depende de la temperatura. El agua más fría puede contener más oxígeno en ella que el agua más caliente.

- **Oreochromis:** nombre científico de la tilapia o mojarra roja.

- **ORP:** potencial de oxidación-reducción. Es una medida de la limpieza del agua y su capacidad para descomponer contaminantes. Los valores ORP se expresan en mili voltios.

- **pH:** medida de la acidez o alcalinidad de una solución. El pH indica la concentración de iones hidronio.

- **PLC:** es un sistema electrónico programable diseñado para ser usado en un entorno industrial, que utiliza una memoria programable para el almacenamiento interno de instrucciones orientadas al usuario.

- **Reversión sexual:** es el cambio de sexo de un animal que el hombre realiza artificialmente mediante la administración de hormonas.

- **Saco vitelino:** es un anexo membranoso adosado al embrión que provee nutrientes y oxígeno al embrión en peces, tiburones, reptiles, aves y mamíferos.

- **SCADA:** *Supervisory Control and Data Adquisition* (Supervisión de Control y Adquisición de Datos): Aplicación de software especialmente diseñado para la captura de información de un proceso o planta industrial (aunque no es absolutamente necesario que pertenezca a este ámbito), con esta información es posible realizar una serie de análisis o estudios con los que se pueden obtener valiosos indicadores que permitan una retroalimentación sobre un operador o sobre el propio proceso.

- **Sensor:** dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas.

- **Sistemas de Control:** Conjunto de dispositivos (instrumentación industrial) adicionales al proceso, que llevan o ayudan al control o regulación del proceso; permitiendo fabricar productos con mayor calidad a menor costo y tiempos óptimos de producción.

- **Tilapia roja:** también conocida como Mojarra roja, es un pez que taxonómicamente no responde a un solo nombre científico. Es un híbrido producto del cruce de cuatro especies de *Tilapia*: tres de ellas de origen africano y una cuarta israelí.

6.3 MARCO TEORICO

6.3.1 Producción tradicional de alevinos de tilapia roja

Las características aparentemente positivas de las tilapias como la maduración precoz, la facilidad de reproducción, la realización de puestas frecuentes y múltiples y el elevado nivel de cuidados parentales pueden formar también parte de la base de muchos desafíos que se presentan en los sistemas tradicionales de producción de semilla de tilapia. El primero de todos, la reproducción incontrolada que conduce a sobrepoblación, que frena el crecimiento en los tanques de engorde y la reproducción en los estanques de reproducción. En los sistemas de estanques de reproducción, la cantidad producida de larvas normalmente aumenta rápidamente después que los reproductores son introducidos y luego disminuye gradualmente.

Este fenómeno se atribuye a dos razones principales:

Uno, es imposible recolectar todas las larvas liberadas, de forma que el estanque pronto estará superpoblado con los animales resultantes de las puestas precoces. Esto permite un aumento en la competencia por el alimento y el espacio, que redunda en una disminución de la producción de semilla. También, se produce un considerable número de casos de canibalismo en larvas jóvenes por parte de larvas mayores que producen un descenso en la producción de semilla.

La segunda razón es que la puesta de las hembras no ocurre de forma sincronizada justo después que los nuevos reproductores introducidos completen su primera puesta. Como resultado, la producción de larvas se produce de forma continua, pero a un ritmo bajo. Debido a este comportamiento de puesta asincrónica también se aumentan las probabilidades de que se produzca canibalismos entre las larvas.

La inversión de tiempo y energías por parte de las hembras en practicar los cuidados parentales también es causa de una inferior productividad en los sistemas de producción de tilapia. Como resultado, necesitan de un periodo de aproximadamente dos semanas para reacondicionarse antes de volver a desovar. Por tanto es necesaria cualquier reducción en el periodo de incubación bucal o de reacondicionamiento para aumentar la productividad de los reproductores.

Conforme se intensifican los sistemas de engorde de tilapia, hay una creciente demanda de producción de larvas y alevines machos. Este método ampliamente practicado de producir sólo tilapias machos es mediante la reversión sexual por hormonas. Este método tiene la ventaja por el hecho que el sexo de la tilapia se determina en las primeras semanas después de la eclosión y puede ser influido por la administración de andrógenos (para producir lotes exclusivamente de

machos) o estrógenos (para producir lotes exclusivamente de hembras) durante esas primeras semanas. La forma más difundida de administración es a través del alimento pero para que el tratamiento sea efectivo, las hormonas deben administrarse tan pronto como sea posible después de la eclosión. También es importante que los alimentos formulados con hormonas sean la principal fuente de alimento de las larvas.

Estas condiciones exigen que las larvas sean separadas de sus madres tan pronto como sea posible, lo que supone un considerable desafío en los sistemas tradicionales de producción de larvas de tilapia.

Los problemas resultantes para el operario de una *planta de incubación* de tilapia son la baja productividad de semilla, las poblaciones de individuos de tamaño no uniforme y el bajo éxito en la producción de poblaciones de un solo sexo.

En el sistema tradicional de producción, los animales se aparean asincrónicamente en el mismo estanque donde se incuban los huevos y eclosionan las larvas y posterior a la cosecha de las larvas, los reproductores se reacondicionan para el siguiente ciclo sin intervención por parte del productor.

En este sistema es imposible recolectar todas las larvas, de forma que el estanque pronto estará superpoblado lo que aumenta la competencia por el alimento, el espacio y al canibalismo reduciendo la producción de semilla.

Las hembras de *Oreochromis* incuban las larvas en su boca durante 10 días, tiempo en el que no consumen alimento. Finalizada esta etapa, requieren dos semanas para reacondicionarse antes de volver a desovar, esto conduce a que los intervalos entre los desoves sean muy largos y se disminuye la vida reproductiva de las hembras.

Uno de los limitantes en el proceso de reversión sexual, es obtener una adecuada cantidad de post-larvas sexualmente indiferenciadas para iniciar el tratamiento hormonal correspondiente.

6.3.2 Ventajas de la incubación artificial

La principal ventaja de la incubación artificial es el control individual que se tiene sobre los lotes de huevos recolectados de cada hembra. Es decir, cada ovoposición de una hembra puede ser incubada separadamente del resto de los huevos.

El sistema de incubación artificial de huevos de Tilapia es muy efectivo para producir una alta calidad de alevinos con un mínimo grado de manipulación, control sobre las condiciones físico-químicas del agua de incubación, mejor

monitoreo de los reproductores en términos de producción de huevos y alevinos, así como el aprovechamiento del 100% de las larvas sexualmente indiferenciadas para someter a tratamientos hormonales de reversión sexual, con resultados por encima del 99%. Al poder incubar embriones de la misma edad, o con diferencia de edades muy cercanas, se obtienen poblaciones con diferencias de tamaño mínimas lo que evita problemas de canibalismo, además la técnica de incubación artificial permite un programa de selección eficiente por familias, y así se evita la disminución de la introgresión genética.

6.3.3 Desventajas de la incubación artificial

Una de las desventajas del sistema es la demanda de tiempo. También necesita que los reproductores sean manejados periódicamente y esto se traduce en un aumento de mano de obra. Logísticamente, no es posible aplicar el método de destete en estanques, en tanques es más aplicable, pero son costosos de construir y manejar; las hapas, jaulas de red de malla fina, que pueden construirse de forma más sencilla y mantenerse en estanques o incluso en lagos y lagunas, han demostrado que son efectivos para el mantenimiento de reproductores.

Adicionalmente, se requiere de una infraestructura adecuada para el montaje del sistema de incubación y larvicultura que mantenga las condiciones de agua óptimas para obtener mejores resultados, esto aumenta los costos de producción.

6.3.4 Obtención de huevos para incubación artificial

La obtención de huevos para incubación artificial requiere de cinco pasos principales:

- Acondicionamiento y siembra de reproductores.
- Adaptación e incubación de los huevos.
- Absorción del saco vitelino en bandejas.
- Adaptación de las larvas a las bandejas y acostumbramiento al alimento.
- Reversión sexual.

En Tailandia y desde hace pocos años en Brasil, se acondicionan los reproductores en donde las hembras tienen un periodo de descanso, esto permite controlar también el crecimiento de las hembras y mantener lotes de reproductores de tallas homogéneas y de tamaño adecuado para no tener dificultades en la manipulación. Las hembras, son mantenidas en jaulas de malla a densidades elevadas (2,5 Kg/m2) durante 10 a 14 días donde reciben alimento balanceado en proporción de 2-3 % de la biomasa. Posteriormente se trasladan a las jaulas de reproducción, de mayor tamaño, donde permanecen de 5 a 7 días con los

machos, a una densidad más baja (6 peces/m2). En este periodo se pueden alimentar, en cuyo caso la cantidad de alimento es menor que en las jaulas de descanso. Una vez se recogen los huevos, en el día 5° o 7°, las hembras regresan nuevamente a las jaulas de descanso; mientras un lote de hembras está trabajando durante 5 o 7 días, debe haber dos lotes descansando durante 10 o 14 días. Los machos eventualmente pueden descansar.[14]

Está muy bien establecido que el destete, la práctica de retirar los huevos y larvas recién eclosionadas de la boca de los reproductores de *Oreochromis* spp., dan como resultado el aumento de la producción de semilla. Se comparó la producción de semilla de tilapia roja en tanques de agua salobre entre los métodos de incubación natural y de destete y encontraron que el primero sólo obtenía una producción de 3.3 semillas/m^2/día, mientras que el segundo método recogía 91.7 semillas/m^2/día.

6.3.5 Incubación

Una vez revisadas las hembras, al día 5° o 7°, sus huevos fecundados son retirados de la cavidad oral y son divididos en lotes dependiendo del estadio de desarrollo. Los huevos se desinfectan con soluciones yodadas, formalina, verde de malaquita o acriflavina, para evitar infecciones bacterianas, principalmente *Aeromona hydrophyla* y *Pseudomona fluorescens*, o de hongos como *Saprolegnia sp., Fusarium sp. y Trichoderma sp.*, lo que puede disminuir los porcentajes de eclosión considerablemente.

Los huevos de las especies de *Oreochromis* se incuban en recipientes con fondo redondeado, lo cual permite la continua rotación de los huevos. Debido a su gran tamaño (1,4 – 2,2 mm) (3), y peso (3,8 – 7,8 mg), tienden a caer rápidamente al fondo del recipiente por lo cual se debe mantener un flujo de agua constante, simulando el movimiento de rotación que los huevos sufren en la boca de la hembra. (Ver Figura 1.)

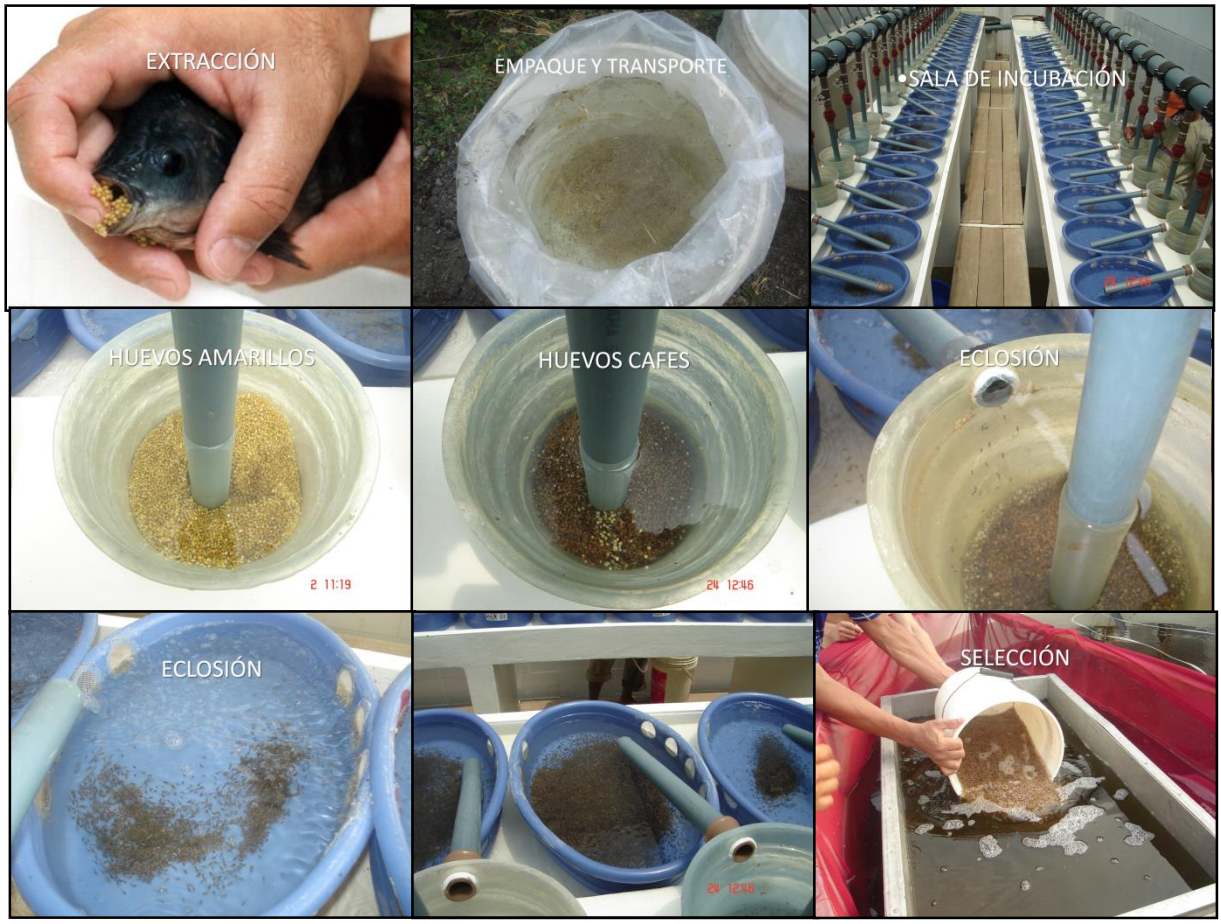
Usando tablas vibradoras o recipientes cónicos con flujo de agua descendente, se obtuvo resultados de hasta un 59% de sobrevivencia. Las principales pérdidas son debidas a daños físicos causados al corion de los huevos y algunas veces por stress debido a un imbalance osmótico y contaminación bacteriana o por hongos.

Incubadoras de 20 litros de capacidad, pueden ser usadas para incubar hasta 80.000 huevos con gran eficiencia en la utilización de agua (10.000 huevos requieren 1 lt s^{-1}, comparado con cerca de 1 lt min^{-1} para 1000 huevos en incubadoras más pequeñas).

[14] PRIETO, Camilo. Incubación artificial de huevos embrionados de tilapia roja. Medellín: Universidad de Antioquia, 2001

La calidad del agua es importante para obtener buenos resultados durante la incubación, esta debe someterse a un proceso de filtración a través de filtros de gravilla o de arena, o un esterilizador de rayos UV con posterior recirculación del agua, para mantener las condiciones constantes.

Figura 1. Etapas para la incubación artificial



Fuente: Piscícola Agroavicola San Marino

6.3.6 Desarrollo

Los rangos de temperatura aconsejados en la etapa de incubación están entre 24-32°C, con un óptimo de 28-32°C; si se mantienen estas temperaturas constantes se pueden lograr supervivencias cercanas al 80% en aproximadamente 96 horas.

El valor del pH debe oscilar entre 7 y 9, la dureza total entre 50 y 350 ppm y el oxígeno disuelto entre 5 y 9 mg/L.

Los huevos en estadios tempranos (2 células, 2-3 horas post-fertilización, blástula, 10-12 horas) son más tolerantes a cambios de temperatura que los zigotos en estadios avanzados (gastrulación 14-30 horas post-fertilización, cierre del blastoporo 30–48 horas), siendo la fase más crítica el momento de la eclosión (90-102 horas post-fertilización).

6.3.7 Larvicultura

Después de la eclosión, las larvas emergen a la superficie y van abandonando las incubadoras para caer atrapadas en bandejas de poca profundidad que pueden ser utilizadas para mantenerlas hasta por 20 días, una vez nadan horizontalmente y comen activamente se trasladan a unidades más grandes como estanques o jaulas. Cada caja puede mantener de 10.000-20.000 larvas, alimentadas con alimento balanceado y formulado con hormonas estrogénicas o androgénicas que aseguren una alta tasa de reversión gonadal. El tiempo que toman las larvas en reabsorber su saco vitelino varía de 4 a 5,5 días, si se mantienen las mismas condiciones ambientales que se presentaron en el proceso de incubación.

Las bandejas de aluminio o plástico deben tener dos filas de perforaciones de 2 cm de diámetro y protegidos con malla fina a lo largo de cada lado de la bandeja, para evitar la fuga de las larvas contenidas en ellas.

El principal riesgo durante la fase de larvicultura es la infección por *Trichodina sp.* o *Dactilogyrus sp*., parásitos que atacan la piel y branquias, produciendo entre 70-80% de mortalidad en la población en un periodo de 10 días. Si se mantienen las condiciones del agua de buena calidad se minimiza este riesgo.

El sistema de las bandejas ha sido evaluado en términos del efecto del flujo de agua y densidad en cada bandeja sobre el comportamiento de las larvas y alevinos. Estos últimos sobrevivieron mejor a altos flujos de agua, aunque su tasa de crecimiento específico fue inversa.

La supervivencia por bandeja es cercana al 90% a densidades entre 5000 a 12000 larvas, con flujos de 3 a 4 lt min-1, mientras que las mejores tasas de crecimiento específico se encuentran a bajos flujos de agua (2 lt min-1). Las larvas producidas en este sistema presentan mejor crecimiento (11%/día vs 8,3%/día) y sobrevivencia que las larvas producidas naturalmente (73% vs 98,4%).

A modo de conclusión podríamos decir que este sistema de incubación ampliamente utilizado en países asiáticos, ha abierto nuevos horizontes a países occidentales y regiones donde las explotaciones no cuentan con extensiones

grandes de tierra o sus condiciones climáticas son muy extremas. Desde el punto de vista investigativo, la técnica se presta para realizar trabajos sobre desempeños genéticos superiores y mayor supervisión individual de los animales.

Trabajos en granjas productoras de alevinos en Brasil, producen un millón de alevinos de tilapia/mes, aún en invierno, con tasas de eficiencia en reversión sexual del 99%.

En Colombia, en los llanos orientales, se está trabajando con este sistema de producción, y actualmente se evalúa su viabilidad económica puesto que la inversión inicial es elevada y requiere personal entrenado.

Aún hay mucho por investigar sobre técnicas de manejo de los reproductores, sincronización de desoves, manipulación y desinfección de los huevos, sin embargo es una herramienta tecnológica de producción de alevinos promisoria ya que hace uso racional de agua y tierra.

6.3.8 Sistemas de recirculación para acuicultura

Hoy la acuicultura a nivel mundial se encuentra en franco crecimiento, según proyecciones de FAO en el año 2015 la producción proveniente de la acuicultura será de 74 millones de toneladas. Para lograr la sustentabilidad es necesario intensificar los cultivos, valiéndose de tecnología como sistemas de recirculación de agua (SRA) y tratamiento de la misma, optimizando un recurso tan valioso.[15]

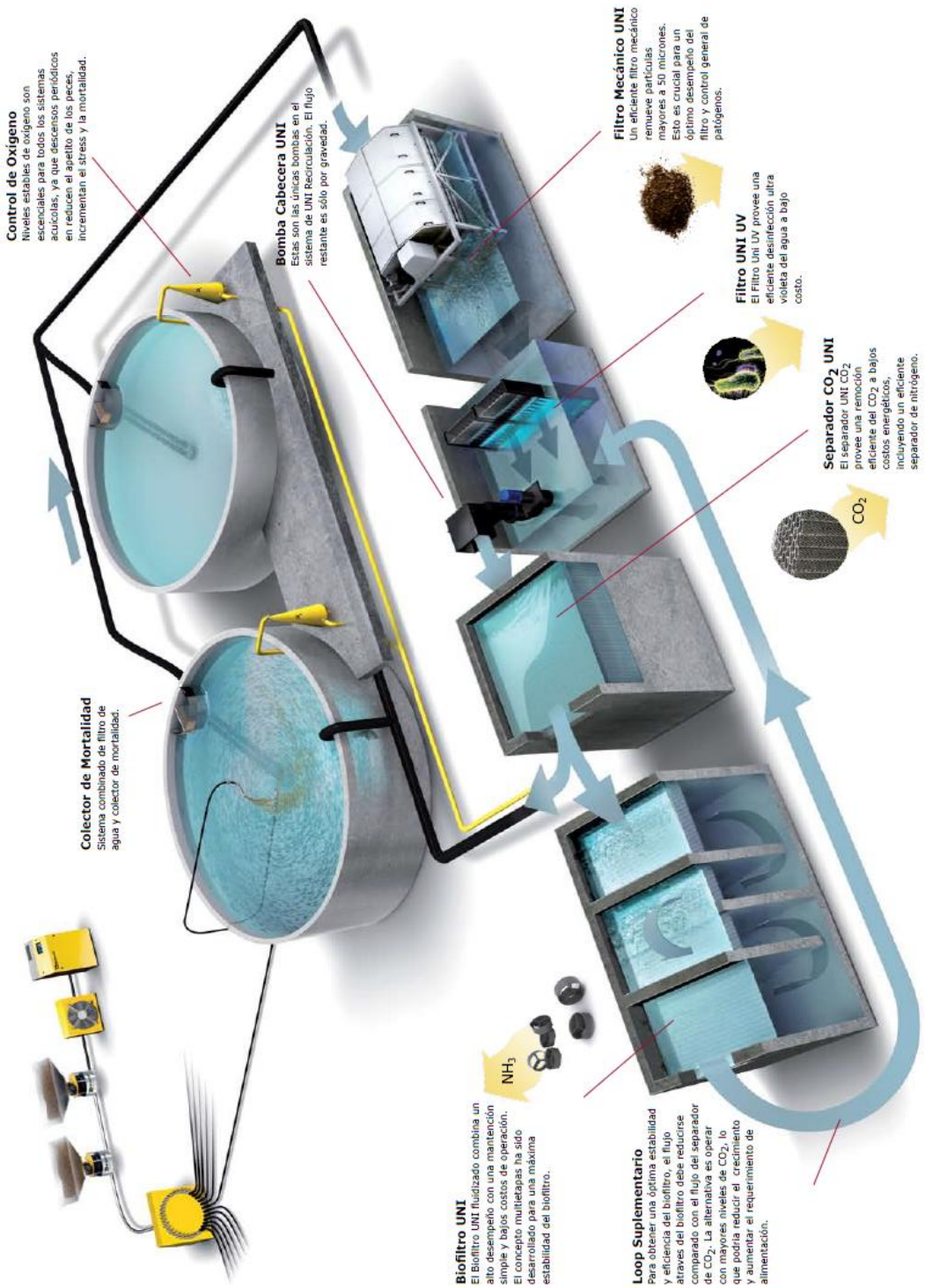
La utilización de la tecnología en el tratamiento del agua tiene como ventajas: un monitoreo y control constante de las variables físico-químicas y sanitarias del agua, la reutilización del agua y producciones de altas densidades, solamente un pequeño porcentaje de agua es reemplazado diariamente. La temperatura, salinidad, pH, alcalinidad, composición química y el oxígeno son monitoreados y continuamente controlados. (Ver Figura 2)

Los residuos sólidos son filtrados y removidos, se incorpora oxígeno para mantener concentraciones suficientes para la densidad de peces en cultivo, Y por último el efluente es tratado en bio-filtro para la conversión biológica del nitrógeno amoniacal a nitrato.

Un sistema de recirculación consiste en los siguientes componentes: una cierta cantidad de depósitos de agua para los peces, una unidad de tratamiento del agua

[15] GALI MERINO, Oscar y SAL, Facundo. Sistemas de recirculación y tratamiento de agua. Santa Ana-Corrientes, Argentina, 2007. 37 p.

Figura 2 Sistema de recirculación de agua



Fuente: Empresa AkvaGoup

unas bombas y unas tuberías para el suministro de agua así como para su retorno. El corazón del sistema es la unidad de tratamiento de agua.

Diseñar y operar un SRA requiere de una sólida comprensión de las operaciones unitarias y procesos incluidos; la falla de cualquiera de estas operaciones puede ocasionar que falle la totalidad del sistema, usualmente resultan en la muerte de los peces en el proceso.

6.3.9 Protocolo MODBUS

El protocolo de comunicaciones industriales MODBUS fue desarrollado en 1979 por la empresa norteamericana MODICON y debido a que es público, relativamente sencillo de implementar y flexible se ha convertido en uno de los protocolos de comunicaciones más populares en sistemas de automatización y control. A parte de que muchos fabricantes utilizan este protocolo en sus dispositivos, existen también versiones con pequeñas modificaciones o adaptadas para otros entornos (como p.ej JBUS o MODBUS II)

MODBUS especifica el procedimiento que el controlador y el esclavo utilizan para intercambiar datos, el formato de estos datos, y como se tratan los errores. No especifica estrictamente el tipo de red de comunicaciones a utilizar, por lo que se puede implementar sobre redes basadas en Ethernet, RS-485, RS-232 etc.

- **Descripción general**

MODBUS funciona siempre en modo maestro-esclavo (cliente - servidor), siendo el maestro (cliente) quien controla en todo momento las comunicaciones con los esclavos que pueden ser hasta 247. Los esclavos (servidores) se limitan a retornar los datos solicitados o a ejecutar la acción indicada por el maestro. La comunicación del maestro hacia los esclavos puede ser de dos tipos:

- “peer to peer”: en que se establece comunicación “maestro - esclavo”, el maestro solicita información y el esclavo responde (se envía el comando a un dispositivo comprendido entre las direcciones 1 a 247).

-“broadcast”: en que se establece comunicación “maestro - todos los esclavos”, el maestro envía un comando a todos los esclavos de la red sin esperar respuesta (se envía a la dirección 0).

Como se puede ver, la secuencia básica en las comunicaciones MODBUS consiste siempre en una trama de pregunta, seguida de su correspondiente trama de respuesta:

- Pregunta: con el código de función que indica al esclavo que operación ha de realizar, y los bytes necesarios (datos, comprobación…) para su ejecución.

- Respuesta: con la confirmación o datos resultantes de la ejecución de la función.

Existe algún caso concreto, en que hay más de una trama de respuesta para una trama de pregunta, como p.ej. cuando el maestro envía una operación cuya respuesta puede llevar al esclavo un tiempo elaborar. En estas situaciones el esclavo envía una primera respuesta indicando que aún no tiene los datos y tardará un tiempo en disponer de ellos, y otra segunda con los datos o confirmación de la operación.

Además las comunicaciones MODBUS se pueden realizar en modo ASCII o en modo RTU. En modo ASCII los bytes se envían codificados en ASCII, es decir, que por cada byte a transmitir se envían dos caracteres ASCII (2 bytes) con su representación hexadecimal (esto permite leer las tramas con un simple editor de texto). En modo RTU se envían en binario, tal cual. En el modo ASCII las tramas comienzan por 3AH (carácter ':'), y terminan en 0DH-0AH (CR LF Carrier Return Line Feed) y cada byte se envía como dos caracteres ASCII. En modo RTU no se utiliza indicador de inicio y final de trama. (Ver Tabla 3)

Tabla 3. Modos de transmisión MODBUS

	Modo ASCII	Modo RTU
Caracteres	ASCII '0'...'9','A'....'F'	Binario 0...255
Comprobación Error	LRC Longitudinal Redundancy check	CRC Cyclic Redundancy Check
Inicio de trama	Carater ':'	3.5 veces t de carácter
Final de trama	Character CR/CL	3.5 veces t de carácter
Distancia max. entre caracteres	1 seg	1.5 veces t de carácter
Bit de inicio	1	1
Bits de datos	7	8
Paridad	Par / Impar / Ninguna	Par / Impar / Ninguna
Bits de parada	1 si hay paridad 2 si ninguna	1 si hay paridad 2 si ninguna

Fuente: Autores

- **Campos de las tramas MODBUS**

El número de campos de las tramas MODBUS varía ligeramente dependiendo de si utilizamos la codificación ASCII o RTU:

Codificación ASCII (formato texto):

-Inicio de trama: 2 caracteres ASCII (que representan 1 byte) codificando el caracter ":" (0x3A)

-Nº Esclavo: 2 caracteres ASCII (que representan 1 byte) codificando la dirección del esclavo destino (u origen) de la trama

-Código Operación: 2 caracteres ASCII (representan 1 byte) con el código de operación

-Dirección, datos y subfunciones Datos: con los parámetros necesarios para realizar la operación.

-LRC(16): H L

-Final de trama: 4 caracteres ASCII (que representan 2 bytes) con los caracteres CR (0x0D) - LF (0x0A)

Codificación RTU (en el formato binario, el inicio de trama debería ser tras 3.5 tiempo de carácter):

-Nº Esclavo: 1 byte con la dirección del esclavo destino (u origen) de la trama

-Código Operación: 1 byte con el código de operación

-Subfunciones Datos: con los parámetros necesarios para realizar la operación.

-CRC(16): H L.

- **Descripción de los campos de las tramas MODBUS**

- Número de Esclavo (1byte): En el caso de las tramas enviadas por el máster, el campo de número de esclavo indica la dirección del destinatario de esta trama. Permite direccionar hasta 247 esclavos, con las direcciones de 1d a 247d (0x00 a 0xF7). El 0x00 es para los mensajes de Broadcast, así el primer esclavo comienza con la dirección 1 (de 1 a 247). En el caso de las tramas enviadas por los esclavos, este byte sirve para indicar al máster a quién pertenece la respuesta. Es decir, cada vez que un esclavo responde, sitúa su propia dirección en el byte de dirección lo que permite saber al maestro a que equipo corresponde cada

respuesta. Las tramas broadcast, no tienen asociada respuesta, y algunas implementaciones de MODBUS no admiten la trama de broadcast.

- Código de Operación o Función (1byte): Indica el tipo de operación que queremos realizar sobre el esclavo. Las operaciones se pueden clasificar en dos tipos:

    - De lectura / escritura en memoria: para consultar o modificar el estado de los registros del mapa de memoria del esclavo.

    - Órdenes de control del esclavo: para realizar alguna actuación sobre el esclavo.

El código de operación puede tomar cualquier valor comprendido entre el 0 y el 127 (el bit de más peso se reserva para indicar error). Cada código se corresponde con una determinada operación. Algunos de estos códigos se consideran estándar y son aceptados e interpretados por igual por todos los dispositivos que dicen ser compatibles con MODBUS, mientras que otros códigos son implementaciones propias de cada fabricante. Es decir que algunos fabricantes realizan implementaciones propias de estos códigos “no estándar”.

Es también mediante el código de función que el esclavo confirma si la operación se ha ejecutado correctamente o no. Si ha ido bien responde con el mismo código de operación que se le ha enviado, mientras que si se ha producido algún error, responde también con el mismo código de operación pero con su bit de más peso a 1 (0x80) y un byte en el campo de datos indicando el código de error que ha tenido lugar.

- Dirección, datos y subfunciones (n bytes): Este campo contiene la información necesaria para realizar la operación indicada en el código de operación. Cada operación necesitará de unos parámetros u otros, por lo que el número de bytes de este campo variará según la operación a realizar. En el caso del esclavo, este puede responder con tramas con o sin campo de datos dependiendo de la operación. En los casos en que se produzca algún error es posible que el esclavo responda con un byte extra para especificar el código de error.

Al establecer la dirección de una variable u otro elemento en el mapa de direcciones MODBUS, direccionamos con 1 unidad menos a la del registro al que queremos acceder, de manera que si p.ej. quisiéramos acceder al relé 127d, lo haríamos situando el valor 126d en el byte del campo de dirección.

- Control de errores LRC o CRC: Se utiliza un sistema de detección de errores diferente dependiendo del tipo de codificación utilizado (ASCII o RTU). En el caso de la codificación ASCII es el checksum (o Longitud Redundancy Check LRC) en módulo 16 expresado en ASCII (2 caracteres representan 1 byte), sin considerar el

":" ni el “CR LF” de la trama. En la codificación RTU se utiliza el método de CRC (Cyclical Redundancy Check) codificado en 2 bytes (16 bits).

Para calcular el CRC se carga un registro de 16 bits todo con ‘1’s, se hace OR con cada uno de los caracteres de 8 bits con el contenido de cada byte y el resultado se desplaza una bit a la izquierda insertando un 0 en la posición de menos peso (la de la derecha). El de la izquierda se extrae y se examina: si es 1 se vuelve a hacer OR con un valor prefijado, si es 0 no se hace ninguna OR... y el proceso se repite hasta que se han hecho los 8 shifts del byte.

- **Modo de Transmisión ASCII**

Teniendo presente la descripción de cada uno de los campos que debe contener una trama Modbus, en la figura 3 se muestra la trama correspondiente a un mensaje utilizando el modo de transmisión ASCII:

Figura 3. Trama en el modo de transmisión ASCII

INICIO	DIRECCIÓN	FUNCIÓN	DATOS	LCR	FIN
1 Caracter :	2 Caracteres	2 Caracteres	n Caracteres	2 Caracteres	2 Caracteres

Fuente: Autores

En modo ASCII los mensajes comienzan con el caracter dos puntos (**:** ASCII 3A hex.), y terminan con CR/LF (retorno de carro/ avance de línea), (ASCII 0D y 0A hex. respectivamente) que delimitan el inicio y el final de cada trama. Todos los dispositivos en red monitorizan el bus continuamente para detectar caracteres ASCII válidos, y una vez detectado el caracter dos puntos sabrán que alguno de los dispositivos ha empezado la transmisión de un mensaje y deberán estar presto para analizar el siguiente campo que corresponde a la dirección para determinar el destino del mensaje. Si esta dirección corresponde con la de algún esclavo, éste deberá iniciar la captura de la trama y posteriormente ejecutar la acción solicitada.

Bits Necesarios por Caracter a Transmitir

Como se mencionó anteriormente los caracteres para el modo de transmisión ASCII tienen una longitud de diez bits. Estos diez bits se distribuyen como sigue:

· 1 bit de inicio

· 7 bits de datos, el bit menos significativo se envía primero

· 1 bit para paridad, si se usa bit de paridad

· 1 bit de parada si se usa paridad y dos bits si no se usa paridad

El primer bit indica el inicio de la transmisión de un nuevo caracter de la trama. Es útil en la transmisión serial asíncrona para advertir a los dispositivos receptores que se ha puesto un nuevo caracter en el bus. Se debe hacer la validación de este bit para descartar que se trate de ruido en el canal que pueda alertar a los dispositivos para la recepción de un nuevo caracter inexistente.

Seguido a esto están siete bits para la información contenida en el carácter enviado.

El hecho de que se envíen solo siete bits de información en un caracter en el modo de transmisión ASCII tiene algunas implicaciones. Por ejemplo, se observa que se deben enviar dos caracteres para indicar la dirección de esclavo. Si un mensaje es enviado al esclavo 06 se deberá enviar en este campo el cero en el primer caracter y el seis en el segundo, que en el código ASCII son el 48 y 54 en decimal respectivamente.

Este último aspecto hace que el aprovechamiento del canal para el flujo de información entre los dispositivos en red sea menor cuando se utiliza este modo de transmisión, comparándolo con el modo de transmisión RTU. Cabe aclarar que la afirmación anterior se hace suponiendo que la velocidad de transferencia de datos es la misma para comparar la eficiencia de los dos modos de operación.

Una ventaja de importancia cuando se utiliza este modo de transmisión es que permite tiempos hasta de un segundo entre caracteres sin que el dispositivo receptor interprete que ha ocurrido un error en la comunicación. Se verá que en el modo RTU el tiempo que transcurre en la recepción de caracteres es de vital importancia para determinar el inicio y fin de una trama libre de errores.

- **Modo de Transmisión RTU**

En la figura 4 se muestra la estructura de un mensaje enviado utilizando este modo de transmisión. En esta figura se observa que los mensajes comienzan con un intervalo de silencio de al menos 3.5 veces el tiempo necesario para enviar un carácter lo cual es mostrado como T1-T2-T3-T4. Después de este silencio el primer campo transmitido es la dirección del esclavo, cuya longitud es de ocho

bits. De igual forma se cuenta con ocho bits para enviar el código de función, múltiplos enteros de ocho bits para los datos si son necesarios y dieciséis para el CRC. La trama terminará con un silencio de al menos 3.5 veces el tiempo necesario para enviar un carácter.

Figura 4. Trama en el modo de transmisión RTU

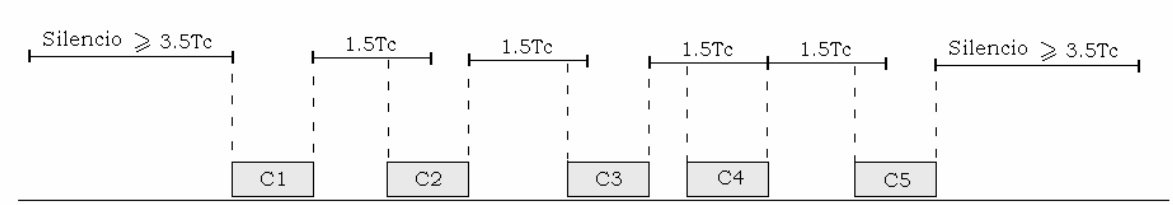
INICIO	DIRECCIÓN	FUNCIÓN	DATOS	CRC	FIN
T1-T2-T3-T4	8 bits	8 bits	n*8 bits	16 bits	T1-T2-T3-T4

Fuente: Autores

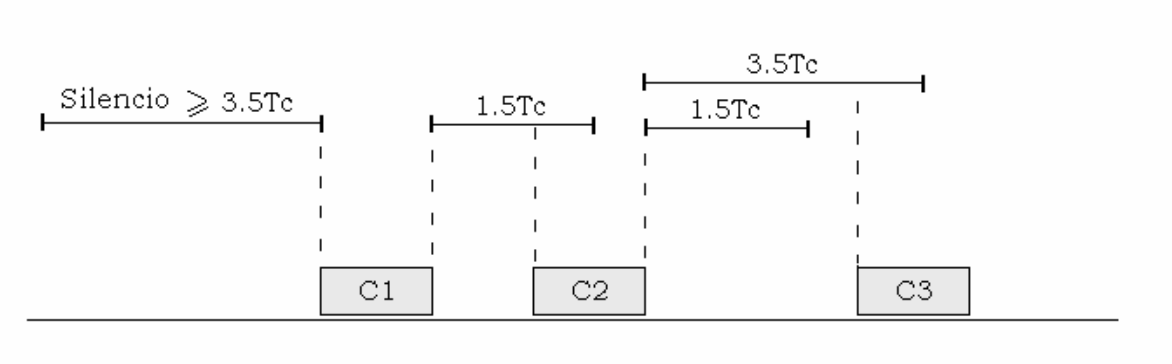
Además del tiempo que limita el inicio y el fin de una trama, en el modo de transmisión RTU se debe tener en cuenta el tiempo que transcurre entre la llegada de caracteres consecutivos. Este tiempo se ha definido para que sea máximo de 1.5 veces el tiempo necesario para enviar un carácter. Si entre el fin de un carácter y el comienzo de otro transcurre un tiempo mayor que 1.5Tc y menor que 3.5Tc se producirá una situación de error en la transmisión y el dispositivo receptor debe ignorar la trama. Cuando se produce este error los esclavos no deberán enviar ningún mensaje de respuesta.

En la figura 5 se muestra en forma gráfica diversas situaciones que se pueden presentar en la recepción o envío de mensaje entre dispositivos. En esta figura 1.5Tc representa 1.5 veces el tiempo necesario para enviar un carácter, 3.5Tc representa 3.5 veces el tiempo necesario para enviar un carácter y los bloques etiquetados como C1, C2,... son los caracteres que conforman el mensaje puesto en el bus. La figura 5a representa una situación normal en el intercambio de un mensaje; se observa que el inicio del primer carácter empieza cuando ha transcurrido un silencio de por lo menos 3.5Tc y que los restantes caracteres empiezan a llegar antes que ocurra 1.5Tc medido a partir del punto final del carácter anterior. También se observa que desde el final de C5 transcurre un tiempo mayor o igual a 3.5Tc, lo que indica que C5 es último carácter del mensaje. Cualquier carácter que se reciba después de este último silencio será el primero de otro mensaje puesto en el bus por algún dispositivo de la red.

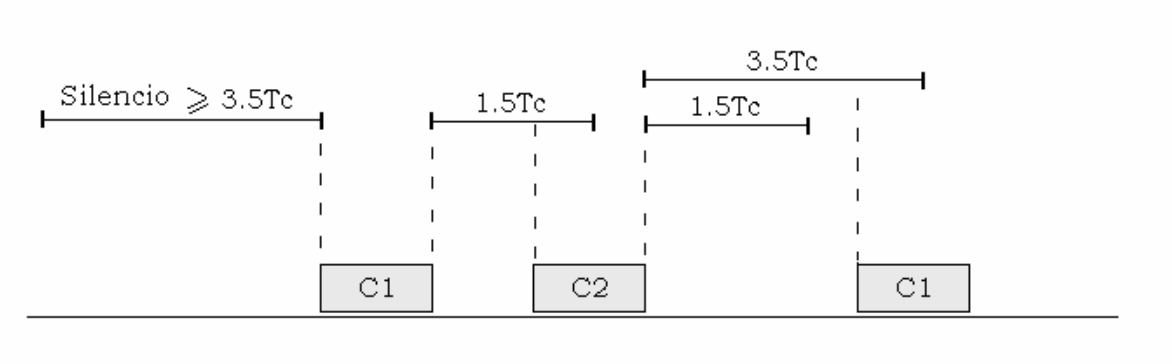
Figura 5. Situaciones posibles en el intercambio de mensajes entre dispositivos conectados en red en el modo de transmisión RTU.



a) situación normal



b) situación de error



c) situación de error

Fuente: UIS

La figura 5b representa una situación de error. Se observa que C1 y C2 se han recibido en tiempo permitidos, pero C3 se ha recibido después de que ha transcurrido 1.5Tc pero antes de que transcurra 3.5Tc. En este caso C3 no se puede entender como un carácter perteneciente al mensaje en recepción que va

después de C2 ni el primer carácter de otro mensaje. Si esto ocurre, el dispositivo receptor debe continuar monitorizando el bus para poder detectar un silencio de por lo menos 3.5Tc y recibir el siguiente carácter como el primero del siguiente mensaje.

De igual forma la figura 5c representa otra situación de error. Se observa que algún dispositivo ha intentado enviar el primer caracter de un mensaje (C1) antes de que transcurra un silencio de por lo menos 3.5Tc medidos a partir del fin de C2.

Como en el caso anterior C1 no se puede entender como un carácter consecutivo a C2 que pertenece al mensaje en recepción ni como el primer carácter de un nuevo mensaje. Otro posible caso es que C1 del nuevo mensaje llegue antes de que se cumpla 1.5Tc de haber arribado el último caracter del mensaje anterior. Si este es el caso, el dispositivo receptor concatenará ese o esos caracteres a los caracteres del mensaje anterior y tendrá como es de esperarse un mensaje errado. Los dispositivos que reciban este mensaje detectarán el error cuando hagan el cálculo del CRC y hagan la comparación con el supuesto CRC que llega en el mensaje en los campos destinados para el chequeo de errores.

Bits Necesarios por Caracter a Transmitir

Los caracteres para el modo de transmisión RTU tienen una longitud de once bits.

Estos once bits se distribuyen como sigue:

· 1 bit de inicio

· 8 bits de datos, el bit menos significativo se envía primero

· 1 bit para paridad, si se usa bit de paridad

· 1 bit de parada si se usa paridad y dos bits si no se usa paridad

Un ejemplo de petición se muestra en la tabla 4; en ésta se observa cómo se construye la consulta en el formato ASCII y RTU. El maestro le pide al esclavo número 06 que envíe tres registros a partir del registro 006B (107d).

Tabla 4. Consulta del maestro con formatos ASCII y RTU.

Nombre del campo Consulta	Ejemplo (Hex)	ASCII Caracteres	RTU Campo de 8 Bit
Cabecera		: (colon)	
Dirección de esclavo	06	0 6	0000 0110
Función	03	0 3	0000 0011
Dirección de inicio (Hi)	00	0 0	0000 0000
Dirección de inicio (Lo)	6B	6 B	0110 1011
No. de Registros (Hi)	00	0 0	0000 0000
No. de Registros (Lo)	03	0 3	0000 0011
Chequeo de errores		LRC (2 caracteres)	CRC (16 Bits)
Fin de la trama		CR LF	
	Total Bytes :	17	8

Fuente: Autores

Es evidente en esta tabla la diferencia en bytes que existe en una misma trama enviada utilizando cada uno de los dos métodos de transmisión utilizadas en Modbus.

De igual forma en la tabla 5 se observa un ejemplo de lo que puede ser una respuesta del esclavo 06 a la petición anterior para cada modo de transmisión.

En el campo de dirección se envía la dirección del esclavo que responde, en este caso 06; igualmente en el campo código de función se coloca el número de la función que el esclavo ha ejecutado. En la respuesta para esta función se debe colocar en el campo contador de bytes, que indica cuantos bytes corresponden al estado de los registros solicitados por el maestro. En este caso, en el campo contador de bytes se encuentra un seis ya que se ha solicitado tres registros y cada registro consta de 16 bits (2 Bytes).

Tabla 5. Respuesta de Esclavo con formatos ASCII y RTU

Nombre del campo Respuesta	Ejemplo (Hex)	ASCII Caracteres	RTU Campo de 8 Bit
Cabecera		: (colon)	Ninguno
Dirección de esclavo	06	0 6	0000 0110
Función	03	0 3	0000 0011
Contador de bytes	06	0 6	0000 0110
Dato Hi	02	0 2	0000 0010
Dato Lo	2B	2 B	0010 1011
Dato Hi	00	0 0	0000 0000
Dato Lo	00	0 0	0000 0000
Dato Hi	00	0 0	0000 0000
Dato Lo	63	6 3	0110 0011
Chequeo de errores		LRC (2 chars)	CRC (16 Bits)
Fin de la trama		CR LF	Ninguno
	Total (Bytes):	23	11

Fuente: Autores

- **Descripción de los códigos de operación o función más frecuentes**

Función 1 o 2 (1 Read Coil Status - 2 Read Input Status):

Permite realizar la lectura del estado de las digital inputs (@1XXXX el comando 2-Read input status) o digital outputs (@0XXXX el comando 1-Read Coil Status). Para ello el maestro solicita el número de bits que desea leer a partir de una determinada dirección. Cada dirección se corresponde con un registro de 1 bit con el estado de la entrada digital. El esclavo responde indicando el número de bits que retorna y sus valores. En la trama de respuesta se aprovechan todos los bits del byte, y puede haber hasta 256 bytes.

Función 3 o 4 (3 Read Holding Registers – 4 Read Input Registers):

Permite realizar la lectura del valor de las analog inputs (@4XXXX el comando 3 Read Holding Registers) o analog outputs (@3XXXX el comando 4 Read Input Registers). El máster indica la dirección base y número de palabras a leer a partir de esta, mientras que el esclavo indica en la respuesta el número bytes retornados, seguido de estos valores. Aunque en realidad se está escribiendo en el rango de registros o valores numéricos, los registros son direccionados a partir de la dirección 0 (así el registro @40001 se direcciona 0)

Función 5 (Force Single Coil):

Permite modificar el estado de una DO del esclavo (mando o relé). Es decir mediante este comando podemos modificar algún bit de alguna de las variables internas del esclavo u ordenar la ejecución o activación de un mando. Actúa sobre la zona de memoria de los DOs @0XXXX. El Maestro especifica la dirección del bit o mando que quiere modificar seguido de 0x00 para ponerlo a 0 o 0xFF para ponerlo a 1. El esclavo responde con una trama similar indicando la dirección que ha modificado y el valor que ha establecido en el bit o mando.

6.4 MARCO INSTITUCIONAL

El desarrollo del proyecto se llevará a cabo en la Empresa Agroavícola San Marino en la granja piscícola ubicada en la vereda Paradero 1 del municipio de Flandes – Tolima.

El 16 de abril de 1996 bajo la dirección del Señor Sebastiano Carbone Bellini con una visión hacia el sector avícola se creó AGROAVICOLA SAN MARINO S.A. La primera actividad que desarrollo la Compañía fue la compra y venta de pollito de un día con un lote de comerciales raza Hy line. En febrero del año 1997 se iniciaron operaciones con pollo de engorde en Santander, en la Mesa de los Santos.

Debido al crecimiento y al reconocimiento de la marca en el año 1999 se construye la primera planta de Incubación cinco máquinas incubadoras y cinco máquinas nacedoras realizando el primer cargue en abril del año 2000. El 1 de mayo de 2008 llega el primer lote de 10578 Hembras y 2027 Machos de Reproductoras Abuelas de la línea Hybro procedentes de Holanda. El 1 de abril de 2009 se traslada el proceso de Abuelas reproductoras a las instalaciones de la granja La Toscana de Ibagué. Con un lote de 7126 hembras y 1274 machos de CobbAvian 48.

En el año 2006 se inició el proyecto de construcción de lagos para reproducción, levante y reversión de alevinos; en ese mismo año, incursionamos con la línea de pollo en canal a través de una alianza estratégica con pollo Piku, teniendo un sacrificio mensual de 70.000 aves alcanzando un crecimiento entre el año 2006 al 2009 de 15.000 mil aves para tener un sacrificio mensual de 220.000 mil aves para el año 2009.

Actualmente Agroavicola Sanmarino S.A., es una empresa del sector primario de ámbito pecuario, dedicada a la producción y comercialización de pollitos de un día CobbAvian 48, venta de pollita de un día Babcock, levante de ponedoras comerciales, alevinos, pollo en canal, venta de medicamentos veterinarios de la línea Aurofarma, producción y venta de huevo fértil, venta de subproductos avícolas como gallinaza, pollinaza y huevo comercial.

Sanmarino S.A., cuenta a nivel nacional con una planta de abuelas, siete plantas de incubación, veinticinco granjas reproductoras, veintidós granjas de levante comercial, once granjas de engorde de pollo, cuarenta y dos estanques para reproducción, levante y reversión de alevinos; y una planta de proceso de pollo en canal.

6.4.1 Misión de la empresa

Producir y Comercializar bienes dirigidos al sector avícola y piscícola, que generen satisfacción y confianza en nuestros clientes, mediante la adopción de nuevas tecnologías y el acompañamiento de un equipo humano altamente calificado.

6.4.2 Visión al año 2020

Ser una empresa del sector agropecuario reconocida nacional e internacionalmente por nuestros productos y servicios de valor agregado, apoyados en la mejora continua de los procesos.

7. METODOLOGIA

7.1 PARTICIPANTES

El proyecto cuenta con el apoyo del señor Harvey Guerrero, técnico piscicultor de Agroavícola San Marino y de los estudiantes de Ingeniería Electrónica Dora Alejandra Parra Hernández y Miguel Suárez Sierra pertenecientes a la institución educativa ITFIP.

7.2 MATERIALES Y DISPOSITIVOS

Con el fin de establecer los valores de las condiciones físico-químicas del agua es necesaria la utilización de los siguientes elementos mostrados en la figura 6:

- Termómetro digital marca Cooper Modelo SRH77A
- Kit de análisis manual de la calidad del agua en piscicultura marca Hach Modelo FF-1A

Figura 6. Elementos de medición.

Fuente: Agroavicola San Marino

7.3 PROCEDIMIENTO

Para el desarrollo del proyecto se parte del conocimiento acerca de la operación actual del proceso, realizada entre los días 1 al 24 de octubre de 2015, que por lo analizado es manual, y la infraestructura disponible, identificando y analizando sus

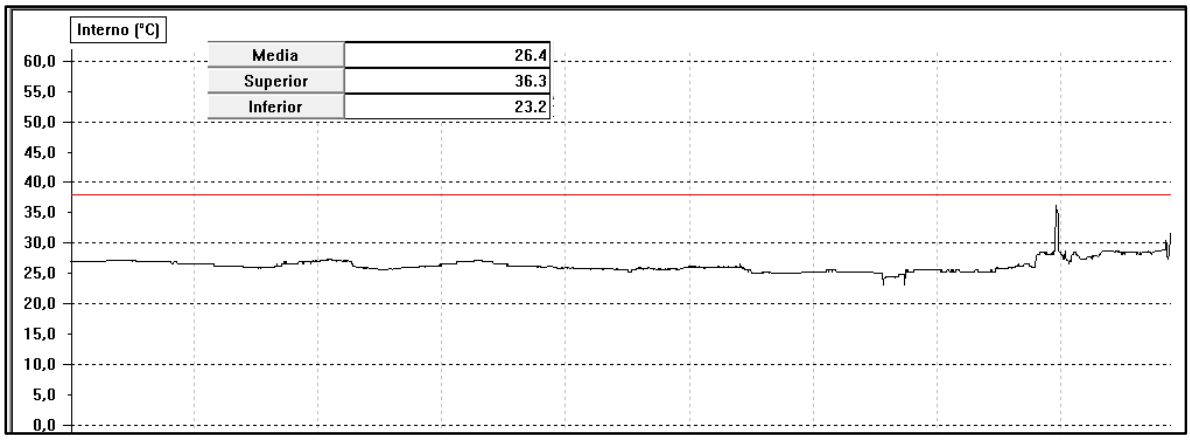
limitaciones y singularidades, con el objetivo de elaborar un diseño para la monitorización de las variables que supla las expectativas y necesidades de los operarios, del técnico piscicultor, de las directivas de la empresa y en general de todos los involucrados.
Se evidencia la no existencia de un control apropiado de las condiciones físico-químicas relevantes del agua que permite la incubación de alevinos de mojarra roja, ya que sólo se limita a la observancia esporádica, por parte del operador, de la temperatura por medio de un termómetro de mercurio y la realización de muestreo poco frecuente de las otras características del agua, como son el pH, dureza y oxígeno disuelto. Esta situación de bajo control del proceso de incubación no permite establecer reglas claras de condiciones apropiadas para el mejor desempeño de la natalidad de los alevinos, al igual, no deja determinar patrones de trazabilidad que oriente hacia el futuro razonamientos de cambios en la producción y comportamientos fuera de los esperados.

7.3.1 Realización de mediciones

- Temperatura

Con el objeto de obtener valores confiables de los parámetros a analizar se llevaron a cabo lecturas de temperatura con el registrador de temperatura y el termómetro digital como se muestra en la figura 7, obteniéndose valores comprendidos entre 23.2 a 36.3 grados centígrados con un promedio de 26.4 grados centígrados, que están claramente por fuera de las especificaciones para la obtención de un buen resultado.

Figura 7. Gráfica registrador temperatura



Fuente: Agroavicola San Marino

Asimismo, se obtuvieron resultados de las condiciones químicas del agua realizando las pruebas con el kit de Hach (ver Figura 8) siguiendo los procedimientos señalados en el manual del usuario.

- pH

Las lecturas de pH oscilaron entre 6 y 9 realizando una muestra cada semana verificándose que el valor mínimo se presenta cuando existe un recambio de agua.

Figura 8. Toma de pH

Octubre 06: pH 6
Octubre 15: pH 8
Octubre 21: pH 9
Octubre 24: pH 6

Fuente: Agroavicola San Marino

Los valores obtenidos tienen una diferencia frente al requerimiento en su valor mínimo ya que debe ser de 7.

- Oxígeno disuelto

En las cuatro muestras que se llevaron a cabo se encontró que el valor mínimo estuvo en 3 mg/L debido a una falla en el sistema de aireación y el máximo en 12 mg/L, remitirse a l figura 9.

Figura 9. Toma de OD

Octubre 06: OD 8mg/L
Octubre 15: OD 10mg/L
Octubre 21: OD 3mg/L
Octubre 24: OD 12mg/L

Fuente: Agroavicola San Marino

La periodicidad de las muestras se estableció por la experiencia del personal a cargo de la piscícola quienes han determinado realizarlas cada semana debido al tiempo que requiere llevar a cabo los análisis que puede ser destinado a otra labor y la variación en el resultado que presenta, sin embargo, fallas como la ocurrida en el sistema de aireación, a pesar que fue detectada a tiempo, señalan la necesidad de una medición automatizada que lleve a cabo registros cada hora.

7.4 ANALISIS ESTADO DEL ARTE

7.4.1 Proyectos universitarios

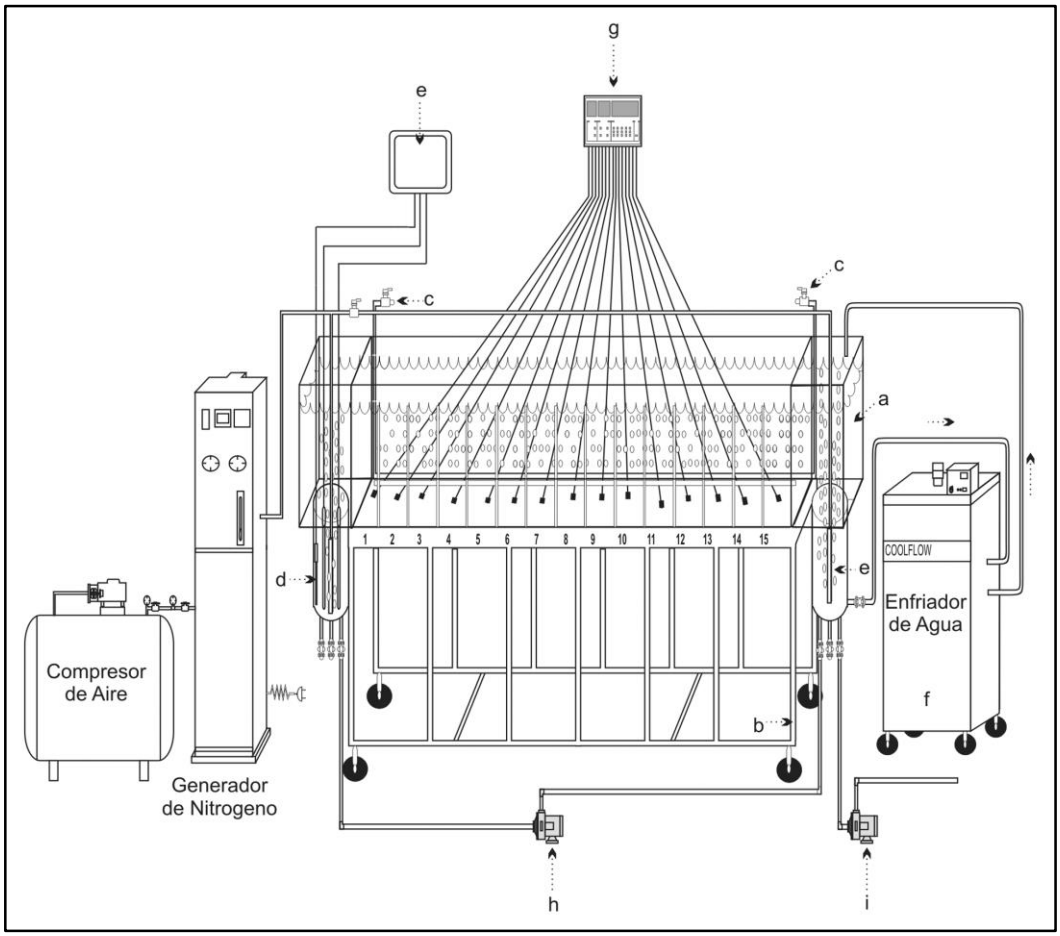
En la documentación estudiada se encuentra que en la Universidad Tecnológica de la Mixteca, Oaxaca, (México), han implementado un sistema que brinda la posibilidad de realizar el monitoreo y control tanto de la temperatura y el oxígeno disuelto para llevar a cabo estudios en eco-fisiología[16], el cual está constituido por una unidad electrónica para el control de la temperatura, dicho dispositivo centra su diseño en un microcontrolador que recibe los datos de un termistor y son mostrados en un display de tres dígitos.

El circuito electrónico, previo a dar comienzo a la medición, ejecuta una calibración con el objeto de brindar mayor confiabilidad en la medida. El segmento

[16] Estudio de cómo los animales se enfrentan a los problemas funcionales que crea cada entorno y cómo explotan las oportunidades energéticas y reproductoras de cada ambiente.

del dispositivo que permite manejar niveles de oxígeno disuelto en el agua cuenta con un tubo de PVC y una bomba de vacío que permite llevar a cabo la desgasificación. Ver figura 10.

Figura 10. Vista general del sistema

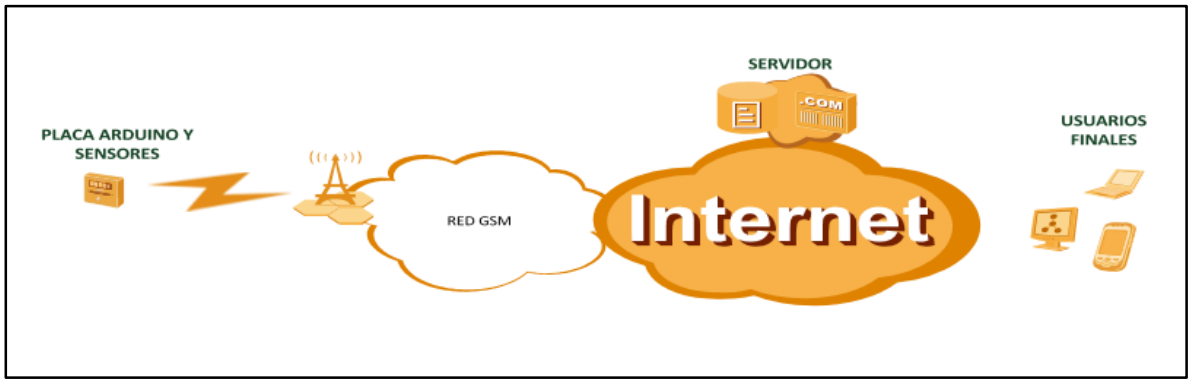


a, estanque; b, mesa metálica con ruedas y tornillos para equilibrar el estanque; c, válvula de paso del difusor que distribuye el aire en el interior del estanque; d, cámara con calentadores de titanio y termistor; e, piedra de aireación; f, intercambiador de calor para enfriar el agua de la cámara fría; g, sensor remoto de temperatura (SR 630) con termistores para cada una de las 15 cámaras virtuales del gradiente; h, motobomba de 1/20 HP para recircular el agua del estanque; i, motobomba de 1HP con turbina magnética que impulsa el agua del estanque al desgasificador a través de la válvula solenoide ubicada en la parte superior del desgasificador.

Fuente: Universidad Tecnológica de la Mixteca

De igual forma se conoce que en la tesis de grado “Diseño e implementación de un prototipo para la medición de calidad del agua y control de la oxigenación en forma remota orientado a la producción acuícola” presentada en la Universidad Politécnica Salesiana de Guayaquil (Ecuador), por los ingenieros Diana Isabel Rivera y Eddy Antonio Yépez, establecen la viabilidad de realizar mediciones de temperatura, pH, oxígeno en estanques de peces y enviarlos por GSM a una base de datos como se puede apreciar en la figura 11.

Figura 11. Esquema funcionamiento



Fuente: Universidad Politécnica Salesiana de Guayaquil

Con el fin de establecer las opciones existentes en el mercado para el manejo y control de los sistemas de recirculación de agua, se llevó a cabo consultas en revistas de acuicultura e internet con los siguientes resultados:

7.4.2 Empresa Innovaqua

Desarrollan un sistema integral de control de parámetros para acuicultura e industrias del agua denominado MIRANDA. Está formado básicamente por dos unidades complementarias e independientes: MIRANDA ACB que conecta y controla los sensores y dispositivos, procesando los datos de forma autónoma y MIRANDA SDA, que permite un acceso seguro por navegador a esta información desde múltiples ubicaciones, como se puede apreciar en la figura 12.

Figura 12. Sistema de control Miranda

Fuente: INNOVAQUA

En la arquitectura de Miranda, los controladores, mostrados en la figura 13, son elementos autónomos y pueden trabajar independientemente de Miranda SDA y Miranda ACB.

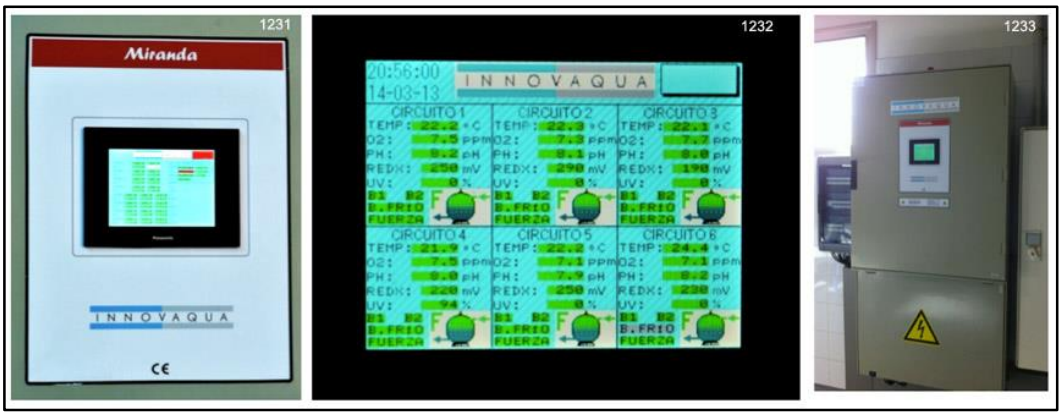
Figura 13. Controladores MIRANDA

Fuente: INNOVAQUA

El segundo nivel es el Miranda ACB (Automation Control Box), como se muestra en la figura 14, que incluye un PLC industrial con pantalla táctil a color. Utilizando las sondas apropiadas, puede leer los distintos parámetros a controlar (pH, oxigeno, temperatura, etc.), actuando sobre los dispositivos (bombas, válvulas, etc.) para mantener sus niveles dentro de los rangos deseados (ver figura 15).

Miranda ACB es un elemento autónomo y puede trabajar independientemente de Miranda SDA. Esta arquitectura incrementa la fiabilidad del sistema minimizando los riesgos por accidentes o pérdidas. Miranda ACB puede ser conectado a cualquier dispositivo estándar con salidas 4-20 mA, 0-20 mA y 0-10 Voltios, o comunicaciones digitales.

Figura 14. Miranda ACB



Fuente: INNOVAQUA

El tercer nivel es el Miranda SDA (Supervisor Data Acquisition), mostrado en la figura 16, un SERVIDOR WEB basado en LINUX (distribución DEBIAN), diseñado para una durabilidad operacional extrema:

- Motor de base de datos MySQL.

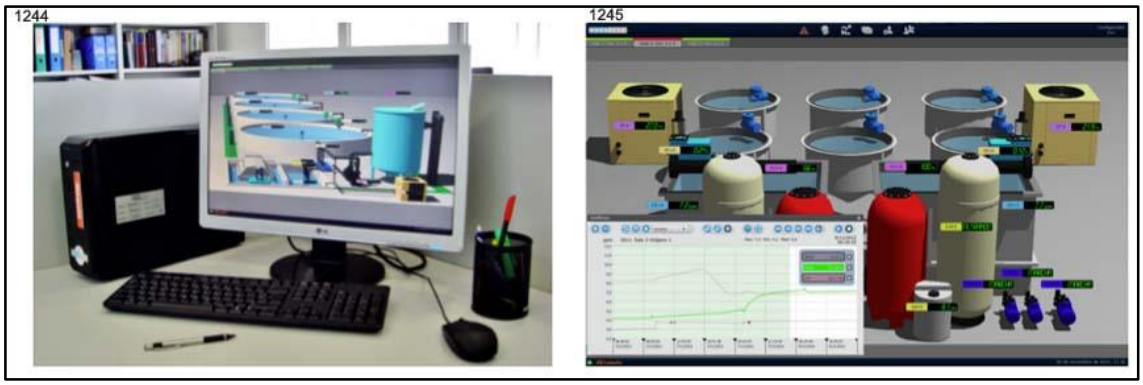
- Módulo de comunicaciones con dispositivos de campo específicamente desarrollado por INNOVAQUA

- Servidor web APACHE con tecnología php, al que puede accederse desde un explorador web estándar y de manera simultánea desde múltiple dispositivos, con control de acceso por usuario/contraseña.

Figura 15. Equipo Miranda

Fuente: INNOVAQUA

Figura 16. Miranda SDA



Fuente: INNOVAQUA

Miranda SDA está instalado en un robusto equipo conectado a los controladores mediante cable, fibra óptica o radio (red de comunicaciones), registra los datos de campo y los incorpora a la base de datos, permitiendo la ejecución de órdenes y programas de diferentes sistemas de control. Incluye las siguientes funciones:

- Presentación de parámetros en pantalla en tiempo real

- Presentación del estado de dispositivos en pantalla en tiempo real

- Información con códigos de colores para una rápida interpretación

- Gestión de datos históricos mediante representación gráfica

- Exportación de la información mediante archivos estándar CSV, compatibles con la mayoría de las aplicaciones de manejo de datos

- Gestión de alarmas a través de SMS y/o correo electrónico

- Actuación sobre dispositivos de modo local (LAN) o remoto (INTERNET)

- Copia de seguridad automática en una tarjeta extraíble SD estándar

- Sistema Key-code, permitiendo reemplazar de forma inmediata un dispositivo defectuoso por otro nuevo al mantener toda la configuración.

7.4.3 Empresa Pentair

Comercializan el producto Aquatic Habitats® C-Series PLCs que se presenta en la figura 17, es un control de tecnología de última generación, para el monitoreo, multiparamétrico de calidad del agua, control de la dosificación, alarmas y gestión de datos en un paquete fácil de usar.

Figura 17. PLC Pentair

Fuente: PENTAIR

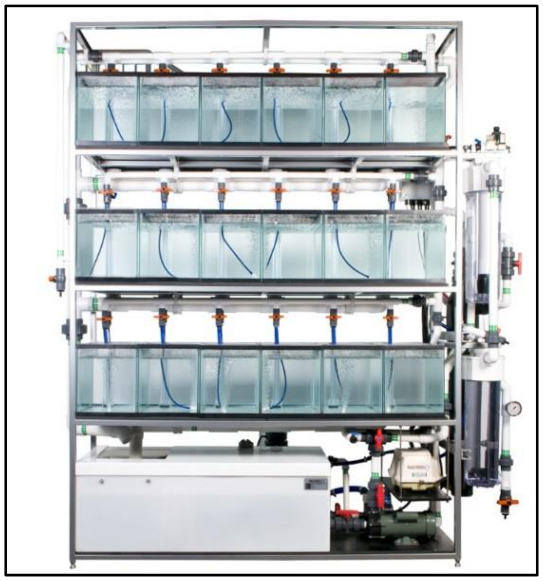
La pantalla es de 256 colores que hace que sea fácil identificar cada lectura de parámetros de un vistazo, y cada parámetro monitorizado tiene control de resolución programable. Si desea acceder a los datos históricos, el PLC tiene

1.000 puntos de datos que se pueden grabar a intervalos definidos por el usuario. Además, el C351 hace más fácil el mantenimiento del sistema a través de avisos de servicio se puede configurar para los filtros desechables, lámparas UV y los cambios de medios de carbono. Ajustar la configuración para el cambio de agua automático, alarmas y seguridad de control de cierre. La configuración avanzada incluye brillo de la pantalla, período de calentamiento UV, alarma de mantenimiento en desconexión y períodos de muestreo.

Por otra parte, es posible obtener el sistema completo para el manejo de la incubación denominado G-Hab, ver figura 18, que ofrece las siguientes características:

- Filtración de gran tamaño, de 5 etapas, garantiza una calidad óptima del agua.
- Alarma de Nivel de agua bajo con controlador lógico programable opcional (PLC)
- Panel de control.
- La aireación es proporcionada a cada tanque individual.
- Resistente, marco de acero inoxidable 316L con pies de nivelación.
- Bypass en la filtración que permite el servicio mientras el sistema funciona.
- Cambio automático de agua que ahorra tiempo y reduce la mano de obra.
- Bastidor de 3 niveles de tanques de vidrio que tiene capacidad para 10, 30 o 55 galones.

Figura 18. Sistema G-Hab Pentair



Fuente: PENTAIR

7.5 MONTAJES LABORATORIOS DE INCUBACIÓN EN COLOMBIA

Se hace una revisión de la tecnología empleada en otras empresas del sector para comprobar las implementaciones en cuanto a la incubación artificial se refiere obteniéndose los siguientes hallazgos.

7.5.1 Empresa Omegafish Acuicultura SAS

Está ubicado a 1200 msnm en la ciudadela de Bombona adscrita al municipio de Consaca Nariño-Colombia a 45 kilómetros de la capital del departamento (San Juan de Pasto) con una temperatura ambiente promedio de 23ºC. Produce alevines de tilapia roja Oreochromis sp; pez muy apetecido por productores acuícolas del suroccidente Colombiano, como se aprecia en la figura 19.

Figura 19. Laboratorio de incubación artificial de Omegafish

Fuente: OMEGAFISH

Con respecto a la utilización de nuevas tecnologías, apuesta por la recirculación de agua, incubación artificial; y también en lo que respecta al alimento alternativo como la tecnología de Bio-floc mediante una formulación con cero recambios de agua. Sin embargo, como puede apreciarse en las imágenes el control es llevado a cabo de manera manual.

7.5.2 Estación piscícola de Gigante

Esta estación de piscicultura tiene un área total 29 hectáreas y 5.5 ha en espejo de agua. Se encuentra ubicada en el municipio de Gigante, departamento del Huila, a 960 m.s.n.m. Geográficamente está localizada a los 0°55'16" de latitud Norte y 74'25"de longitud Este. Cuenta con una temperatura promedio de 24°C y pluviosidad de 1250 mm al año. Ver figura 20.

Figura 20. Laboratorio de incubación artificial de Gigante

Fuente: Estación piscícola de Gigante

En la estación de Gigante se vienen desarrollando programas de generación y validación de nuevas tecnologías, producción masiva de especies de peces nativas y exóticas, repoblamientos de cuerpos de agua y capacitación en piscicultura y procesamiento productos pesqueros. La producción de especies exóticas se realiza mediante dos métodos, el de reproducción natural de tilapia y el de incubación artificial. De igual manera, el manejo de la incubación es netamente manual.

7.6 DISEÑO DE LA SOLUCIÓN PROPUESTA

La incubación artificial de alevinos de mojarra roja de la granja piscícola de la empresa Agroavícola San Marino está desarrollada con un sistema de recirculación de agua cuyos parámetros físico-químicos son manejados hasta el momento de forma manual, lo que se busca con la propuesta de automatización es implementar el monitoreo de las variables para contar con herramientas de diagnóstico que permitan conocer las condiciones de producción de los peces teniendo en cuenta los siguientes parámetros:

- **Temperatura**. El rango óptimo de temperatura para la incubación de alevinos de tilapia roja fluctúa entre 28ºC y 32ºC. Los cambios de temperatura afectan directamente la tasa metabólica, mientras mayor sea la temperatura, mayor tasa metabólica y, por ende, mayor consumo de oxígeno.

- **pH**. Es la concentración de iones de hidrógeno en el agua, el rango apropiado debe oscilar entre 7.0 a 9.0.

- **Oxígeno disuelto**. Para mantener un cultivo exitoso de tilapia, los valores de oxígeno disuelto deberían estar entre 5 y 9 mg/L, el cual debería ser medido en la estructura de salida del estanque.

- **Nivel de agua**. Se comprueba en los tanques de recirculación para determinar pérdidas de agua, malfuncionamiento de las bombas o taponamiento de las tuberías.

A continuación se detallan los criterios de construcción del prototipo (ver figura 21) teniendo en cuenta la selección de elementos involucrados en el desarrollo de los circuitos electrónicos que conforman la parte de hardware, así como de la programación necesaria para el logro del monitoreo correspondiente al software.

Tanto el hardware como el software a emplear son de uso libre con Licencia Pública General de GNU.

Los elementos que integran el prototipo son los siguientes:

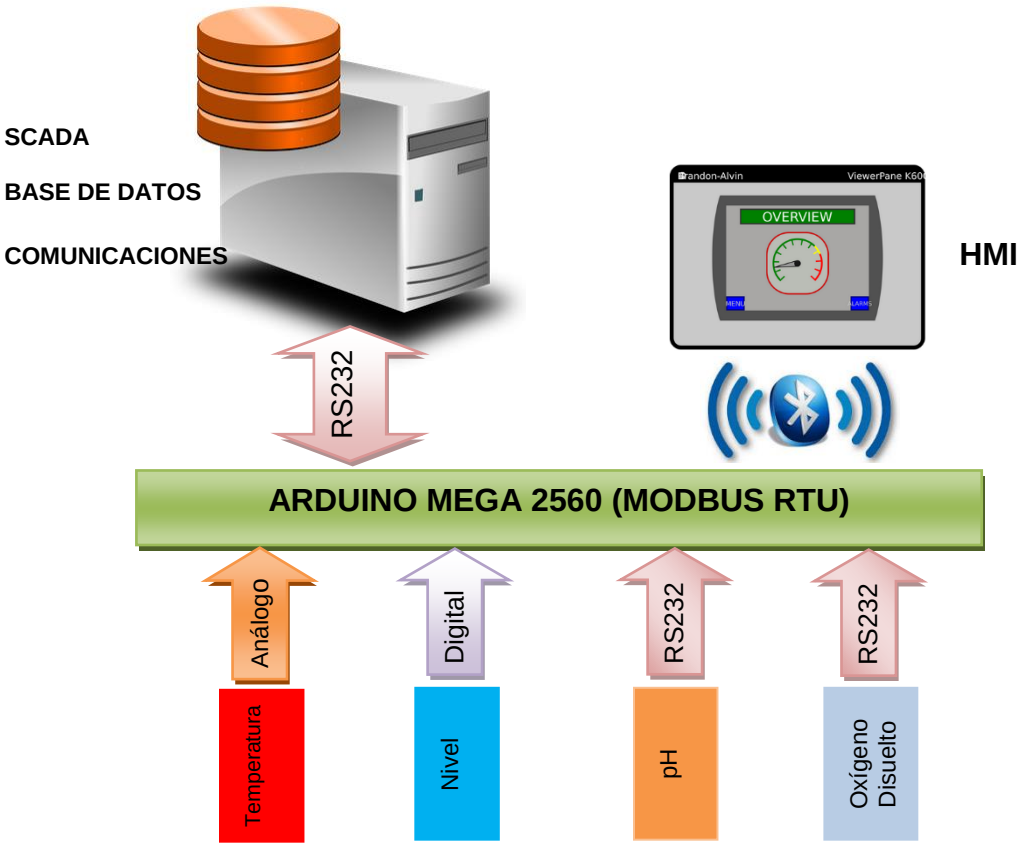
7.6.1 Unidad Terminal Remota (RTU)

Para llevar a cabo la adquisición de los datos de los sensores se emplea la placa Arduino Mega 2560 que es una plataforma de electrónica abierta flexible y fácil de

usar que se programa mediante un lenguaje de programación y un entorno de desarrollo basados en Wiring y en Processing respectivamente. Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software.

Entre las características de la tarjeta electrónica se encuentra que: posee una mayor capacidad de memoria en comparación a sus modelos antecesoras siendo esta de 256KB, posee 54 pines que pueden ser usados como entradas o salidas digitales y 15 de estos pines poseen características que permiten usar la funcionalidad PWM, además un grupo específico de pines puede ser utilizada para comunicación serial con periféricos externos (Serial 0 pin 0 y 1 Rx/Tx; Serial 1: 19 y 18 Rx/Tx; Serial 2: 17 y 16 Rx/Tx; Serial 3: 15 y 14 Rx/Tx.), también poseen 16 entradas analógicas con una resolución de 10bits.

Figura 21. Arquitectura del prototipo



Fuente: Autores

Con el objeto de enviar y recibir datos desde un SCADA es necesario llevar a cabo la implementación del protocolo MODBUS en modo RTU para el Arduino que se obtiene del *sketch* desarrollado por Juan Pablo Zometa para dispositivo esclavo.

7.6.2 Unidad Central Maestra (MTU)

El desarrollo del sistema SCADA se basa en el programa *"Acimut Monitoriza para Arduino"* que permite crear soluciones para la captura de información en procesos industriales o de cualquier otro ámbito. Con esa información se retroalimenta el proceso y se emplea como ayuda en la toma de decisiones.

Consta de tres partes:

- Un editor de proyectos en el que se definen todos los elementos a tratar.
- Un servidor que ejecutará el proyecto y se ocupará de las comunicaciones con los procesos (adquisición de datos, establecimiento de parámetros del proceso, base de datos, historial de alarmas, etc.)
- Un cliente que mostrará, de forma visual, la información de los procesos que se estén supervisando.

Monitoriza es un sistema flexible en cuanto a su configuración, ya que puede ejecutarse por varios usuarios simultáneamente. Puede funcionar sobre una infraestructura monopuesto o multipuesto, tanto en una red local como a través de puestos remotos conectados a través de internet.

7.6.3 Sensores

Se requieren de 3 sensores que se encargan de la medición constante de los parámetros de calidad de agua de la incubadora de peces, como es la temperatura, el pH y el OD. Para este propósito los sensores deben tener la capacidad de permanecer sumergidos dentro de un estanque con agua de forma constante y por largos periodos de tiempo. Y un último sensor con el cual se determina el nivel del mismo, el cual tendrá su montaje a una distancia prudente del estanque.

7.6.4 Pantalla HMI

En cualquier proceso industrial, es casi inevitable encontrar, al menos, una interfaz Hombre-Máquina (Human-Machine Interface, HMI) que ayude a los operarios a

monitorear y controlar el funcionamiento de un equipo, por tal motivo, el prototipo hará uso de una Tablet en la cual se implementa una aplicación desarrollada en Android llamada *"HMI Controller for Arduino"* que permite visualizar las lecturas de los sensores mencionados anteriormente.

7.7 DESCRIPCIÓN DE LOS SENSORES Y CÓDIGOS DE LECTURA

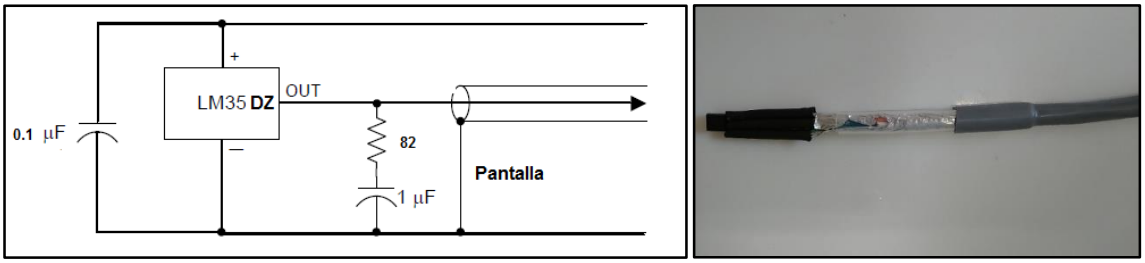
En primera instancia se dan a conocer las características de los sensores que se implementaran en el prototipo:

7.7.1 Sensor de temperatura LM35DZ

Es un circuito integrado de precisión, cuya tensión de salida es linealmente proporcional a la temperatura Celsius (centígrados) a razón de cada grado equivale a 10 mV. El LM35 no requiere ninguna calibración externa o recorte para proporcionar precisiones típicas de ¼ °C a temperatura ambiente y ¾ °C en el rango de -55 a 150 °C. El LM35 no requiere de circuitos adicionales para calibrarlo externamente. La baja impedancia de salida, su salida lineal y su precisa calibración hace posible que éste integrado sea instalado fácilmente en un circuito de control. Debido a su baja corriente de alimentación de 60 µA se produce un reducido efecto de auto calentamiento.

Con este sensor se realiza la construcción de la sonda como se muestra en la figura 22, para la cual se emplea 3 metros de cable de instrumentación 3X24 AWG. Con el objeto de evitar interferencias se conecta un condensador de 0.1 µF entre V+ y GROUND, de igual manera, una resistencia de 82 Ω en serie con un condensador de 1 µF de la salida del LM35DZ a GROUND.

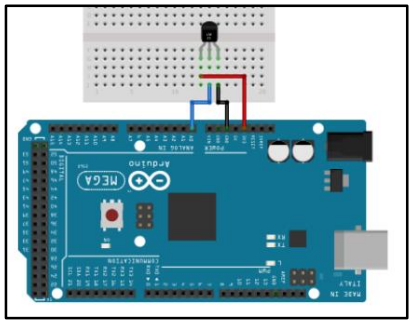
Figura 22. Sonda de temperatura



Fuente: Autores

La salida del sensor se conecta al pin A0 del Arduino Mega (ver figura 23), el mismo se encuentra seleccionado previamente y definido dentro del código fuente, con el propósito de leer la temperatura.

Figura 23. Conexión sonda a la placa Arduino Mega



Fuente: Autores

El Conversor Análogo Digital de Arduino es de 10 bits (2^{10} = 1024), por lo tanto para convertir el valor de la lectura en temperatura usamos la siguiente expresión, tempC = (1.1* lectura *100) / 1024. El factor adicional que aparece en la expresión es a causa de la especificación del sensor LM35, el que tiene una escala de 10mV/C por lo tanto para obtener la temperatura directamente en grados Celsius debemos multiplicar por 100. Adicionalmente, se especifica el voltaje de referencia análogo a 1.1 voltio con el objeto de lograr una resolución de 0.1 °C. (1.1 * 100 / 1024).

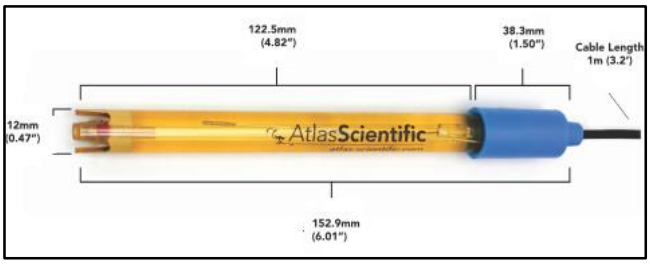
A continuación se presenta el código empleado para obtener el valor de temperatura:

```
//Definir variables temperatura

int tempC;       //Almacena la conversión de temperatura a grados Celsius
int tempSensor;  //Almacena lectura análoga del sensor de temperatura
int tempPin = 0; // Define la entrada análoga para temperatura en pin A0

void setup()
        {
         analogReference(INTERNAL1V1); //referencia de conversión análoga incorporada a 1.1
voltio
          }
void loop()
        {
         tempSensor = analogRead(tempPin); // Lee el valor análogo de temperatura desde el
       sensor  LM35
        tempC = (1.1 * tempSensor * 100.0)/1024.0;  // Convierte el valor de temperatura en
Celsius
         }
```

7.7.2 Sensor de Potencial de Hidrógeno (pH)

El sensor de Potencial de Hidrógeno (pH) es un transductor que permite conocer el pH de una solución (ver figura 24), esto lo realiza a través de un método electroquímico que utiliza una membrana de vidrio que separa dos sustancias con diferentes cantidad de protones, el sensor es un elemento pasivo que genera una pequeña cantidad de corriente de acuerdo al nivel de pH que se encuentre en el medio acuático.

Figura 24. Sensor de pH



Fuente: Atlas-Scientific, 2015

Entre las especificaciones generales en la tabla 6 se describen las siguientes:

Tabla 6. Características sensor de pH

Parámetro	Valor
Rango de pH	0-14
Temperatura de operación	1°C - 99°C
Presión máxima	690 kPa (100PSI)
Velocidad de respuesta	95% in 1 segundo
Punto isopotencial	pH 7.00 (0 mV)
Dimensiones	12mm X 150mm (1/2" X 6")
Tipo de Conector	BNC

Fuente: Atlas-Scientific, 2015

El sensor de pH de *Atlas Scientific* permite una rápida solución para diseños de bajo costo sin que se sacrifique la operatividad del diseño, consta de una sonda de pH que produce una corriente (que puede ser positiva o negativa) muy débil y no puede ser detectada con un multímetro o un convertidor analógico a digital, ésta señal eléctrica débil puede ser fácilmente interrumpida y se debe tener cuidado de sólo utilizar conectores y cables apropiados, asimismo, se cuenta con un circuito embebido llamado *EZO pH Circuit* mostrado en la figura 25, la sonda se conecta directamente a dicho circuito y este último a la etapa de adquisición de datos basada en el Arduino Mega 2560.
El circuito EZO para medición de pH de Atlas Scientific brinda una gran estabilidad, precisión y resolución. Soporta el modo de comunicación UART facilitando el diseño del prototipo al no tener que efectuar la conversión de la señal analógica entregada por la sonda a un valor digital disminuyendo la carga de procesamiento de la placa electrónica Arduino. Sus características se ven en la tabla 7.

Figura 25. Circuito embebido EZO pH

Fuente: Atlas-Scientific, 2015

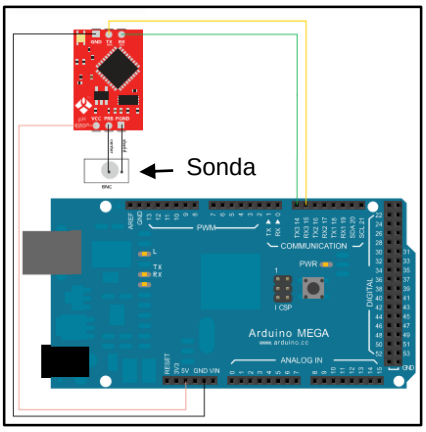
Tabla 7. Características circuito EZO pH

Parámetro	Valor
Gama de lectura	0,01 a 14
Precisión	+/- 0,02
Formato de los datos	ASCII
Tiempo entre lecturas	1 segundo

Fuente: Atlas-Scientific, 2015

Para realizar la lectura de pH en el Arduino se habilita un puerto para la recepción del valor enviado por el sensor. El puerto asignado para la conexión hacia el sensor de pH es el Serial 3, como se observa en la figura 26, que se encuentra conformado por los pines 14 (Tx) y 15 (Rx) y debidamente inicializado en el código.

Figura 26. Conexión del circuito EZO pH a la placa Arduino



Fuente: Atlas-Scientific, 2015

La habilitación del puerto Serial 3 no es suficiente ya que la etapa de adquisición de datos debe interpretar lo recibido en el puerto para poder realizar el resto de procesos programados en ella, para tal fin se codifica una función que recibe caracter a caracter el valor de pH enviado desde el circuito EZO pH y una vez obtenido el valor completo lo guarda en una variable, como se detalla a continuación:

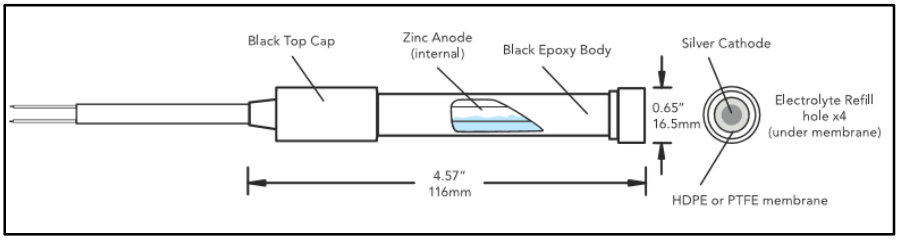
```
//Definir variables pH

String sensorstring_pH = "";  //un string para guardar el dato del producto Atlas Scientific
boolean sensor_string_pH_complete = false; //verificar si se ha recibido todos los datos del producto Atlas Scientific
int pH;   //guardar el número equivalente al pH

void setup()
          {
          void serialEvent3() { //si el hardware serial port_3 recibe un char
          sensorstring_pH = Serial3.readStringUntil(13);  //leer el string hasta ver un <CR>
           sensor_string_pH_complete = true;    //establecer la bandera utilizada para saber si se ha recibido una cadena completa
                                    }
          Serial3.begin(9600); //seleccion de baud rate para puerto 3 serial a 9600
sensorstring_pH.reserve(30); //reservar bytes para recibir datos del sensor pH de Atlas Scientific
          }
```

```
void loop()
            {
             if (sensor_string_pH_complete == true)  //si un string del sensor pH de Atlas
       Scientific ha sido recibido en su totalidad
                {
                if (isdigit(sensorstring_pH[0])) //si el primer caracter del string es un
digito
                  {
                    char buf_pH[sensorstring_pH.length()];
                    sensorstring_pH.toCharArray(buf_pH, sensorstring_pH.length());
                    pH = 100*(atof(buf_pH));   //convertir el string a un numero de punto
flotante
                  }
              sensorstring_pH = "";   //limpiar el string
            sensor_string_pH_complete = false;     //resetear la bandera que se llamó cuando
se recibió el string completo del sensor de pH
             }
```

7.7.3 Sensor de oxígeno disuelto (OD)

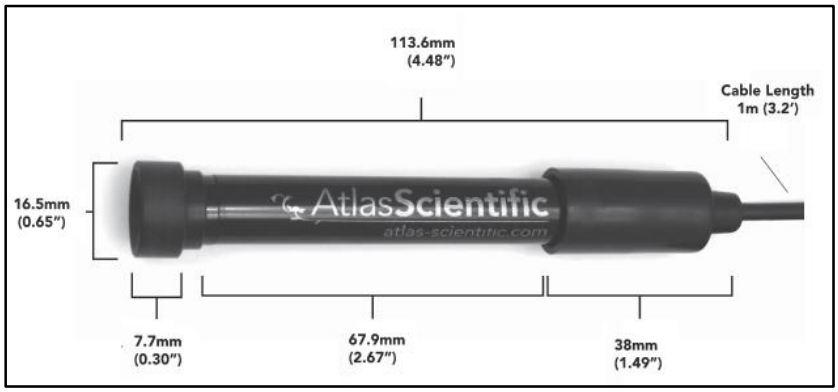
El sensor de Oxígeno disuelto mostrado en la figura 27 mide la saturación de Oxigeno dentro de un cuerpo de agua con el cual se puede determinar la calidad de la misma, el sensor es un elemento pasivo que genera una pequeña cantidad de voltaje desde 0 mV a 47mv dependiendo de la saturación de oxígeno de la membrana de detección de HDPE (High Density Polyethylene). Esta tensión puede ser fácilmente leída por un multímetro o un convertidor de analógico a digital. El sensor de OD de *Atlas Scientific* que se observa en la figura 28 permite una rápida solución para diseños de bajo costo sin que se sacrifique la operatividad del diseño al no tener que efectuar la conversión de la señal analógica entregada por la sonda a un valor digital y aumentar la carga de procesamiento de la placa electrónica Arduino cuyas características se especifican en la tabla 8. La sonda consta de una membrana HDPE y un circuito embebido llamado *EZO OD Circuit*, la sonda se conecta directamente a dicho circuito y este último a la etapa de adquisición de datos basada en el Arduino Mega 2560.La sonda está construida de un tubo con una varilla de zinc (ánodo) sumergido en un electrolito. El elemento de detección es la membrana de HDPE comprimida contra un disco de plata (cátodo).

Figura 27. Estructura interna y externa de la sonda de Oxígeno disuelto



Fuente: Atlas-Scientific, 2015

Figura 28. Sensor de OD



Fuente: Atlas-Scientific, 2015

Tabla 8. Características sensor de OD.

Parámetro	Valor
Rango de OD	0-20 mg/L
Temperatura de operación	50 °C
Presión máxima	690 kPa (100PSI)
Material del cuerpo	Resina epóxica y noryl
Dimensiones	16.5mm X 116mm (0.65" X 4.17")
Tipo de Conector	BNC

Fuente: Atlas-Scientific, 2015

El circuito EZO para medición de OD de Atlas Scientific, ver figura 29, brinda una gran estabilidad, precisión y resolución. Soporta el modo de comunicación UART, las lecturas son presentadas directamente en miligramos por litro (mg/L) y tiene la capacidad de compensar variaciones de temperatura y salinidad. Sus características se desglosan en la tabla 9.

Figura 29. Circuito embebido EZO OD

Fuente: Atlas-Scientific, 2015

Tabla 9.Características circuito EZO OD

Parámetro	Valor
Gama de lectura	0,01 a 35.99 mg/L
Precisión	+/- 0,2
Formato de los datos	ASCII
Tiempo entre lecturas	1 segundo

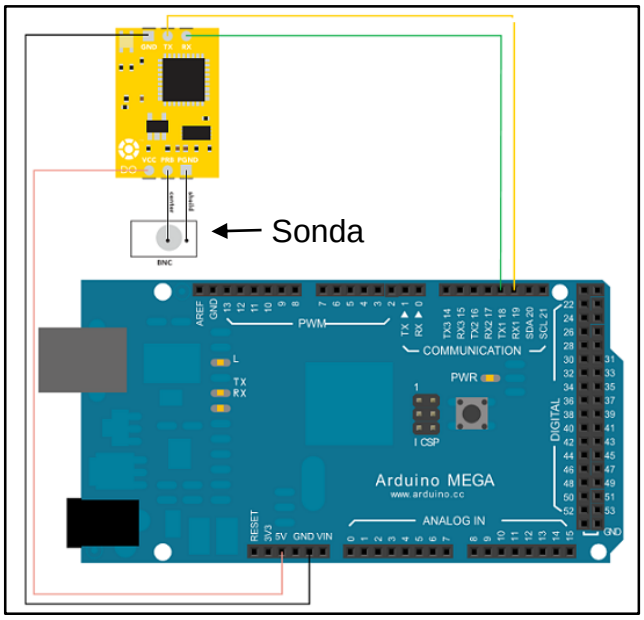
Fuente: Atlas-Scientific, 2015

La conexión entre el circuito EZO DO y el Arduino se realiza al puerto Serial 1, como se ve en la figura 30, que se encuentra formado por los pines 18 (Tx) y 19 (Rx) y se inicializa en el código del Arduino, así mismo, se crea una función para la lectura del valor de Oxígeno Disuelto desde el sensor, que funciona de manera similar a la indicada para el sensor de pH. El código implementado es el que sigue:

```
//Definir variables OD

String sensorstring_OD = "";  //un string para guardar el dato del producto Atlas Scientific
boolean sensor_string_OD_complete = false; //verificar si se ha recibido todos los datos del producto Atlas Scientific
int OD;   //guardar el número equivalente al OD
```

```
void setup()
          {
          void serialEvent1() { //si el hardware serial port_1 recibe un char
          sensorstring_OD = Serial1.readStringUntil(13);  //leer el string hasta ver un <CR>
          sensor_string_OD_complete = true;    //establecer la bandera utilizada para saber
si se ha recibido una cadena completa
                                }

          Serial1.begin(9600); //seleccion de baud rate para puerto 1 serial a 9600
          sensorstring_OD.reserve(30); //reservar bytes para recibir datos del sensor OD de
Atlas Scientific
          }

void loop()
           {
            if (sensor_string_OD_complete == true)   //si un string del sensor pH de Atlas
       Scientific ha sido recibido en su totalidad
               {
               if (isdigit(sensorstring_OD[0])) //si el primer caracter del string es un
digito
                 {
                   char buf_OD[sensorstring_OD.length()];
                   sensorstring_OD.toCharArray(buf_OD, sensorstring_OD.length());
                   OD = 100*(atof(buf_OD));   //convertir el string a un numero de punto
flotante
                 }
             sensorstring_OD = "";   //limpiar el string
          sensor_string_OD_complete = false;     //resetear la bandera que se llamó cuando
se recibió el string completo del sensor de OD
            }
```

Figura 30. Conexión del circuito EZO OD a la placa Arduino Mega

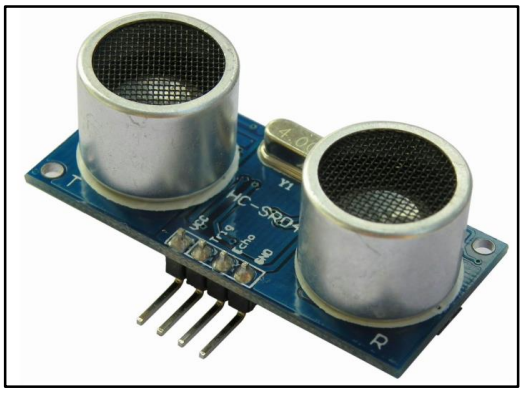


Fuente: Atlas-Scientific, 2015

7.7.4 Sensor de distancia para medir nivel de estanque

Para asegurar alta confiabilidad al momento de realizar el monitoreo del nivel del estanque se optó por utilizar un sensor ultrasónico, se decidió por este tipo de dispositivos ya que el agua para la incubación de los peces no puede entrar en contacto con elementos metálicos que pueden afectar a los alevinos. Los sensores de ultrasonido son los más económicos, fáciles de manipular y pueden detectar objetos localizados a distancias en el orden de los metros sin necesidad de algún filtro o adaptación especial. Después de investigar en los mercados diferentes tipos de sensores existentes, se optó por elegir el sensor de ultrasonido HC-SR04 mostrado en la figura 31.

Figura 31. Sensor de ultrasonidos HC-SR04



Fuente: Cytron Technologies, 2013

Para realizar la conexión del sensor de distancia, desde el punto donde se llevará a cabo la medición al Arduino Mega, se emplea 3 metros de cable de instrumentación de 4X24AWG, como se aprecia en la figura 32 y sus principales características se describen en la tabla 10.

Figura 32. Sensor con cable de conexión

Fuente: Autores

Tabla 10. Características sensor de ultrasonido

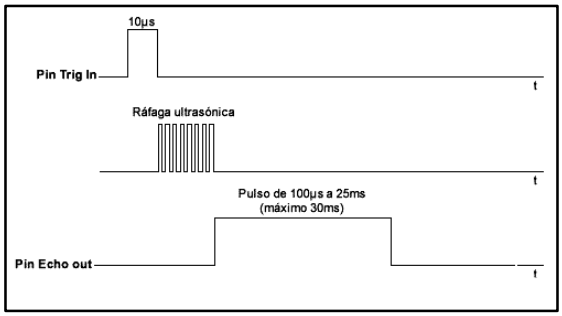
Parámetro	Valor
Voltaje de funcionamiento	5 vdc
Corriente inactivo	< 2 mA
Corriente de trabajo	15 mA
Angulo efectivo	< 15 °
Distancia de medición	2 cm – 400 cm
Resolución	0.3 cm
Angulo de medición	30 °

Fuente: Cytron Technologies, 2013

El funcionamiento de este módulo es muy sencillo. Esta alimentado con 5 V y se debe suministrar un pulso de 10 µs para activar el modulo a través del pin Trig el cual está conectado al pin 7 del Arduino Mega. En ese momento, el modulo lanzará una ráfaga de 8 pulsos ultrasónicos a 40Khz y la salida Echo pasa a nivel alto hasta que el modulo recibe un eco, momento en el que volverá de nuevo a pasar a un nivel bajo. Por tanto, la salida Echo es un pulso cuyo ancho será proporcional a la distancia respecto a un objeto. Si no se detecta un objeto, la

salida Echo pasara a nivel bajo después de 30 ms. Esta señal se recibe en el pin 6 del Arduino Mega con la cual se realiza el cálculo de la distancia, lo anterior se representa en la figura 33.

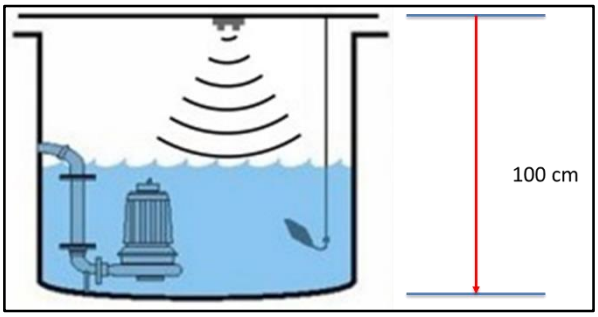
Figura 33. Señales en el sensor HC-SR04



Fuente: Cytron Technologies, 2013

Si el ancho del pulso se mide en µs, el resultado se debe dividir entre 58 para saber la distancia en centímetros. Estos valores son obtenidos de:
Si la velocidad del sonido es 340 metros por segundo o 29 µs por centímetro, y como el sonido tiene que viajar dos veces la distancia hacia el objeto, una de ida y otra de vuelta, entonces cada 2 × 29 = 58 µs recorrerá un centímetro. Para determinar el nivel de agua se midió la altura del estanque desde el fondo a la ubicación del sensor y éste valor es una constante, por lo tanto, el nivel resulta de restar a dicha constante la distancia al agua. Ver figura 34.

Figura 34. Medición del nivel del agua



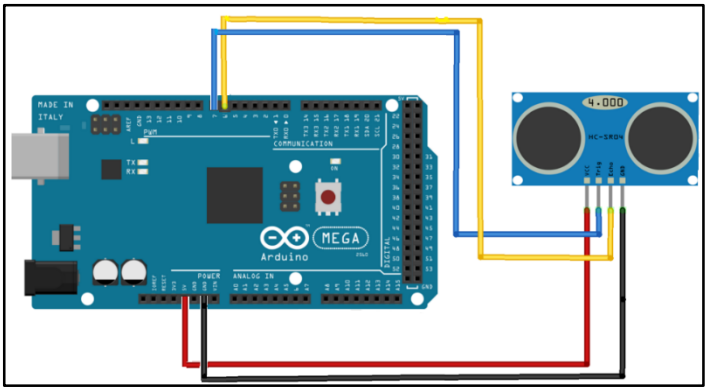
Fuente: Autores

El modulo debe activarse cada 50 ms como mínimo, de esta manera se asegura que la ráfaga ultrasónica haya desaparecido completamente y no provocará un falso eco en la siguiente medición de distancia. En seguida se detalla el código desarrollado para efectuar la lectura de la distancia que se identificara en el prototipo como un valor de nivel:

```
//Definir variables distancia para obtener nivel del estanque

       /* HC-SR04 conexiones:
          VCC al Arduino 5v
          GND al Arduino GND
          Echo al Arduino pin 6
          Trig al Arduino pin 7
       */

#define Pecho 6   //Pin de eco
#define Ptrig 7   //Pin de disparo

int duracion, distancia;   //Almacenan la duracion del recorrido del ultrasonido y el
resultado de la medición en cm

//Variables establecer retardo para tomar lectura de distancia

unsigned long wdog = 0;      /* watchdog */
unsigned long tprev = 400;   /* tiempo previo*/
unsigned long tactual = 0;   /* tiempo actual*/

void setup()
          {
           pinMode(Pecho, INPUT);     // define el pin 6 como entrada (echo)
           pinMode(Ptrig, OUTPUT);    // define el pin 7 como salida  (triger)
            }

void loop()
           {
            wdog = millis();

            if (wdog > tactual + tprev)
            {

            tactual = wdog;

            digitalWrite(Ptrig, LOW);
            delayMicroseconds(2);
            digitalWrite(Ptrig, HIGH);   // genera el pulso de disparo por 10ms
            delayMicroseconds(10);
            digitalWrite(Ptrig, LOW);

            duracion = pulseIn(Pecho, HIGH);

            distancia = (duracion/2) / 29;  // calcula la distancia en centimetros
               }
            }
```

La forma como se conecta el sensor al Arduino Mega se muestra en la figura 35.

Figura 35. Conexión de HC-SR04 a la placa Arduino Mega



Fuente: Autores

7.8 CREACIÓN DE LA ESTRUCTURA SCADA

Para esta estructura se implementará una interfaz usando el programa Acimut Monitoriza que se comunica con la placa Arduino Mega, que se comportará como esclavo, mediante el protocolo de comunicación ModBus en modo RTU. En seguida se establecen cada una de las etapas a seguir.

a. Como primer paso se programa por medio del IDE de Arduino el algoritmo de control. En el código se declaran los parámetros de comunicación del esclavo y los 6 registros que corresponden a las 6 variables del proceso que van a ser solicitados por el maestro ModBus y se enumeran así: 4 variables que corresponden con los datos entregados por los sensores y 2 variables para la indicación de encendido y señal de alarma. El código realizado se muestra más abajo:

```
void configure_mb_slave(long baud, char parity, char txenpin); //parametros de la comunicacion serial

/* baud:       baud rate en bps (valores tipicos 9600, 19200... 115200)
 * parity:     un solo carácter establece el modo de paridad (character frame format):
 *             'n' no parity (8N1); 'e' even parity (8E1), 'o' for odd parity (8O1).
 * tx_en_pin:  numero de pin de arduino que controla transmision/recepcion
 *             de un dispositivo externo half-duplex (ejm. un RS485 interface chip).
 *             0 o 1 desactiva esta función (para una red de dos dispositivos)
 *             >2 para topologia punto-a-multipunto (ejm. varios arduinos)
 */
```

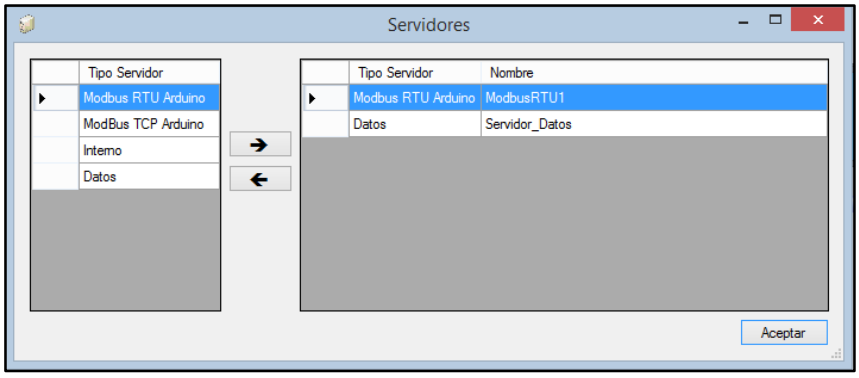
```
int update_mb_slave(unsigned char slave, int *regs,
unsigned int regs_size);

* slave:  id esclavo (1 a 127)
 * regs:  una matriz con los holding registers. Comienzan en la direccion 1 (punto de vista
del maestro)
 * regs_size: numero total de holding registers.


/* Parámetros comunes Modbus RTU, el Master DEBE usar los mismos parámetros */

enum {
        COMM_BPS = 9600, /* baud rate */
        MB_SLAVE = 1, /* modbus slave id */
        PARITY = 'n' /* even parity  (8N1)*/
     };

/* Registros del esclavo */

enum {
        MB_TEMP,     /* Temperatura */
        MB_DIST,     /* Nivel estanque */
        MB_PH,       /* Nivel de pH */
        MB_OD,       /* Nivel de Oxigeno Disuelto */
        MB_IND,      /* Indicador de encendido */
        MB_ALARM,    /* Indicador de alarma */
        MB_REGS      /* numero total de  registros en esclavo */

     };


/* Variables a utilizar */

int regs[MB_REGS];          //Registros para ModBus
int IncubaPowerOn = 31;    //Indicador de encendido
int IncubaPowerOff = 33;  //Indicador de apagado
int Pulsador_OFF = 2;     //Pulsador de encendido
int Pulsador_ON = 3;   //Pulsador de apagado
int Alarma = 35;       //Indicador de alarma


void setup()
          {
           configure_mb_slave(COMM_BPS, PARITY, 0);  /* configurar comunicacion modbus 9600
bps, 8N1, para una red de dos dispositivos*/
          }


void loop()
            {
             if(update_mb_slave(MB_SLAVE, regs, MB_REGS))    /*comprobar si hay solicitud del
maestro*/
             }

/* REGISTROS PARA LAS VARIABLES */

regs[MB_TEMP] = tempSensor;  //Registro 0 Modbus de temperatura para SCADA
regs[MB_DIST] =  distancia;  //Registro 1 Modbus de distancia para SCADA
regs[MB_PH] =  pH;           //Registro 2 Modbus de pH para SCADA
regs[MB_OD] =  OD;           //Registro 3 Modbus de OD para SCADA


/* VISUALIZACION ENCENDIDO DEL EQUIPO */

if (regs[MB_IND] == 0)              //Registro 4 Modbus de Indicador de encendido para SCADA

   {digitalWrite(IncubaPowerOn, HIGH);
```

```
    digitalWrite(IncubaPowerOff, LOW);}

if (regs[MB_IND] == 1)
   {digitalWrite(IncubaPowerOn, LOW);
    digitalWrite(IncubaPowerOff, HIGH);}


/* VISUALIZACION ALARMA*/

if (tempC < 280 || tempC > 320 || distancia < 50 || distancia > 90 || pH < 300 || pH > 900
|| OD < 500 || OD > 900)

{regs[MB_ALARM] = 1;}       //Registro 5 Modbus de Indicador de alarma para SCADA

else {regs[MB_ALARM] = 0;}
```

b. Realizado el programa para el Arduino Mega, se procede a crear la interfaz del proceso usando el editor de Monitoriza que es uno de los tres componentes principales del sistema, con él se va a crear, diseñar y modificar el proyecto de SCADA que luego se ejecutará a través del Servidor de Comunicaciones y del Cliente SCADA.

Como primera medida se debe definir un servidor de base de datos en el que se almacena permanentemente los valores de las variables que estamos monitorizando y otro de comunicaciones para el nuevo proyecto con la placa Arduino Mega, en este caso ***Modbus RTU Arduino,*** al cual se le establecen los mismos parámetros de comunicación señalados para el esclavo, como se evidencia en la figura 36 y 37. Para comprobar el puerto de comunicaciones serial COM asignado a la conexión USB del Arduino Mega se hace mediante el Administrador de equipos de Windows, en el proyecto SCADA, se establece en la propiedad **Port,** el número de puerto que se tiene asignado para dicha conexión, en este caso el 5.

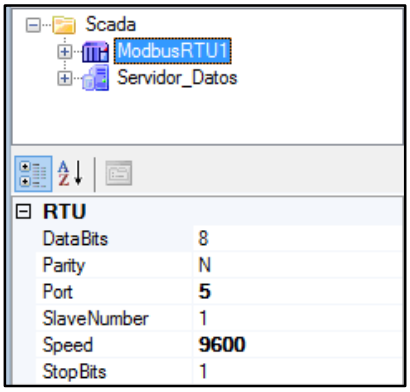
Figura 36. Definición de los servidores



Fuente: Autores

Figura 37. Parámetros de comunicación con el esclavo Modbus

```
COMM_BPS = 9600, /* baud rate */
MB_SLAVE = 1,       /* modbus slave id */
PARITY = 'n' /* even parity  (8N1)*/
```



Fuente: Autores

De igual forma se deben establecer las variables definidas anteriormente a las que se va a acceder, que estarán unidas en un grupo al que se dará nombre de ***ArduinoMega***, para ello en la ventana de servidores se pulsa sobre los tres puntos de la propiedad **Grupos**. Ver figura 38.

Las propiedades que se definen en el grupo del servidor ModBUS RTU son las siguientes:

- **GroupType:** Tipo de Grupo, indica si las variables son de tipo registro o de tipo marca. Se selecciona **ReadHoldingRegisters** ya que las que se van a leer/escribir son de tipo registro (una palabra, 2 bytes), utilizando por ello la función 3 del protocolo Modbus.

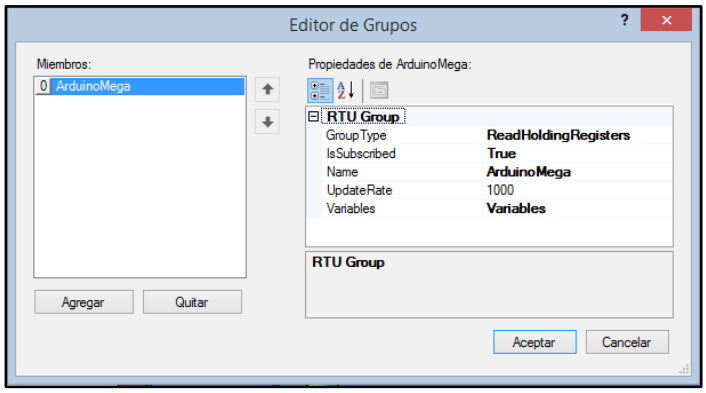
- **IsSubscribed:** Se deja en TRUE para indicar que este grupo de variables se leerá continuamente.

- **Name:** Denomina al grupo, para el proyecto a crear se le dará el nombre de ***ArduinoMega.***

- **UpdateRate:** Como IsSuscribed está a True esta propiedad indicará cada cuántos milisegundos se leerá el grupo. Se selecciona un valor de 1000 para obtener una lectura cada segundo.

- **Variables:** En esta sección se discriminan las variables a utilizar descritas anteriormente. Al dar clic aquí se establecen las propiedades de las mismas.

Figura 38. Configuración del Editor de grupos del proyecto

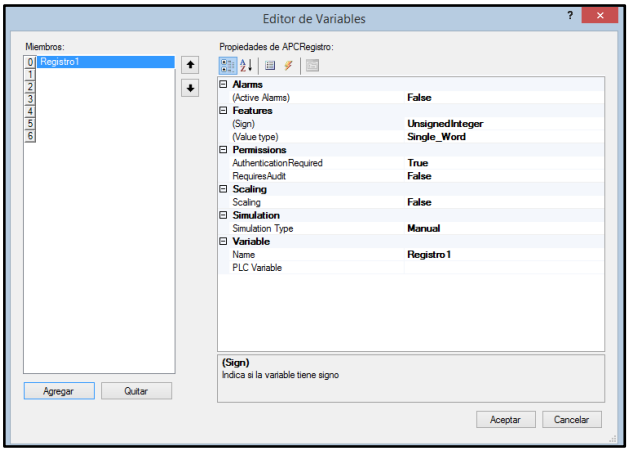


Fuente: Autores

Una vez especificado el grupo se procede a definir las variables o registros. Para ello en la misma ventana se tiene la propiedad **Variables,** como se muestra en la figura 39, donde se debe pulsar el botón de tres puntos para que aparezca la ventana de definición de variables o registros.

Antes de realizar este paso se especifica a continuación los ítems que aparecen en dicha ventana para tener una idea general de las opciones que se pueden adjudicar a cada una de las variables.

Figura 39. Propiedades del editor de variables



Fuente: Autores

- **Active Alarms:** Indica si deben activarse (TRUE) o no (FALSE) las alarmas para esa variable.
  Cuando se escoge TRUE aparece la propiedad **Alarms** en la que al pulsar en el botón de tres puntos se puede definir las alarmas asociadas a la variable.

- **Value type:** Define el tamaño de la variable. Las opciones posibles son: Single_Word (2 bytes) y Double_Word (4 bytes).

- **Sign:** Define si las variables se consideran con signo o no. Las opciones posibles son: UnsignedInteger – Entero sin signo y SignedInteger – Entero con signo.

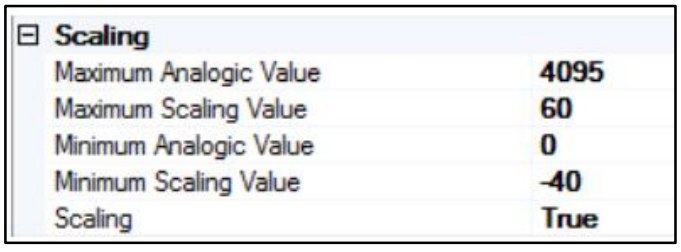
- **Name:** Nombre simbólico que se le da a la variable. Debe estar formado por caracteres alfanuméricos y no contener ni espacios ni símbolos.

- **PLC Variable:** Identifica el registro de la placa Arduino Mega al que se quiere hacer referencia.

- **Permissions:** Las propiedades AuthenticationRequired y RequiresAudit establecen si se deberá autenticar un usuario para poder cambiar un valor y si se guardará un registro de los cambios efectuados respectivamente.

- **Scaling:** Indica si el valor de este registro se escalará, es decir se puede indicar los valores Máximo y Mínimo para los valores Analógico y Escalado, si por ejemplo se cuenta con un registro que va a dar el valor de entrada de una sonda, y se tiene que la señal analógica es de 12 bits (4096) y mide temperaturas entre -40 ºC y 60 ºC se configura estas propiedades como en la figura 40:

Figura 40. Ejemplo de escalado



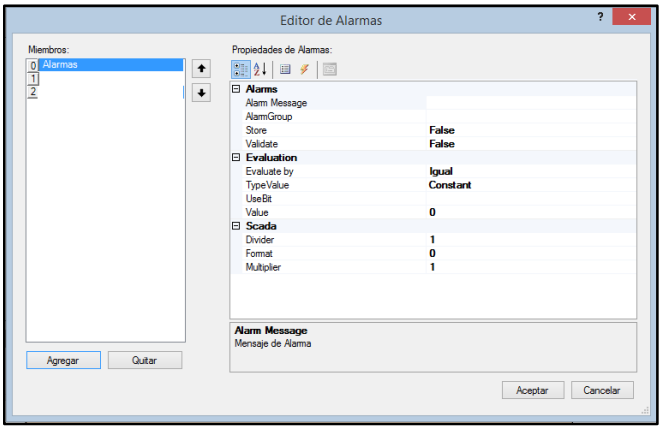
Fuente: Autores

Hay que tener en cuenta que el valor escalado es el que se utilizará en las alarmas y base de datos. Sin embargo los valores en modo simulación no se escalan. Se utiliza un escalado lineal.

- **Simulation Type:** Cuando en diseño se “ejecuta” un proyecto, esta opción permite cambiar de forma manual o automática el valor de las variables.

En cuanto a la propiedad **Alarms** como se ve en la figura 41, es posible crear una o varias alarmas en función de las condiciones que se quieran controlar. Al definir la alarma se establece el mensaje que se desea mostrar, si se quiere almacenar el evento para tener constancia de que ha ocurrido o si es necesario que un usuario valide la alarma, así como las condiciones en las que se debe disparar la alarma.

Figura 41. Propiedades del editor de alarmas



Fuente: Autores

A continuación se especifica cada uno de los ítems que hacen parte del editor de alarmas:

- **Store:** Permite indicar si queda constancia de la alarma. Para poder almacenar las alarmas se tendrá que definir adicionalmente un servidor de base de datos en el que, en su propiedad **Alarms**, se establezca que es el que se utilizará para guardar éstas.

- **Alarm Message:** Mensaje que mostrará la alarma al activarse.
- **AlarmGroup:** Grupo de permisos a la que está vinculada la alarma.

- **Validate:** Si se establece a True, exigirá que un usuario del proyecto SCADA la valide para que deje de mostrarse en la ventana de alarmas cuando ya no está activa. Para que la validación actúe correctamente es necesario almacenar la alarma.

- **Evaluate by:** Establece la condición de comparación que determina cuando estará activa la alarma. Los posibles valores son: *Igual, Mayor Que, Menor Que y Distinto*.

- **TypeValue:** Los posibles valores son *Constant* y *Variable*.

- **Value:** Establece el valor que al aplicarle la condición de **Evaluate by** hará que se active la alarma.

El grupo de propiedades SCADA permite especificar el valor de la variable para su almacenamiento en la base de datos y por tanto para su visualización en el Visor de Alarmas.

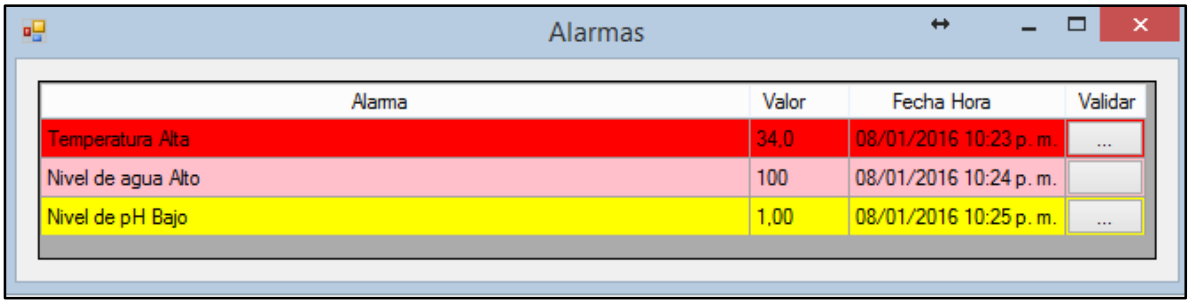
- **Divider:** Mostrará el valor de la variable dividido por el número indicado en la propiedad.

- **Multiplier:** Mostrará el valor de la variable multiplicado por el por el número indicado en la propiedad.

- **Format:** Modifica el aspecto del valor de la variable, por ejemplo un formato igual a ##.00 hará que el la variable se muestre con dos decimales.

El visor de alarmas de la figura 42 es el elemento central de notificación y validación de las alarmas que se producen en la ejecución del proyecto SCADA.

Figura 42. Visor de alarmas



Fuente: Autores

Cada vez que se active una alarma, porque se cumplen las condiciones que se han establecido al diseñar el SCADA, se muestra el visor de alarmas, en él se tienen diferentes posibles estados de la alarma en función de si ha definido como que necesita validación o no y estos estados están representados por diferentes colores.

- **Rojo:** Indica que la alarma está activa y requiere que se valide, o sea, hay que pulsar sobre el botón *Validar* para hacer constar que un usuario ha tenido en cuenta la alarma y ha tomado las medidas oportunas.

- **Rosa:** Indica que la alarma está activa y se ha validado, o sea, un usuario de SCADA ha pulsado sobre el botón *Validar*.

- **Amarillo:** Indica que la alarma ya no está activa pero todavía no se ha validado.

- **Azul:** Indica que la alarma está activa y no requiere validación. En este caso al no requerir validación desaparece en el momento que la alarma deja de estar activa.

Antes de comenzar a establecer cada una de las propiedades mencionadas anteriormente a cada una de las variables que hace parte del proyecto, se señala que las siguientes propiedades van a tener la misma configuración para todas ellas:

Active Alarms: TRUE. (Salvo indicadores de encendido y alarma).

Value type: Single_Word (2 bytes).

Sign: UnsignedInteger.

Permissions: AuthenticationRequired a TRUE y RequiresAudit a FALSE.

Scaling: Se aplica solo para la variable de temperatura. (TRUE).

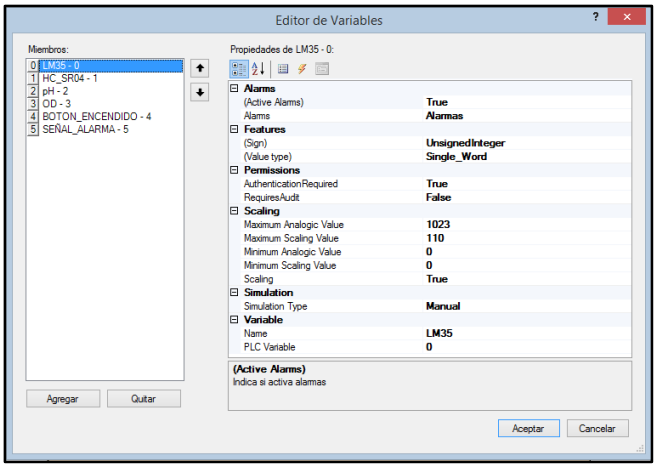
Simulation Type: Manual.

7.8.1 Propiedades asignadas a la variable temperatura

Partiendo de que el valor correspondiente a la temperatura resulta de una conversión análoga a digital realizada por la placa de Arduino Mega, es posible realizar el escalado de la misma, ver figura 43, teniendo en cuenta que el ADC es de 10 bits (1024) y que el rango de temperatura está comprendido de 0 a 110 °C (la referencia de la conversión es 1.1 voltio). El nombre que se le proporciona a la

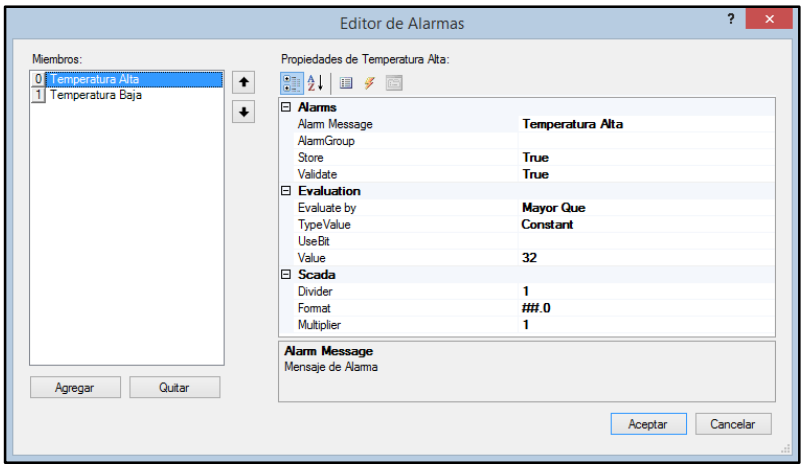
variable está relacionado con el sensor utilizado para tal fin, **LM35**, y que según el código desarrollado para el Arduino Mega corresponde con el registro 0.En lo relacionado con las alarmas, se establecen dos, la primera para mostrar la condición de alta temperatura cuando el valor sobrepase los 32 °C y la segunda, para baja temperatura cuando sea inferior a 28 °C, con la presentación de sus respectivos mensajes, la validación y el almacenamiento en la base de datos, como se observa en las figuras 44 y 45. El formato de visualización para ambos casos está determinado a un decimal (##.0).

Figura 43. Propiedades de la variable temperatura



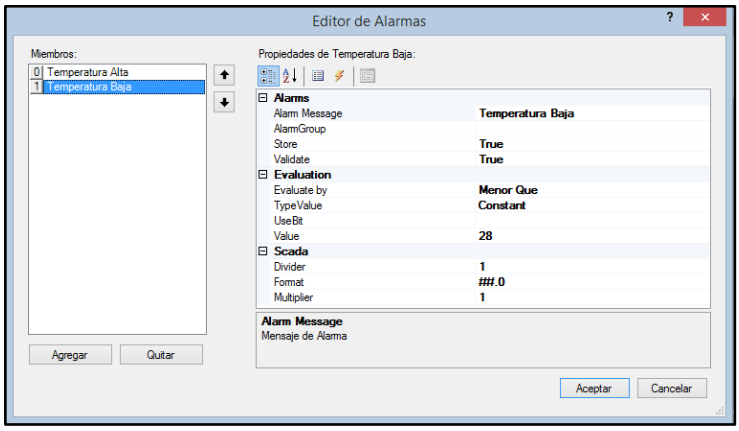
Fuente: Autores

Figura 44. Alarma para alta temperatura



Fuente: Autores

Figura 45. Alarma para baja temperatura

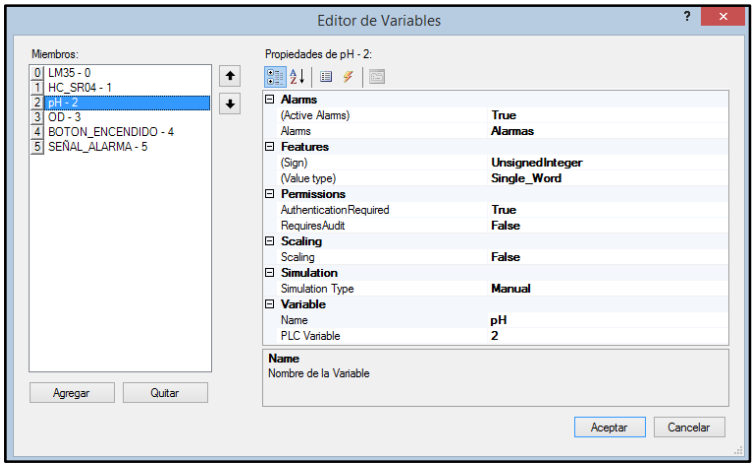


Fuente: Autores

7.8.2 Propiedades asignadas a la variable pH

El nombre que identifica a la variable dentro del SCADA es el de **pH** correspondiente al registro 2, tal como se indica en la figura 46. Es importante resaltar que el valor de la variable es presentado como un entero ya que los registros son de este tipo, sin embargo, el valor real corresponde a una cantidad con formato de dos decimales (##.00), es por ello, que debe ser dividido entre 100 para su presentación.

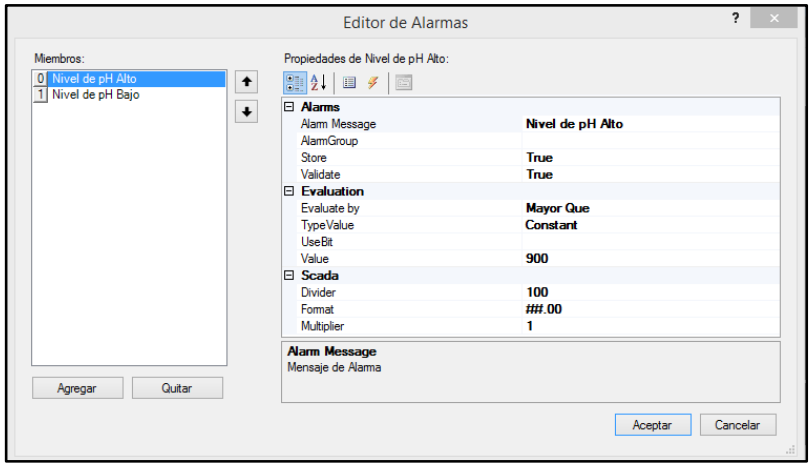
Figura 46. Propiedades de la variable pH



Fuente: Autores

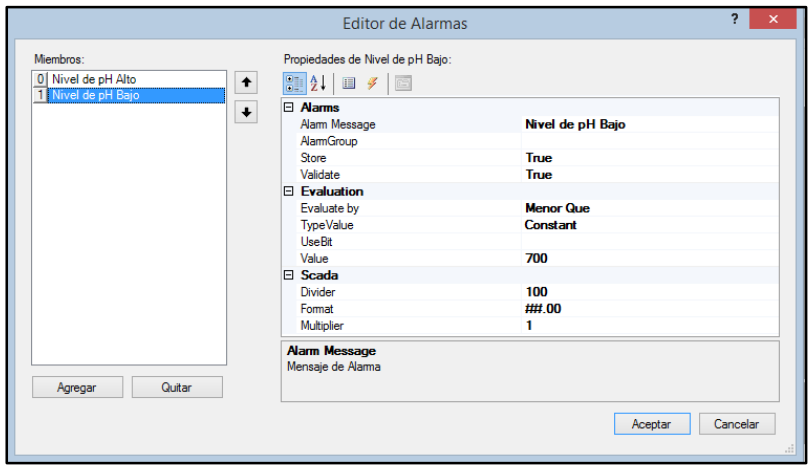
Las alarmas designadas son la de alto y bajo nivel de pH, cuyos valores son establecidos a 900 y 700 respectivamente, ver figuras 47 y 48, teniendo en cuenta lo mencionado anteriormente con respecto al tipo de variable.

Figura 47. Alarma para alto nivel de pH



Fuente: Autores

Figura 48. Alarma para bajo nivel de pH

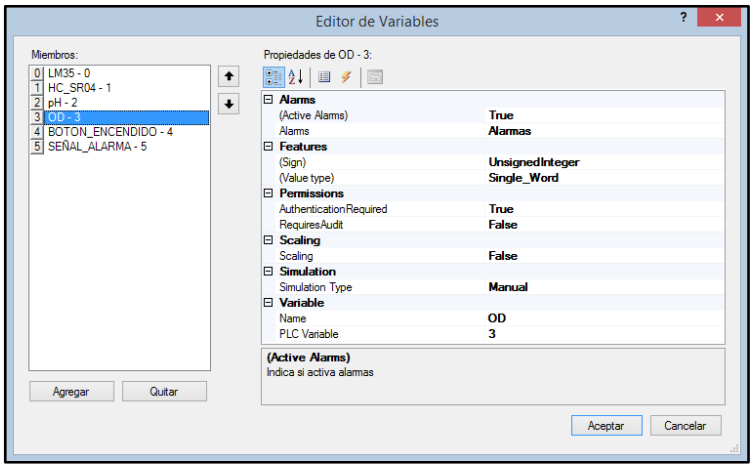


Fuente: Autores

7.8.3 Propiedades asignadas a la variable de oxígeno disuelto

Se utiliza **OD** para relacionar la variable de oxígeno disuelto dentro del proyecto y corresponde al registro 3, lo cual se señala en la figura 49. Esta variable presenta las mismas condiciones de la variable **pH** en cuanto a su manejo en el SCADA, en lo referente al tipo de dato.

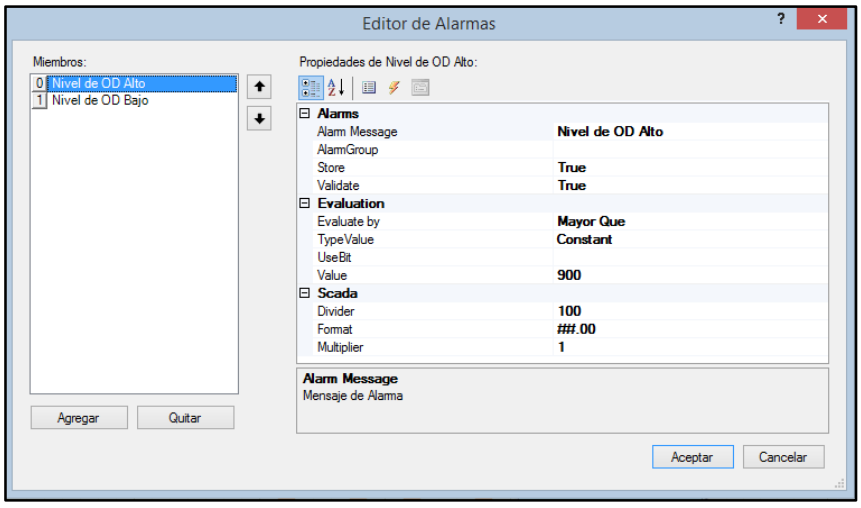
Figura 49. Propiedades de la variable oxígeno disuelto



Fuente: Autores

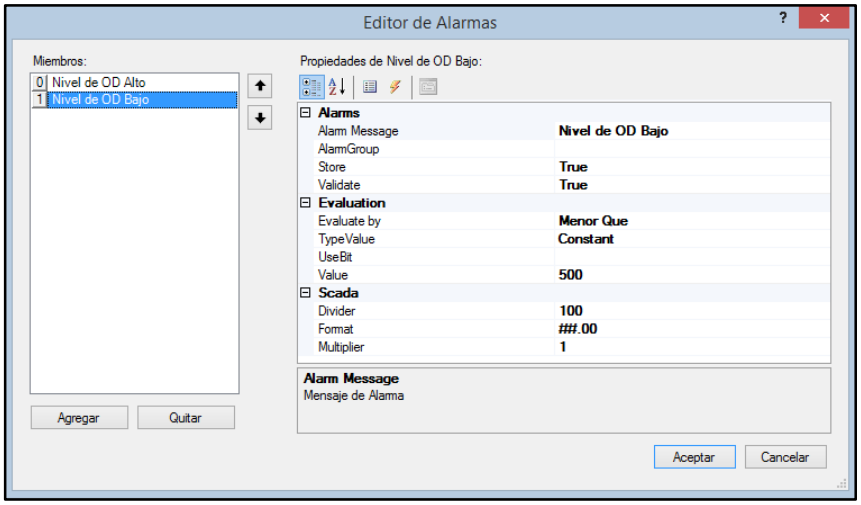
Se establecen alarmas de valor límite de 900 para indicar alto nivel de OD y de 500 para bajo nivel de OD, remitirse a las figuras 50 y 51, teniendo en cuenta el manejo del tipo de dato descrito anteriormente.

Figura 50. Alarma para alto nivel de OD



Fuente: Autores

Figura 51. Alarma para bajo nivel de OD

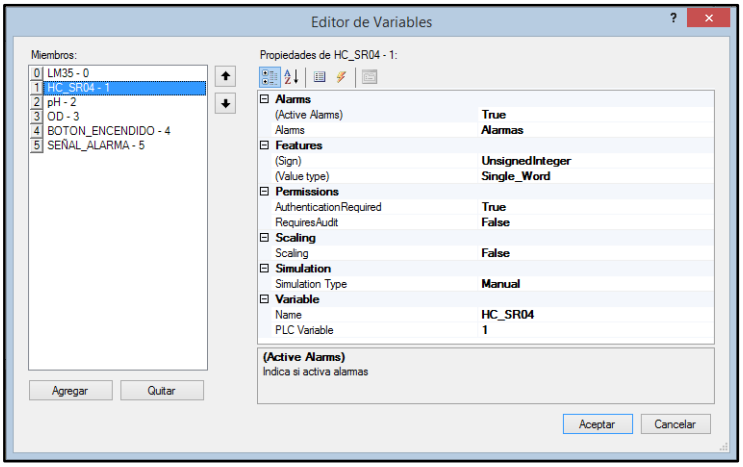


Fuente: Autores

7.8.4 Propiedades asignadas a la variable nivel del estanque

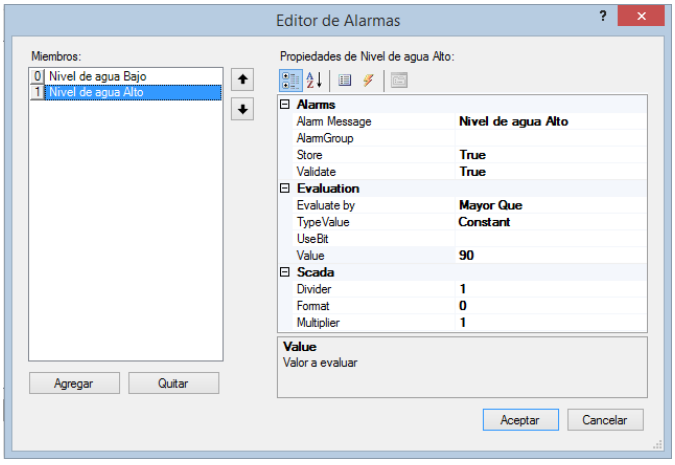
La variable se denomina **HC_SR04** nombre heredado del sensor que se emplea para llevar a cabo la medición y cuyo registro en el código del Arduino Mega corresponde a registro 1, como se aprecia en la figura 52. Su unidad de medida es el centímetro. Su valor mínimo es de 50 y el máximo de 90 mostrados en las figuras 53 y 54.

Figura 52. Propiedades de la variable nivel del estanque



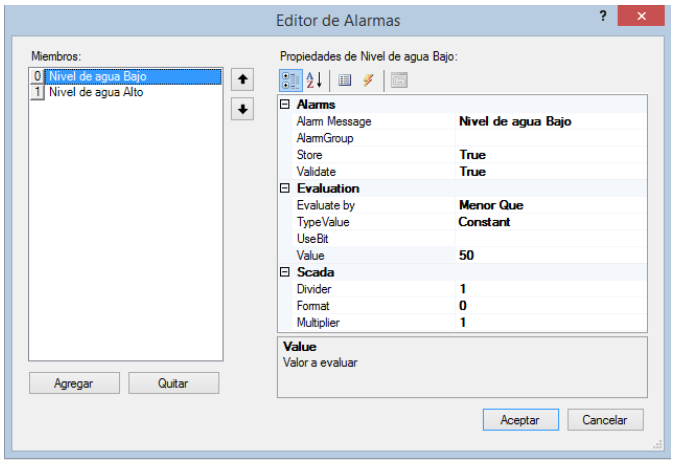
Fuente: Autores

Figura 53. Alarma para alto nivel del estanque



Fuente: Autores

Figura 54. Alarma para bajo nivel del estanque

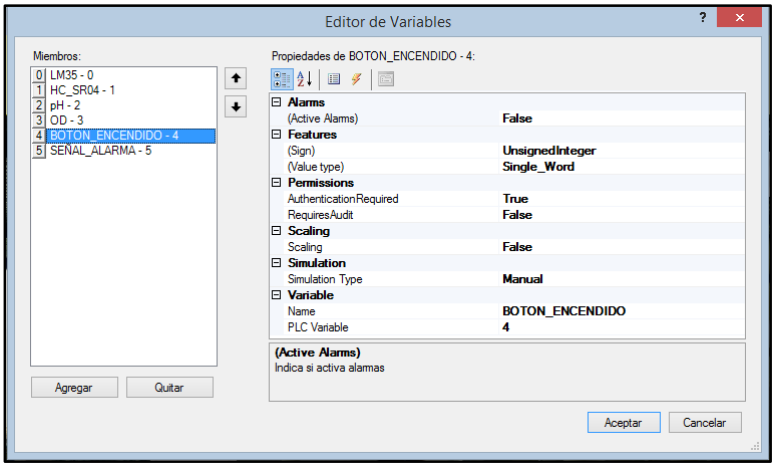


Fuente: Autores

7.8.5 Propiedades asignadas a la variable indicador de encendido

Esta variable va a ser representada por dos botones en el SCADA que tienen por objeto forzar un valor de 1 ó 0 (apagado – encendido) para accionar dos relevos estado sólido de referencia G4 OAC5 fabricados por OPTO 22 (lógica negativa), que permite activar el sistema que proveerá energía a los actuadores tales como bomba y aireador, y las luces de señalización de dicho estado. La variable se denomina BOTON_ENCENDIDO y corresponde con la asignada en el código del Arduino como registro 4. Ver figura 55.

Figura 55. Propiedades de la variable indicador de encendido



Fuente: Autores

7.8.6 Propiedades asignadas a la variable indicadora de alarma

Al igual que la anterior se emplea un relé de estado sólido de la misma referencia, mostrado en la figura 57, pero éste se activara por la condición de alarma que puedan presentar las lecturas de las condiciones del sistema (Temperatura, pH, etc.) descrito en el código, lo cual forzará un valor de 0 ó 1 en la variable designada como SEÑAL_ALARMA correspondiente al registro 5. Ver figura 56.

Figura 56. Propiedades de la variable indicadora de alarma

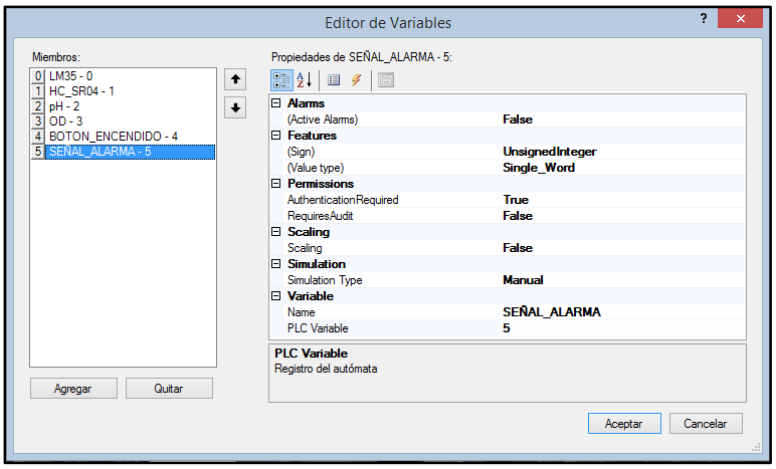
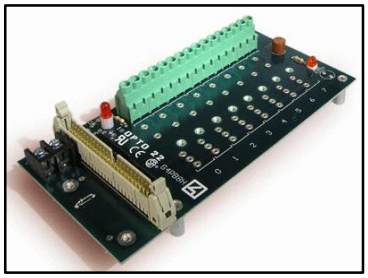


Figura 57. Relé de estado sólido empleado para actuadores y alarma

Fuente: www.opto22.com

Los relevos se instalan sobre una base mostrada en la figura 58, proporcionada por el fabricante de referencia **G4PB8H** que brinda facilidad para llevar a cabo tanto las conexiones de la señal de control como las de potencia.

Figura 58. Base de 8 canales para montaje de los relevos



Fuente: www.opto22.com

7.8.7 Creación de los formularios del SCADA

Para crear la interfaz de usuario se diseñan siete formularios para el proyecto y para ello se le agregan controles disponibles en la ventana ToolBox del software Acimut Monitoriza para Arduino de cualquiera de sus la solapas: SCADA, Diseño y Windows.

El primer formulario, ver figura 59, se denomina *"Incubadora Artificial de Alevinos"*, en él se hace una presentación general de la disposición de la incubadora mostrando los valores de las variables por medio de indicadores analógicos (temperatura, pH y OD) y barra de leds (nivel de agua) ubicados dentro de la gráfica en los puntos donde se realizan las mediciones en el montaje real.

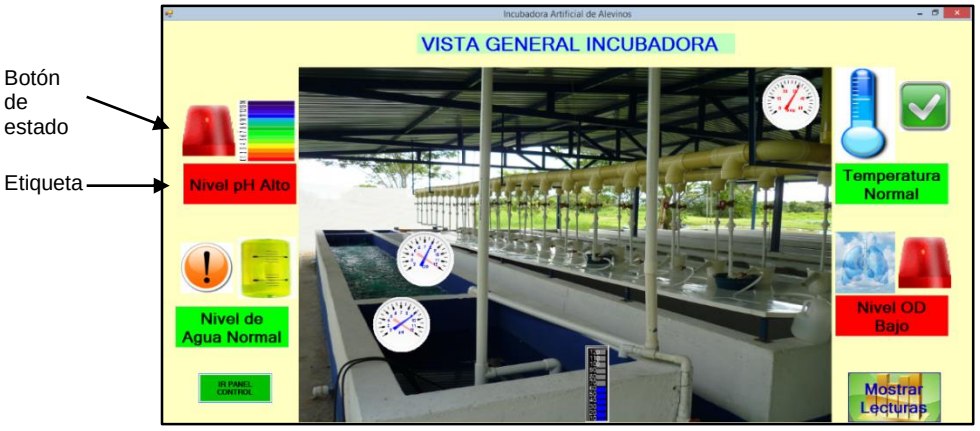
De igual forma se presenta las condiciones de normalidad o alarma actuales de las variables, representadas por medio de botones de estado (imágenes son cargadas desde archivos) y etiquetas que cambian de color y texto que describen la situación, esto se percibe como sigue:

 Indica comportamiento dentro de los parámetros establecidos.

 Señala condición de cercanía a situación de baja o alta.

 Presencia de alarma por estado de baja o alta.

Figura 59. Formulario inicial del SCADA



Fuente: Autores

Para lograr lo anterior se relaciona cada uno de los elementos mencionados anteriormente con la variable correspondiente en las propiedades del mismo las cuales son **Server, Group, Value y Variable** que hacen referencia a la variable del Arduino Mega en función del servidor de comunicaciones, el grupo en el que está definida, el valor que debe tomar y el nombre simbólico que se le ha dado, como se indica en la figura 60. Las opciones **Divider** y **Multiplier** ya fueron expuestas anteriormente.

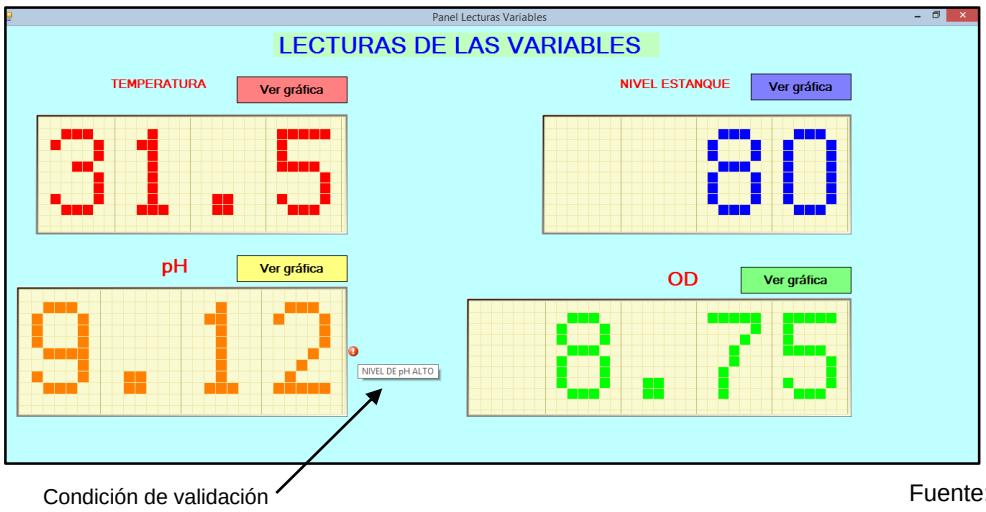
Figura 60. Ejemplo de propiedad para la variable temperatura

Scada	
(Server)	**ModbusRTU1**
CodeBindings	
Divider	**1**
Group	**ArduinoMega**
Multiplier	**1**
Value	**30**
Variable	**LM35**

Fuente: Autores

Desde el formulario *"Incubadora Artificial de Alevinos"* se puede acceder al siguiente cuyo nombre es "*Panel Lecturas Variables",* ver figura 61, por medio del botón "Mostrar Lecturas", en él se visualiza los valores actuales de las variables mostrados por medio de indicadores tipo LCD.

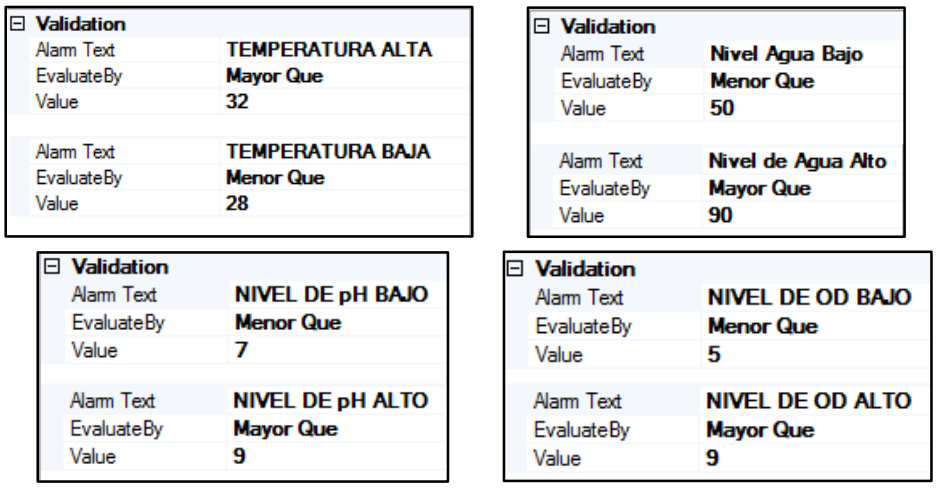
Figura 61. Formulario para visualizar los valores actuales de las variables



Fuente: Autores

A través de la propiedad **Validations** se definen para cada indicador LCD las condiciones de aviso, sobre el valor de la variable a la que esté vinculado éste, de forma que si en la ejecución del proyecto se cumple alguna de esas condiciones se mostrará un mensaje de alerta junto al indicador. A continuación en la figura 62 se muestran las matrices de validación para temperatura, nivel, pH y OD.

Figura 62. Matrices de validación para cada variable

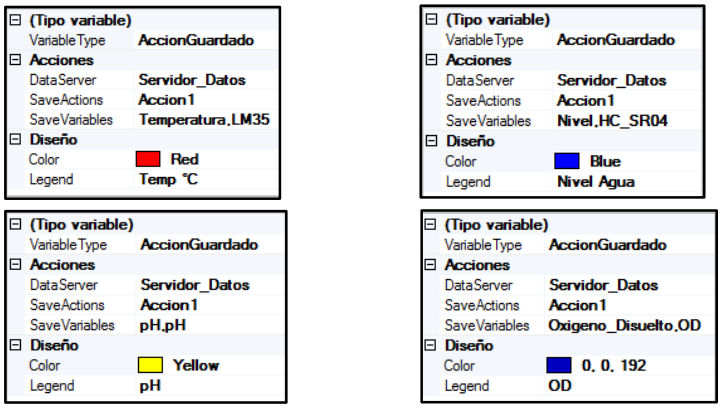


Fuente: Autores

Asimismo, cada variable tiene vinculado un botón para acceder a un formulario donde se despliega una gráfica de tendencia que permite analizar la evolución a lo largo del tiempo y guardar los datos obtenidos en una base de datos que debe ser seleccionada en la ventana de servidores como se vio anteriormente. En el control de tendencia se deben hacer ajustes a las propiedades del mismo de acuerdo a la variable que se muestra, para el caso del proyecto se configura cada una de la siguiente manera:

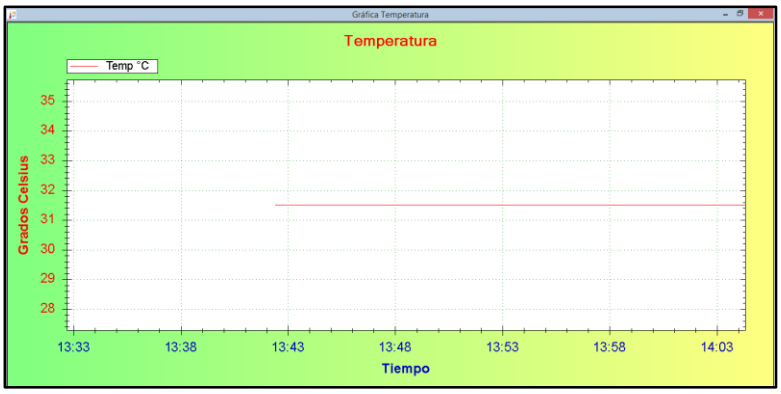
- **Frequency:** Se emplea una frecuencia de actualización de 1 segundo.
- **Period:** El intervalo de tiempo para mostrar en la gráfica es de 60 minutos.
- **Series:** Se indica la variable a representar en cada gráfica por medio del editor de serie como se detalla en la figura 63. En la figura 64 se evidencia un ejemplo de gráfica de tendencia.

Figura 63. Configuración editor de serie para cada variable



Fuente: Autores

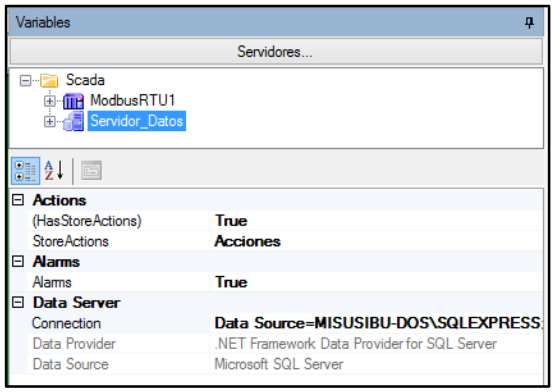
Figura 64. Ejemplo gráfica de tendencia para temperatura



Fuente: Autores

Para habilitar la acción de guardado debe ser puesto a TRUE la propiedad **HasStoreActions** del servidor de datos creado, y de esta manera ser definidas éstas en la propiedad **StoreActions**, desde la cual se accede al *"Editor de Acciones de Guardado"*, como se observa en la figura 65.

Figura 65. Configuración de almacenamiento en base de datos



Fuente: Autores

En dicho editor se establece el nombre de la tabla "dbo.Variables_Incubadora" que previamente se ha creado en el programa SQL EXPRESS la cual está conformada por cinco campos, ver la tabla 11, la cual hace parte de la base de datos INCUBAPECES. El tipo de guardado es establecido a **cadencia** con un tiempo de 1 segundo como se ve en la figura 66.

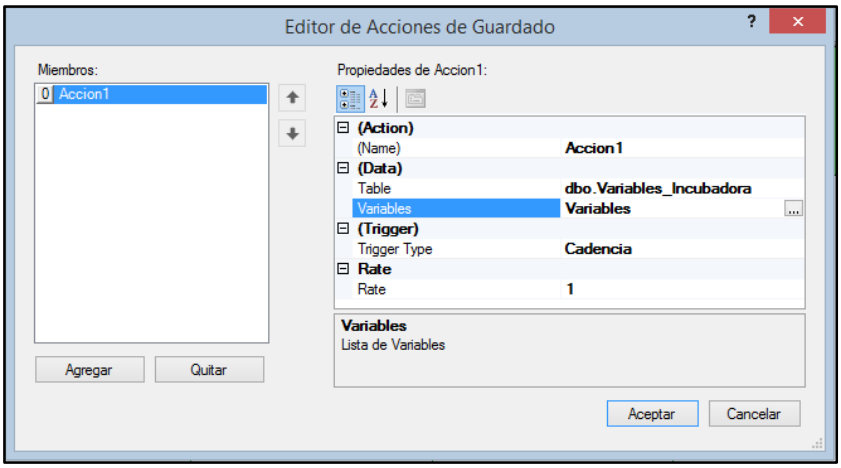
Tabla 11. Tabla de la base de datos INCUBAPECES

MISUSIBU-DOS\SQL...iables_Incubadora

Nombre de columna	Tipo de datos	Permitir val...
Fecha	datetime	☑
Temperatura	decimal(3, 1)	☑
Nivel_Agua	int	☑
pH	float	☑
Oxigeno_Disuelto	float	☑

Fuente: Autores

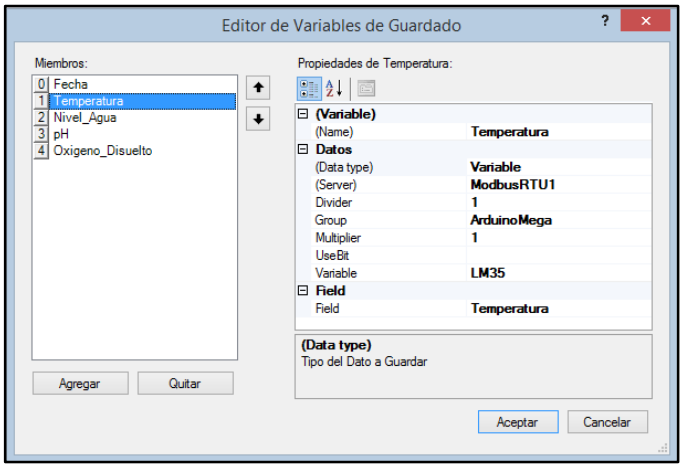
Figura 66. Editor de acción de guardado de los datos.



Fuente: Autores

Al seleccionar la propiedad **Variables,** como se visualiza en la figura 67, se accede al *"Editor de Variables de Guardado"* donde se relaciona a cada una de ellas y el campo (Field) de la tabla en donde se almacenará su valor correspondiente.

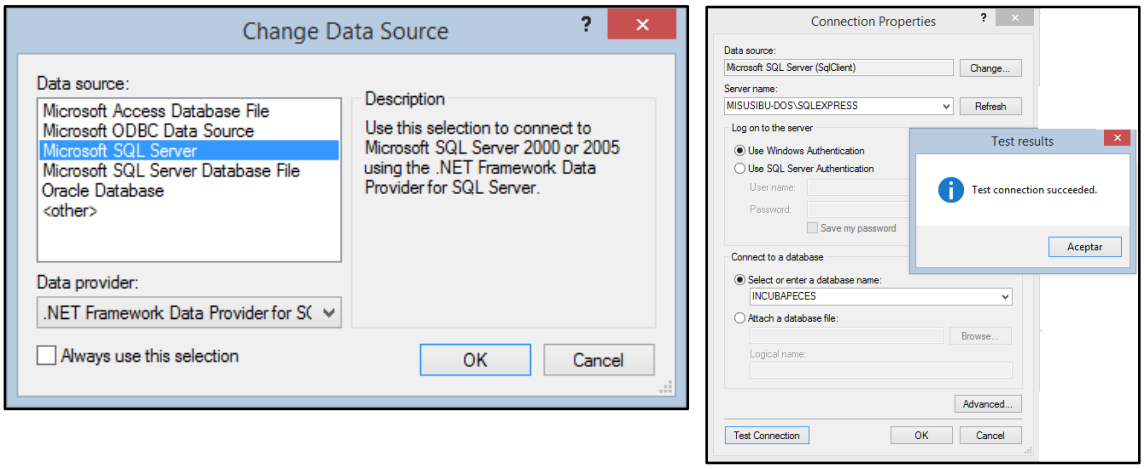
Figura 67. Ejemplo de configuración para la variable de temperatura



Fuente: Autores

Para establecer la conexión con el motor de base de datos SQLEXPRESS se selecciona la propiedad **Connection** del servidor de datos lo cual activa el asistente en el cual se escoge la fuente de datos en este caso Microsoft SQL Server, enseguida se indica el nombre del servidor (‘nombre equipo’ \SQLEXPRESS) y la base de datos creada (INCUBAPECES). La autenticación del usuario para el motor de base de datos puede darse por Windows o por Servidor SQL, se escoge el primero. Terminada la selección de las propiedades de la conexión se realiza la comprobación para verificar su validez. Ver figura 68.

Figura 68. Establecimiento de las propiedades de la conexión

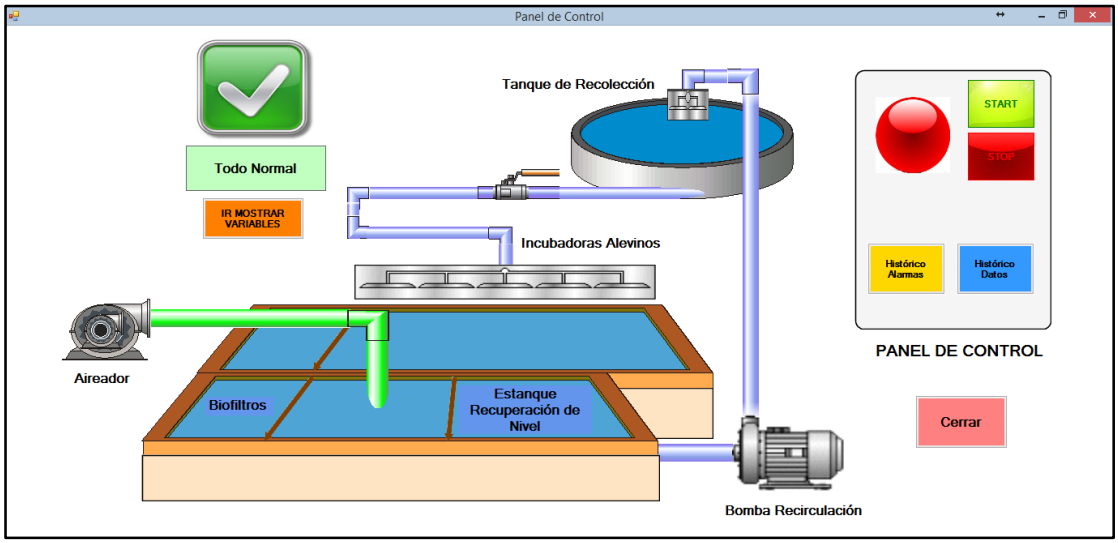


Fuente: Autores

Por otra parte, desde el mismo formulario *“Incubadora Artificial de Alevinos”* se puede ingresar por medio del botón ubicado en la parte inferior izquierda, a otro denominado *“Panel de Control”* , lo cual se muestra en la figura 69 y 70, en el cual se puede apreciar una representación gráfica del sistema de recirculación de agua empleado para la incubación de los alevinos.

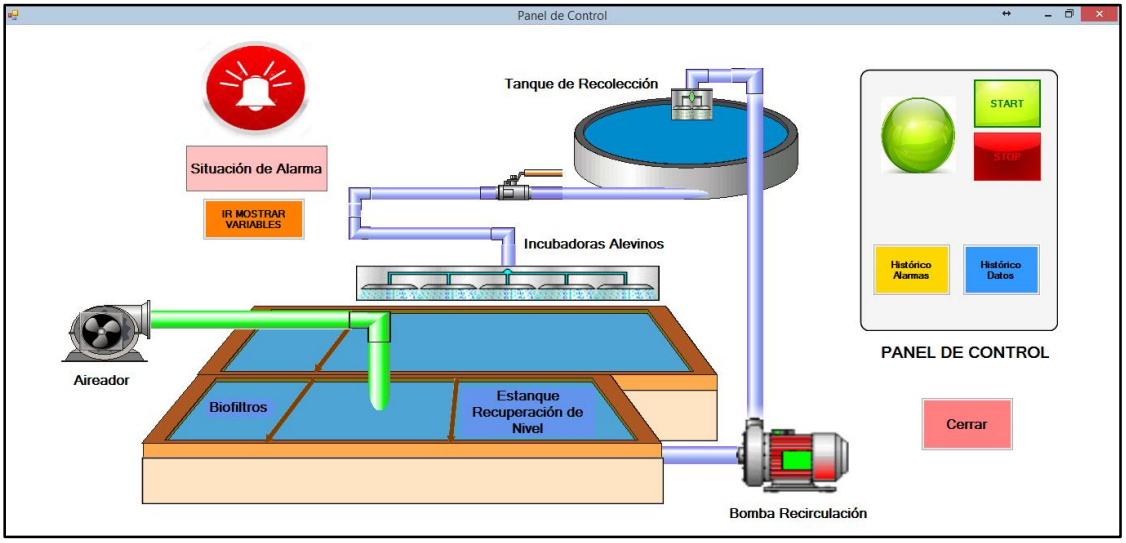
En este formulario se muestran dos pulsadores que permiten activar el relevo encargado del suministro eléctrico a los actuadores (bomba y aireador), como se definió anteriormente, al tiempo que por medio de botones de estado cargados con imágenes alegóricas a los dispositivos se pueden ver si está ON u OFF.

Figura 69. Formulario representación gráfica sistema apagado



Fuente: Autores

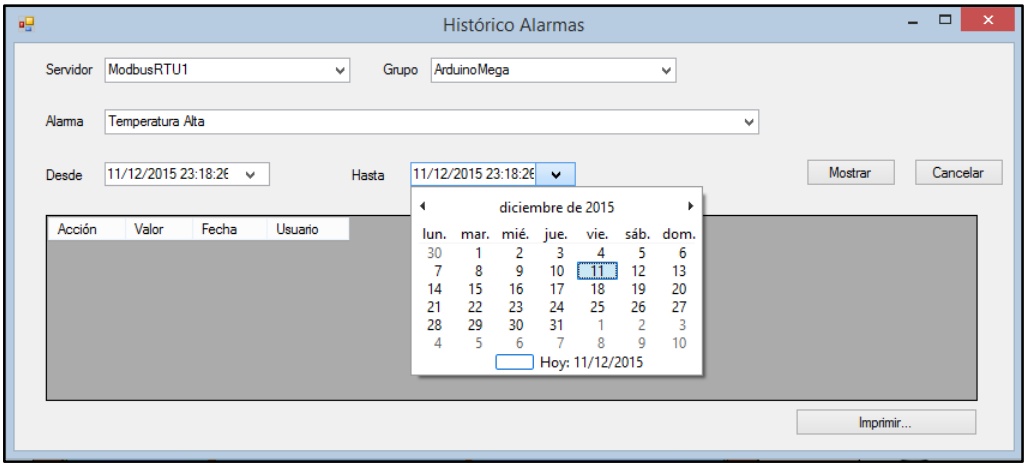
Figura 70. Formulario representación gráfica sistema encendido



Fuente: Autores

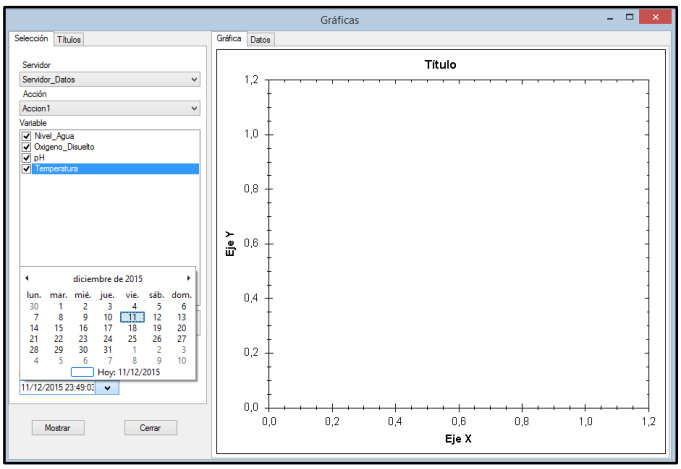
También cuenta con dos botones, uno de ellos permite desplegar el histórico de alarmas y el otro el histórico de datos, provenientes de la base de datos creada para tal fin, referenciados en el formulario de la misma forma respectivamente. Para dar la funcionalidad anterior a los botones es necesario seleccionar de la propiedad **Action** correspondiente, la opción *MostrarGrafica* y *MostrarHistoricoAlarmas.* Ver figuras 71 y 72.

Figura 71. Ventana que muestra el histórico de alarmas



Fuente: Autores

Figura 72. Ventana que muestra el histórico de datos



Fuente: Autores

7.9 HMI PARA ARDUINO

Por medio de la aplicación HMI Controller para Arduino que funciona con el sistema operativo Android, desarrollado por Sergio Daniel Castañeda, se enlaza la placa Arduino Mega con una tableta de 10 pulgadas a través de la conexión Bluetooth por medio del módulo HC-05, mostrado en la figura 73, el cual debe estar configurado a la misma cantidad de bits por segundo que el puerto serial al cual se conecta para lo que nos atañe en 9600 bps.

Figura 73. Módulo Bluetooth HC-05 y tableta

Fuente: Autores

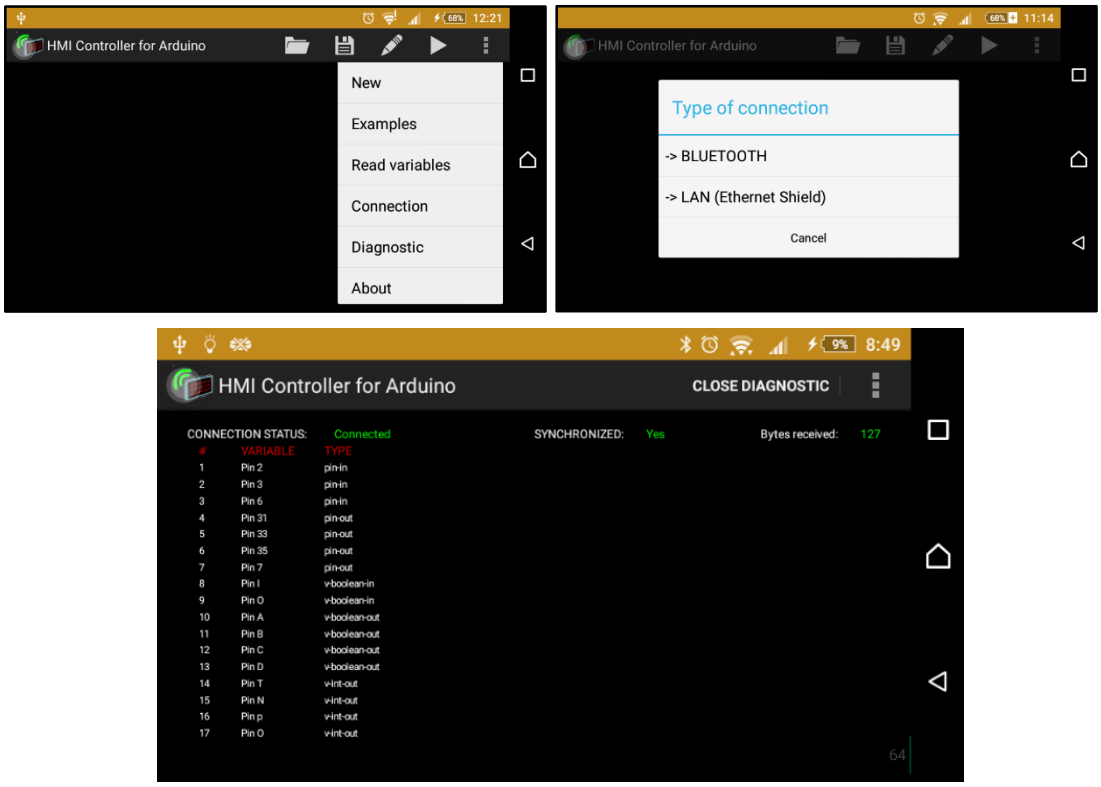
Para crear el HMI se realiza la programación en una versión modificada de la Arduino IDE donde, partiendo de las variables establecidas en el código implementado para la conexión SCADA, se incluye la función *Hmi.attachIntOut* para la transmisión de los valores de las lecturas formato entero ya que no es posible otro tipo, la función *Hmi.attachBooleanOut* para presentar las alarmas de las mismas y *Hmi.attachBooleanIn* con el objeto de implementar los pulsadores de control de encendido de los actuadores.

```
//Definir variables

int tempC;
int distancia;
int pH;
int OD;
boolean turnOnIncuba = false, turnOffIncuba = false;   //Botones encendido y apagado
boolean led_Temp = false;     //Variable Booleana para indicar alarma de temperatura a HMI
boolean led_Nivel = false;   //Variable Booleana para indicar alarma de nivel a HMI
```

```
boolean led_pH = false;      //Variable Booleana para indicar alarma de pH a HMI
boolean led_OD = false;     //Variable Booleana para indicar alarma de oxigeno disuelto a HMI

void setup()
         {
         Serial2.begin(9600); //seleccion de baud rate para puerto 2 serial a 9600 para
comunicacion bluetooh

          //Variables globales del sistema HMI

         Hmi.attachBooleanIn(turnOnIncuba,'I');   //Virtual INPUT, bool I para HMI
         Hmi.attachBooleanIn(turnOffIncuba,'O');  //Virtual INPUT, bool O para HMI
         Hmi.attachIntOut(tempC,'T');             //Virtual OUTPUT, int T
         Hmi.attachIntOut(distancia,'N');         //Virtual OUTPUT, int N
         Hmi.attachIntOut(pH,'p');                //Virtual OUTPUT, int p
         Hmi.attachIntOut(OD,'O');                //Virtual OUTPUT, int O
         Hmi.attachBooleanOut(led_Temp,'A');      //Virtual OUTPUT, bool A para HMI
         Hmi.attachBooleanOut(led_Nivel,'B');     //Virtual OUTPUT, bool B para HMI
         Hmi.attachBooleanOut(led_pH,'C');        //Virtual OUTPUT, bool C para HMI
         Hmi.attachBooleanOut(led_OD,'D');        //Virtual OUTPUT, bool D para HMI
        }

void loop()
          {
           if (tempC < 280 || tempC > 320)
              {led_Temp = false;
              }
           else
              {led_Temp = true;
              }
           if (distancia < 50 || distancia > 90)
              {led_Nivel = false;
              }
           else
              {led_Nivel = true;
              }
           if (pH < 700 || pH > 900)
              {led_pH = false;
              }
           else
              {led_pH = true;
              }
           if (OD < 500 || OD > 900)
              {led_OD = false;
              }
           else
              {led_OD = true;
              }

       Hmi.hardSerial(Serial2); //implementar conexión Bluetooth puerto 2 serial

     }
```

Una vez cargado el código al Arduino Mega se selecciona la conexión desde la aplicación en el dispositivo Android. Para corroborar que existe sincronismo se escoge en el menú la opción **Diagnostic** que permite ver las variables señaladas en el código. Lo anterior se visualiza en la figura 74.

Figura 74. Conexión de la aplicación HMI al Arduino Mega

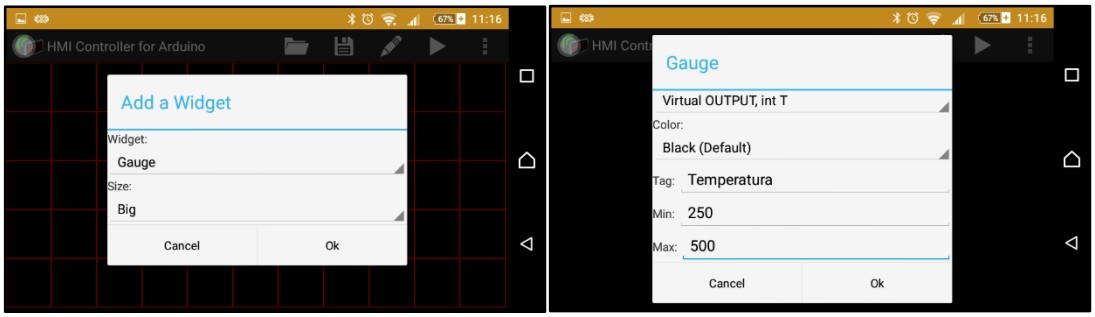


Fuente: Autores

A continuación se establecen las relaciones entre las variables y los elementos disponibles para mostrarlas por medio de agregar *widget* pulsando sobre el icono en forma de lápiz. Con respecto a la temperatura se escoge la representación **Gauge** para el cual se define el tamaño, el color, la etiqueta, los valores mínimo y máximo del mismo y la posición dentro del HMI. Ver figura 75. Se tiene en cuenta que solo se pueden mostrar datos de tipo entero, es por esto que los valores seleccionados oscilan entre 200 y 500 con el objeto de mostrar temperaturas entre 20,0 °C y 50,0 °C. Dicha condición se tiene en cuenta desde el la programación del Arduino vista en los apartados de la creación de los códigos para leer los sensores.

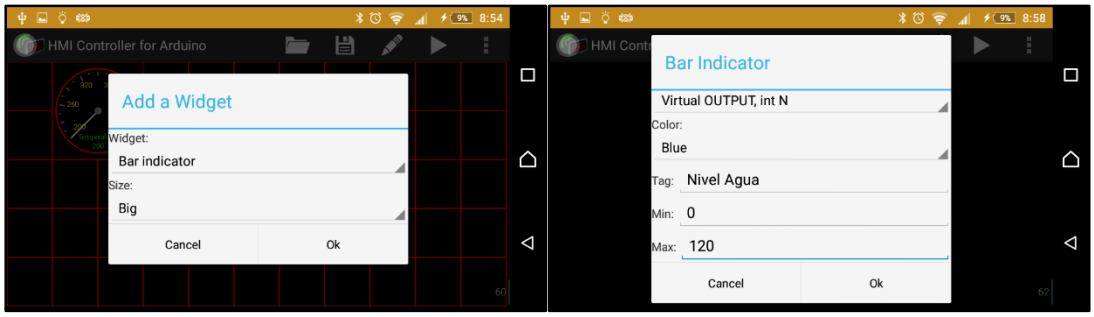
Para la visualización del nivel se emplea **Bar Indicator** grande de color azul con valores entre 0 y 120 (ver figura 76), para el pH y el OD, **7-segments Display** grandes, de color amarillo para el primero y azul agua para el segundo (ver figuras 77 y 78). Al igual que el *widget* de temperatura se les asignan la etiqueta y el lugar correspondiente dentro de la pantalla.

Figura 75. Configuración componente para mostrar temperatura



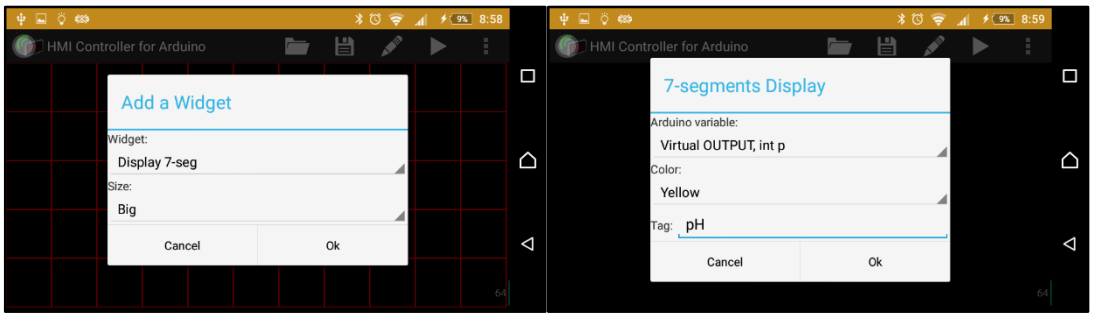
Fuente: Autores

Figura 76. Configuración componente para mostrar nivel



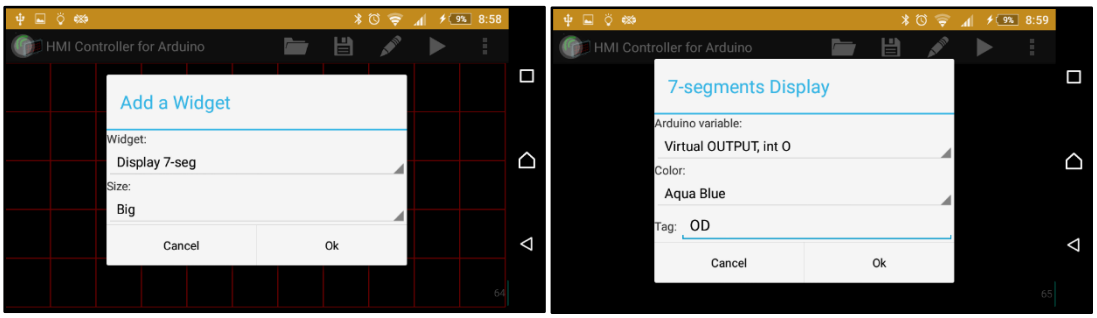
Fuente: Autores

Figura 77.Configuración componente para mostrar pH



Fuente: Autores

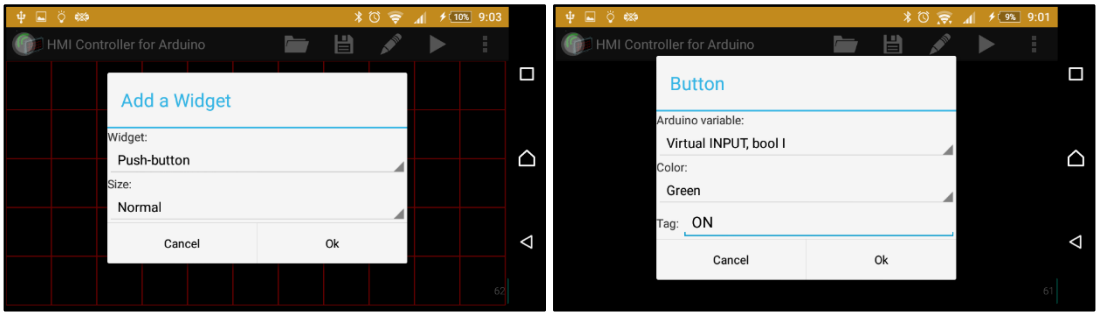
Figura 78. Configuración componente para mostrar OD



Fuente: Autores

Así mismo, como se observa en la figura 79, también se crea los pulsadores con el **push-button** y el indicador de encendido de los actuadores con el **led**.

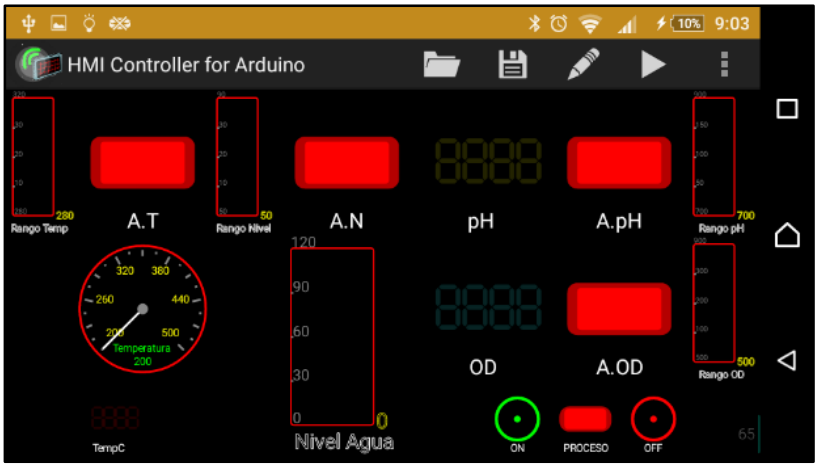
Figura 79. Configuración componente para pulsador encendido



Fuente: Autores

Con el propósito de visualizar la ocurrencia de las posibles alarmas se recurre al *widget* que representa a un LED escogiendo la opción de *red / green* vinculando a cada una de las variables booleanas dispuestas para ello permitiendo ver los dos estados, rojo para alarma y verde para normal, asimismo se utilizan indicadores tipo nivel para comprobar si las variables se encuentran dentro del rango. En la figura 80 es posible detallar el resultado de la construcción del HMI.

Figura 80. Presentación final del HMI



Fuente: Autores

7.10 Configuración correo electrónico envío notificación de alarma

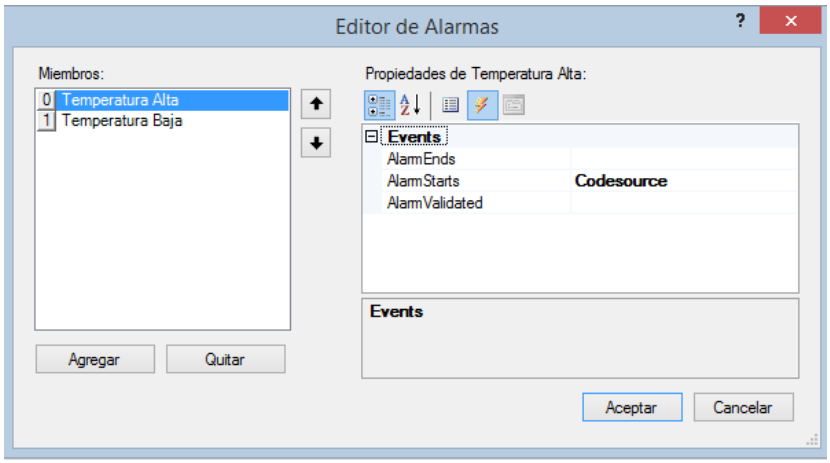
Con el objeto de enviar un e-mail que permita notificar cuando se ha iniciado una alarma en cualquiera de las variables es necesario crear una cuenta en Gmail a nombre del proyecto desarrollado, como se ve en la figura 81.

Figura 81. Cuenta en Gmail para enviar notificaciones de alarma.

Fuente: Autores

En la ventana del Editor de Alarmas del SCADA y por medio del editor de eventos en *AlarmStarts*, como se ve en la figura 82, se introduce el código en VisualBasic necesario para lograr que el servidor de Acimut Monitoriza por medio de la creación de un objeto SmtpClient establezca la conexión con el servidor de correo y un objeto MailMessage permita definir el remitente, los destinatarios, el asunto y cuerpo del mensaje.

Figura 82. Establecer evento de correo electrónico



Fuente: Autores

El código empleado para notificar la alarma por alta temperatura es el siguiente:

```
Imports System
Imports System.Drawing
Imports System.Data
Imports System.Xml
Imports System.Windows.Forms
Imports Microsoft.VisualBasic
Imports System.Net.Mail

 Namespace Scada
       Public Class Monitoriza
              Public Sub AlarmStarts(ByRef clsVariables As Object)

                     Try
                            'Crear un objeto de mensaje de correo
                            Dim MyMailMessage As New MailMessage()

                            'El campo From requiere una instancia del tipo MailAddress
                            MyMailMessage.From = New MailAddress("incubapeces@gmail.com")
```

```
        'El campo To es una colección de tipos MailAddress

        MyMailMessage.To.Add("misusibu@hotmail.com,alejandraparra649@hotmail.com,lcartagena@
        itfip.edu.com,diegoparlag@gmail.com,lenguajeitfip@gmail.com")

        'Este es el asunto del mensaje
        MyMailMessage.Subject = "Inicio de Alarma por Alta Temperatura"

        'Cuerpo del mensaje donde se especifica de donde se toma el valor de la variable
        Dim strBody As String

        strBody = String.Format("{0}. La variable ha alcanzado el valor {1}", Now(),
        clsVariables.Variables("ModbusRTU1","ArduinoMega","LM35",""))

        MyMailMessage.Body = strBody

        'Crear el objeto SmtpClient y especificar las credenciales del usuario

        Dim SMTPServer As New SmtpClient("smtp.gmail.com")
        SMTPServer.Port = 587
        SMTPServer.Credentials = New System.Net.NetworkCredential( "incubapeces@gmail.com",
        "INCUBAPECES2015")
        SMTPServer.EnableSsl = True

        'Enviar el mensaje

                                Try
                                        SMTPServer.Send(MyMailMessage)
                                        Catch ex As SmtpException
                                        MessageBox.Show("Ex1=" & ex.Message)
                                End Try

                                Catch ex2 As SmtpException
                                MessageBox.Show("Ex2=" & ex2.Message)
                        End Try
                End Sub
        End Class
End Namespace
```

De igual manera se crea el código para las otras notificaciones teniendo en cuenta de cambiar el asunto y la variable a enviar de acuerdo a las designaciones contempladas en la creación del SCADA (servidor, grupo, etc.).

8. CRONOGRAMA

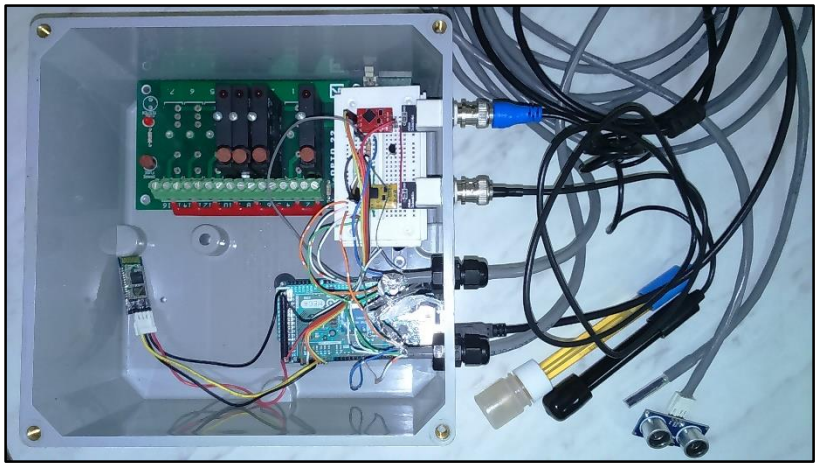
Cronograma	Septiembre de 2015				Octubre de 2015				Noviembre de 2015				Diciembre de 2015				Enero de 2016			
Actividad / Semana	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Investigar estado del arte funcionamiento incubadora	X	X	X	X	X	X														
Realizar mediciones de variables					X	X	X	X												
Estudiar marco teórico sobre equipos y software de supervisión necesario.						X	X	X	X											
Diseñar sistema de monitoreo								X	X	X	X	X								
Presentación de resultados del proyecto													X	X						
Sustentación proyecto de monitorización																			X	

9. RESULTADOS

Teniendo en cuenta que la economía nacional busca altos índices de competitividad, es necesario buscar alternativas que optimicen la productividad en los procesos que se desarrollan en las empresas, reduciendo la ineficiencia e impidiendo al máximo la presencia de situaciones que en últimas terminen por ocasionar pérdidas en la producción, lo antepuesto puede ser reducido con la implementación de sistemas de monitoreo que ayuden a llevar a un mejor desenvolvimiento de la operación. En miras de lo anterior, los elementos que fueron considerados en el diseño de la propuesta fueron implementados en su totalidad cubriendo los objetivos que el proyecto persigue, consiguiendo que la operación de la incubadora de alevinos sea más manejable y predecible mermando, por tanto, el margen de error en la toma de decisiones frente a los resultados. El prototipo terminado se puede ver en la figura 83.

La monitorización de las condiciones físico-químicas del agua cumple con las expectativas del personal que la manipula y de las directivas de la compañía. Partiendo de que el proceso anterior a la implementación conllevaba la prestación de gran atención, cuidado e inversión de tiempo, la solución desarrollada permite que la cantidad de tareas ejecutadas por los operarios disminuya.

Figura 83. Prototipo terminado

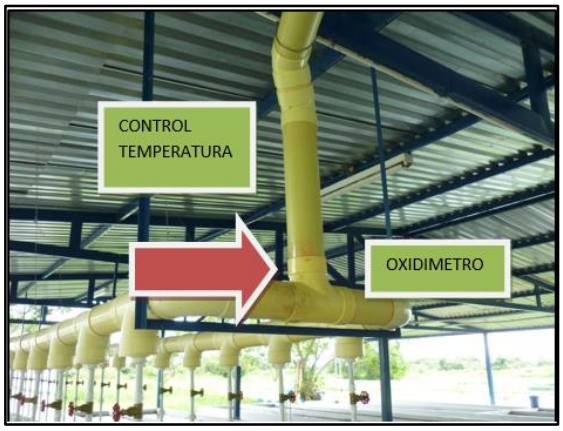


Fuente: Autores

9.1 INSTALACIÓN DE LOS SENSORES EN LA ESTRUCTURA DE LA INCUBADORA

Una vez terminado el montaje de los elementos dentro de la caja se realizó la prueba de funcionamiento ubicando cada uno de los sensores en los puntos designados en la estructura de la incubadora, ver figuras 84 y 85, se conectó el prototipo al computador donde se ejecuta la aplicación SCADA y se procedió a verificar el comportamiento de todas las partes (hardware y software).

Figura 84. Ubicación de los sensores de temperatura y OD



Fuente: Piscícola Agroavícola San Marino

Figura 85. Ubicación sensores de pH y nivel

Fuente: Piscícola Agroavícola San Marino

9.2 COMPROBACION DE LA CALIBRACION DE LOS SENSORES

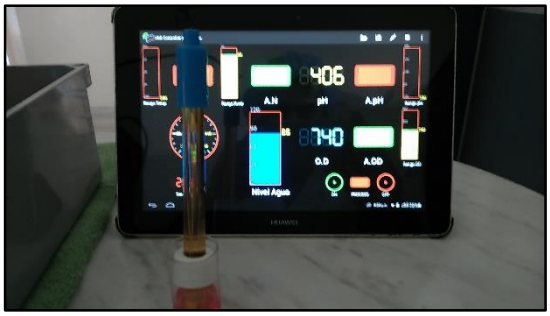
Se utiliza un termómetro digital de marca Cooper para establecer el correcto funcionamiento de la sonda de temperatura, como se puede ver en la figura 86, encontrándose que existe una diferencia de 0.2 °C que está entre lo esperado en la medición del sensor LM35DZ.

Figura 86. Lectura de la temperatura

Fuente: Autores

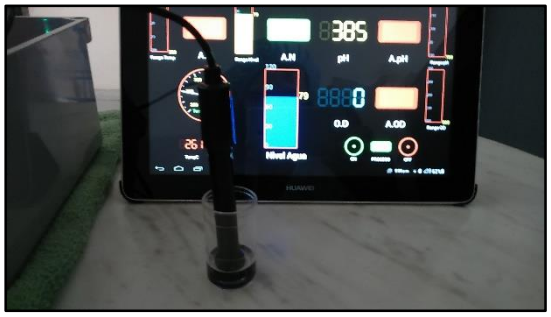
En cuanto a las sonda de pH y OD se emplean sustancias químicas que sirven de patrón, para el pH una certificada de 4 y para el OD de 0 mg/L, los resultados pueden verse en las figuras 87 y 88, donde se aprecia que la primera presenta una medición de 4.06 y la segunda de 0.

Figura 87. Comprobación calibración pH



Fuente: Autores

Figura 88. Comprobación calibración OD

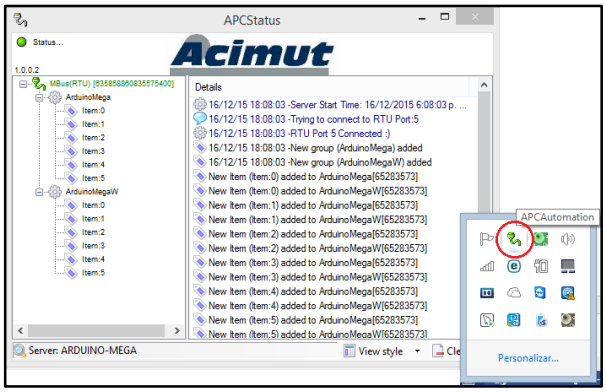


Fuente: Autores

9.3 PUESTA EN MARCHA DEL SCADA Y EL HMI

En primera instancia se accede al servidor de comunicaciones de la aplicación Acimut Monitoriza para observar que se estableció la conexión con la tarjeta Arduino Mega, como se evidencia en la figura 89, indicando que no existe ninguna problemática.

Figura 89. Conexión establecida del SCADA con Arduino Mega



Fuente: Autores

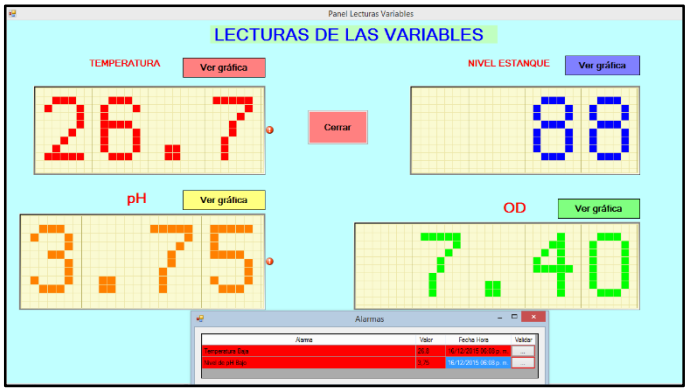
Paso seguido se analiza el estado de las variables de la incubadora observándose que existen alarmas por temperatura y nivel de pH bajo como se observa en la figura 90.

Figura 90. Estado actual de las variables

Fuente: Autores

De igual forma se accede al formulario de visualización de los valores de las variables, mostrado en la figura 91, encontrándose que la temperatura está en 26,7 °C, el nivel de agua se ubica en 88, el pH tiene una medida de 3,75 y el oxígeno se encuentra en 7,40, al tiempo que se observa el panel de alarmas señalando las alarmas identificadas anteriormente junto con la fecha y la hora que se producen.

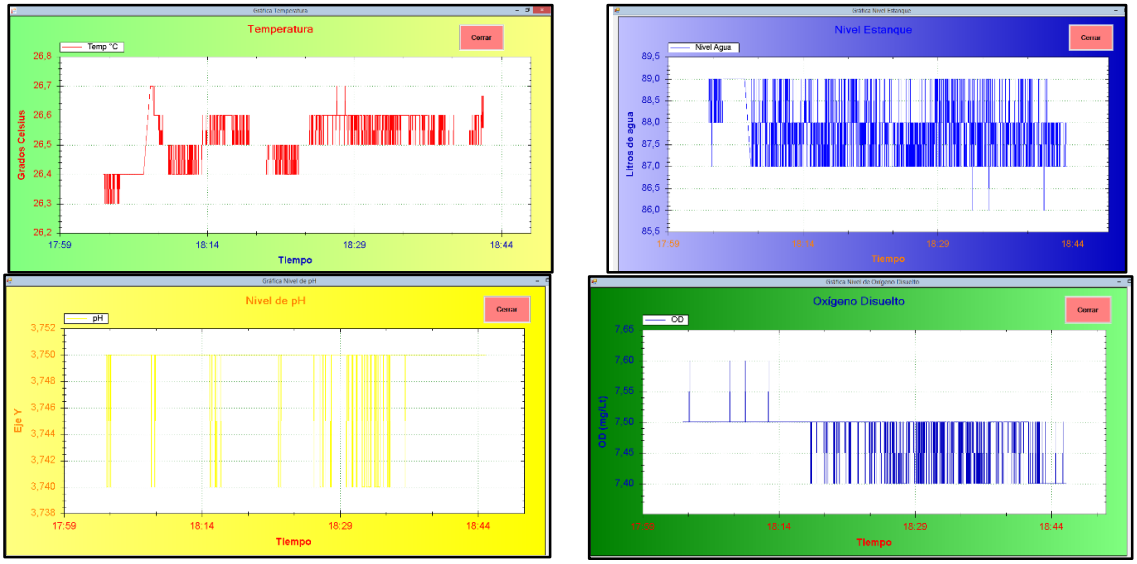
Figura 91. Visualización de los valores de las variables y alarmas



Fuente: Autores

Las gráficas obtenidas de la adquisición de los datos por cada variable, presentadas en la figura 92, también pueden ser observadas y posteriormente analizadas por el técnico piscícola quien debe determinar los ajustes necesarios para cumplir con los requerimientos.

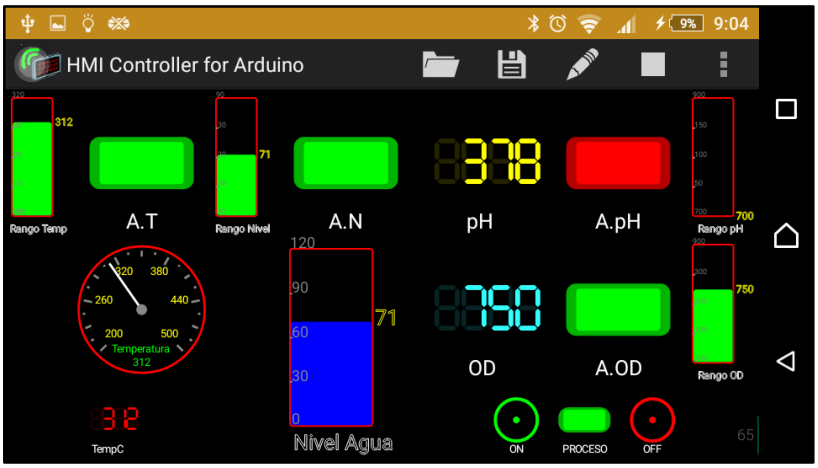
Figura 92. Gráficas del comportamiento de las variables



Fuente: Autores

Los valores mostrados en la pantalla del dispositivo Android, en este caso la tableta de 10 pulgadas, como se aprecia en la figura 93, también permiten establecer el estado actual de las variables de la incubadora de una manera rápida. Se observa la presencia de alarma por bajo nivel de pH.

Figura 93. Visualización valores variables en Android



Fuente: Autores

Los datos y alarmas almacenados en la base datos pueden ser consultados para su respectivo análisis y si se requiere para crear gráficas de tendencias y demás validaciones, ejecutando los siguientes códigos directamente en SQLEXPRESS:

```
/****** Script para el comando SelectTopNRows de SSMS  ******/
SELECT TOP 10000 [Fecha]
      ,[Temperatura]
      ,[Nivel_Agua]
      ,[pH]
      ,[Oxigeno_Disuelto]
  FROM [INCUBAPECES].[dbo].[Variables_Incubadora]


/****** Script para el comando SelectTopNRows de SSMS  ******/
SELECT TOP 1000 [ID]
      ,[Servidor]
      ,[Grupo]
      ,[Variable]
      ,[TextoAlarma]
      ,[Accion]
      ,[Valor]
      ,[Usuario]
      ,[FechaHora]
  FROM [INCUBAPECES].[dbo].[Alarmas]
```

El resultado de ejecución de los códigos puede verse en la tabla 12 y 13.

Tabla 12. Consulta de variables en base de datos

	Fecha	Temperatura	Nivel_Agua	pH	Oxigeno_Disuelto
4313	2015-12-16 18:08:21.950	26.7	87	3,75	7,5
4314	2015-12-16 18:08:22.950	26.7	87	3,75	7,5
4315	2015-12-16 18:08:23.950	26.7	88	3,75	7,5
4316	2015-12-16 18:08:24.950	26.7	88	3,75	7,5
4317	2015-12-16 18:08:25.950	26.7	88	3,75	7,5
4318	2015-12-16 18:08:26.950	26.7	88	3,75	7,5
4319	2015-12-16 18:08:27.950	26.7	87	3,74	7,5
4320	2015-12-16 18:08:28.950	26.7	87	3,75	7,5
4321	2015-12-16 18:08:29.950	26.7	87	3,75	7,5
4322	2015-12-16 18:08:30.950	26.6	87	3,74	7,5
4323	2015-12-16 18:08:31.950	26.6	88	3,75	7,5
4324	2015-12-16 18:08:32.950	26.7	89	3,75	7,5
4325	2015-12-16 18:08:33.950	26.7	88	3,75	7,5
4326	2015-12-16 18:08:34.950	26.7	87	3,75	7,5
4327	2015-12-16 18:08:35.950	26.7	87	3,75	7,5
4328	2015-12-16 18:08:36.950	26.6	88	3,75	7,6
4329	2015-12-16 18:08:37.950	26.6	88	3,75	7,5
4330	2015-12-16 18:08:38.950	26.6	88	3,75	7,5
4331	2015-12-16 18:08:39.950	26.6	88	3,75	7,5
4332	2015-12-16 18:08:41.137	26.6	88	3,75	7,5
4333	2015-12-16 18:08:42.153	26.6	87	3,75	7,5

Fuente: Autores

Tabla 13. Consulta de alarmas en base de datos

	ID	Servidor	Grupo	Variable	TextoAlarma	Accion	Valor	Usuario	FechaHora
22	22	ModbusRTU1	ArduinoMega	LM35	Temperatura Alta	Inicio	32,0		2015-12-13 01:21:23.997
23	23	ModbusRTU1	ArduinoMega	LM35	Temperatura Alta	Validada	32,0	técnico piscicola	2015-12-13 01:21:48.807
24	24	ModbusRTU1	ArduinoMega	pH	Nivel de pH Bajo	Validada	3,67	técnico piscicola	2015-12-13 01:21:49.470
25	25	ModbusRTU1	ArduinoMega	pH	Nivel de pH Bajo	Inicio	3,66		2015-12-13 01:33:56.203
26	26	ModbusRTU1	ArduinoMega	LM35	Temperatura Alta	Inicio	32,0		2015-12-13 01:35:00.327
27	27	ModbusRTU1	ArduinoMega	LM35	Temperatura Alta	Fin	31,9		2015-12-13 01:36:44.383
28	28	ModbusRTU1	ArduinoMega	pH	Nivel de pH Bajo	Inicio	3,67		2015-12-13 01:43:00.453
29	29	ModbusRTU1	ArduinoMega	LM35	Temperatura Alta	Inicio	32,0		2015-12-13 01:43:22.790
30	30	ModbusRTU1	ArduinoMega	pH	Nivel de pH Bajo	Inicio	3,66		2015-12-13 01:52:48.817
31	31	ModbusRTU1	ArduinoMega	LM35	Temperatura Alta	Inicio	32,0		2015-12-13 01:53:13.517
32	32	ModbusRTU1	ArduinoMega	LM35	Temperatura Alta	Fin	31,9		2015-12-13 01:54:27.747
33	33	ModbusRTU1	ArduinoMega	LM35	Temperatura Alta	Validada	32,0	técnico piscicola	2015-12-13 01:56:08.417
34	34	ModbusRTU1	ArduinoMega	pH	Nivel de pH Bajo	Validada	3,66	técnico piscicola	2015-12-13 01:56:09.937
35	35	ModbusRTU1	ArduinoMega	LM35	Temperatura B...	Inicio	26,5		2015-12-16 18:03:22.090
36	36	ModbusRTU1	ArduinoMega	pH	Nivel de pH Bajo	Inicio	3,75		2015-12-16 18:03:22.170
37	37	ModbusRTU1	ArduinoMega	LM35	Temperatura B...	Inicio	26,8		2015-12-16 18:08:09.777
38	38	ModbusRTU1	ArduinoMega	pH	Nivel de pH Bajo	Inicio	3,75		2015-12-16 18:08:09.777
39	39	ModbusRTU1	ArduinoMega	LM35	Temperatura B...	Validada	26,8	técnico piscicola	2015-12-16 18:48:49.390
40	40	ModbusRTU1	ArduinoMega	pH	Nivel de pH Bajo	Validada	3,75	técnico piscicola	2015-12-16 18:48:50.313

Fuente: Autores

Asimismo, se reciben por e-mail notificaciones de la alarmas, en este caso con respecto al bajo nivel de pH, como se detalla en la figura 94.

Figura 94. Notificación por correo electrónico

Fuente: Autores

10. CONCLUSIONES

- De acuerdo a la investigación que se llevó a cabo y a la información recolectada de artículos, libros, consulta al técnico en piscicultura, se estableció que los parámetros más importantes que nos brinda una acertada evaluación de la calidad de agua para la producción de los alevinos son el Oxígeno Disuelto, Temperatura y potencial de Hidrógeno los cuales se tuvieron en cuenta en el desarrollo del prototipo, conformado éste por el hardware y software.

- En la construcción de la sonda de temperatura es importante tener en cuenta la utilización de resistencia y condensadores que permitan hacer frente a las interferencias presentes en el área donde se implementa el sistema.

- Se escogen los sensores para pH y oxígeno disuelto de la marca Atlas Scientific porque disponen de circuitos embebidos que ofrecen características importantes al momento del diseño y la implementación que evitan menor trabajo de procesamiento por parte de la tarjeta que adquiere los datos.

- El código usado que permite establecer el protocolo ModBus en la placa de Arduino facilita la implementación de SCADA con cualquier programa utilizado para conexión con PLC lo que brinda grandes posibilidades de creación de soluciones de automatización en diferentes áreas.

- Sobre la base de las consideraciones relacionadas en el desarrollo del proyecto se identifica que con las plataformas de hardware y recursos de software libre, es posible implementar sistemas de supervisión y adquisición de datos de costo reducido. El presente trabajo da fundamento a lo anterior al realizarse el diseño e implementación de un sistema de monitorización SCADA para la incubadora artificial de alevinos, que afecta positivamente el presupuesto del proyecto de automatización industrial y que cumple con los objetivos propuestos.

- Para cada uno de los aspectos a solucionar dentro de la propuesta se definieron objetivos concretos que, junto con las condiciones actuales del sistema de incubación y los procedimientos de operación, integran la base fundamental para el análisis y desarrollo del trabajo.

- Se verificó el correcto funcionamiento del sistema operando de forma integrada, mediante una prueba operativa durante un intervalo de tiempo, permitiendo

comprobar que las lecturas de los parámetros se encuentran dentro de lo esperado según las características de los dispositivos empleados.

- Se destaca la importancia y aplicaciones de la gestión de base de datos en los procesos automatizados, con el objetivo de realizar un buen control de la producción, mantenimiento del mismo e interpretación de la información para una adecuada toma de decisiones y la mejora continua del proceso.

- Con el propósito de resaltar la influencia de la automatización del sistema de incubación artificial se planteó un nuevo procedimiento de operación ajustado a la solución propuesta. Al hacer la comparación entre los dos procedimientos, el actual y el propuesto con la solución, se puede señalar que hay beneficios notables en aspectos como: reducción de los riesgos de fallas en el proceso, disminución del número de tareas repetitivas, que pudieran resultar en efectos negativos a la calidad de los alevinos y desempeño posterior en la etapa de engorde, de igual manera, existe también un ahorro en tiempo en la ejecución de las tareas.

- Es importante resaltar que la presentación de propuestas de automatización permite afianzar los conocimientos adquiridos en los procesos de formación del ITFIP ya que logra dimensionar la búsquedas de alternativas ante problemáticas como la presentada, asimismo, se adquieren destrezas de evaluación de sistemas y medición de variables con el objeto de realizar investigaciones sobre las tecnologías que se desarrollan en la actualidad o a futuro como es el caso de la incubación de alevinos de mojarra roja.

11. PRESUPUESTO DEL PROTOTIPO

Costo partes

• Kit Sensor OD	$ 922.600.
• Kit Sensor pH	$ 534.400.
• Base SSR OPTO22	$ 280.000.
• Tres SSR OPTO22	$ 195.000.
• Arduino Mega	$ 122.000.
• Caja PVC	$ 92.000.
• Módulo BT HC-05	$ 25.000.
• Botonera	$ 22.000.
• Adaptador 7.5vdc 1A	$ 18.000.
• Seis metros cable instrumentación	$ 25.000.
• Sensor Ultrasonido HCSR04	$ 10.000.
• Miniprotoboard	$ 6.000.
• Cables conexiones	$ 4.000.
• Sensor LM35DZ	$ 2.600.
TOTAL	**$ 2.258.600.**

Realización

• Personal científico	$ 120.000.
• Personal de apoyo	$ 40.000.
• Materiales e insumos	$ 2.258.600.
• Trabajo de campo	$ 150.000.
• Equipos	$ 80.000.
• Bibliografía	$ 25.000.
• Diseño e implementación	$ 800.000.
TOTAL	**$3.473.600.**

BIBLIOGRAFIA

Acimut Monitoriza for ARDUINO. [Artículo de internet] http://www.acimut.es/monitoriza/monitorizaforarduino.html [Consulta: Octubre 20 de 2015].

Arduino Mega 2560. [Artículo de internet] https://www.arduino.cc/en/Main/ArduinoBoardMega2560 [Consulta: Octubre 20 de 2015]

BARNABÉ, Gilbert. Acuicultura 1.Barcelona. Ediciones Omega S.A, 1991. 29-153 p.

BUCKLE, L. Fernando, BARON, Benjamín, HERNANDEZ, Mónica. Sistema de temperatura, oxígeno y salinidad para la experimentación en ecofisiología. Universidad Tecnológica de la Mixteca. Hidrobiológica 2003. 277 - 287 p.

COLOMBIA. CONGRESO DE COLOMBIA. Ley 13 de 1990. Por la cual se dicta el estatuto general de pesca. Diario oficial. Bogotá, 1990. 28 p.

Estación piscícola de Gigante. [Artículo de internet] http://www.ica.gov.co/getdoc/3c75d067-7f91-434a-b09e-07e470f0700b/Estacion-piscicola-de-Gigante.aspx [Consulta: Octubre 18 de 2015]

GALI MERINO, Oscar y SAL, Facundo. Sistemas de recirculación y tratamiento de agua. Santa Ana-Corrientes, Argentina, 2007. 37 p.

GUERRERO GOMEZ, Lorena y SANDOVAL, Felkin Eduardo. Colombia: Los recursos hídricos y el marco legal. Bogotá, 2006. 11 p.

Hmi Controller for Arduino. [Artículo de internet] http://hmicontroller.weebly.com/ [Consulta: Octubre 20 de 2015].

INSTITUTO COLOMBIANO DE NORMAS TECNICAS. Requisitos para materias primas, componentes e insumos. Primera actualización Bogotá. INCONTEC, 2008. NTC 2050.

Laboratorio de incubación artificial de Omegafish. [Artículo de internet] http://omegafish.com.co/index.php/nuestra-empresa/instalaciones [Consulta: Octubre 18 de 2015]

Laboratory Animal Housing (Aquatic-habitats). [Artículo de internet] http://pentairaes.com/laboratory-animal-housing-aquatic-habitats [Consulta: Octubre 15 de 2015]

LM35 Precision Centigrade Temperature Sensors. [Artículo de internet] http://www.ti.com/lit/ds/symlink/lm35.pdf [Consulta: Octubre 20 de 2015].

Miranda – Monitoring & Control System. [Artículo de internet] http://innovaqua.com/fichas_es/espana/miranda.html [Consulta: Octubre 15 de 2015]

Módulo Bluetooth HC-05. [Artículo de internet] http://www.ardobot.com/aprende/bluetooth-hc-05 : [Octubre 20 de 2015].

Monitoriza - Manual de usuario v. 3.2. [Artículo de internet] http://www.acimut.es/monitoriza/Manual%20Monitoriza.pdf [Consulta: Octubre 25 de 2015].

PRIETO, Camilo. Incubación artificial de huevos embrionados de tilapia roja. Medellín: Universidad de Antioquia, 2001. 6 p.

Probes. [Artículo de internet] http://atlas-scientific.com/probes.html [Consulta: Octubre 20 de 2015].

Product User's Manual – HC-SR04 Ultrasonic Sensor. [Artículo de internet] https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit?pref=2&pli=1# [Consulta: Octubre 20 de 2015].

Protcolo de Comunicación Modbus. [Artículo de internet] http://www.ecured.cu/Protocolo_de_Comunicaci%C3%B3n_Modbus. [Consulta: Octubre 14 de 2015]

REGLAMENTO TECNICO DE INSTALACIONES ELÉCTRICAS. Requisitos cables de control. Bogotá. RETIE, 2010, Articulo 17.

RIVERA, Diana Isabel y YÉPEZ, Eddy Antonio. Diseño e implementación de un prototipo para la medición de calidad del agua y control de la oxigenación en forma remota orientado a la producción acuícola. Tesis de Grado Ingeniero de Sistemas e Ingeniero Electrónico. Guayaquil: Universidad Politécnica Salesiana. 2015. 91 p.

SALAZAR TORRES, Roger. Modbus RTU. Implementación del protocolo en microcontrolador. Trabajo de Grado Ingeniero Electrónico. Bucaramanga: Universidad Industrial de Santander. 2006. 22-29 p.

Sistemas UNI Recirculación. [Artículo de internet] http:// www.akvagroup.com [Consulta: Octubre 12 de 2015]

SQL Server Editions. [Artículo de internet] https://www.microsoft.com/en-us/server-cloud/Products/sql-server-editions/sql-server-express.aspx [Consulta: Noviembre 14 de 2015].

SSR G4OAC5. [Artículo de internet] http://www.opto22.com/site-/pr_details.aspx?cid=4&item=G4OAC5A [Consulta: Octubre 30 de 2015].

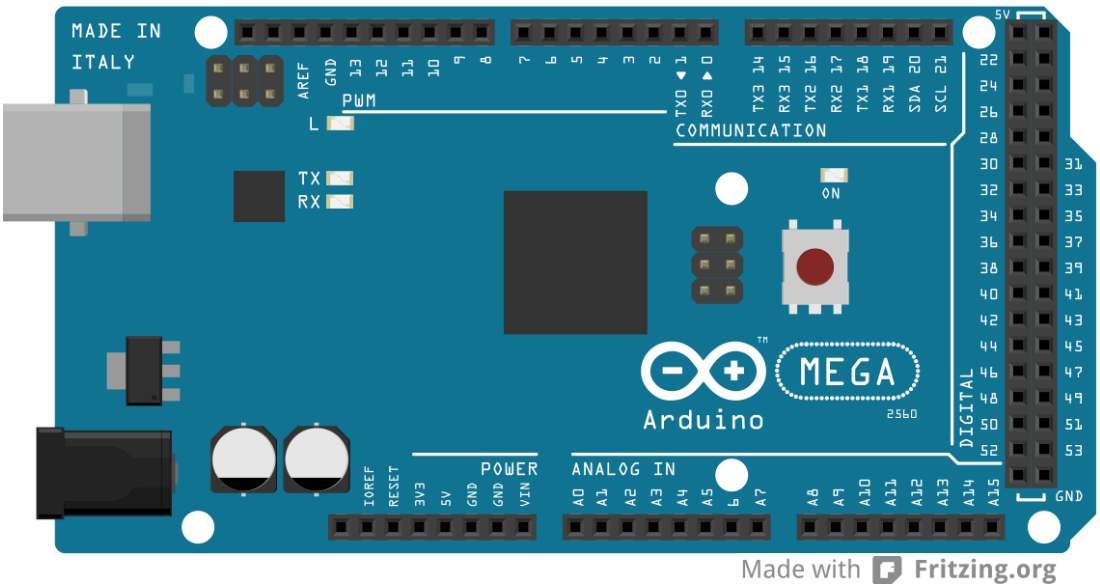
TIMMONS, Ebeling y VINCI, B.J. Sistemas de recirculación para la acuicultura. Santiago de Chile. Editado por Fundación Chile. 2002, 207-258; 278-279 p.

Anexo A

Placa Arduino Mega 2560

Es una versión ampliada de la tarjeta original de Arduino y está basada en el microcontrolador Atmega2560. Ver figura 95.

Figura 95. Arduino Mega 2560



Fuente: ARDUINO

Dispone de 54 entradas/salidas digitales, 14 de las cuales se pueden utilizar como salidas PWM (modulación de anchura de pulso). Además dispone de 16 entradas analogicas, 4 UARTs (puertas series), un oscilador de 16MHz, una conexión USB, un conector de alimentación, un conector ICSP y un pulsador para el reset. Para empezar a utilizar la placa sólo es necesario conectarla al ordenador a traves de un cable USB , o alimentarla con un adaptador de corriente AC/DC. También, para empezar, puede alimentarsemediante una batería.

Una de las diferencias principales de la tarjeta Arduino MEGA 2560 es que no utiliza el convertidor USB-serie de la firma FTDI. Por lo contrario, emplea un microcontrolador Atmega8U2 programado como actuar convertidor USB a serie.

Esta placa debido a su gran poder es utilizada para grandes proyectos, entre los mas importantes se encuentras los de DOMOTICA y IMPRESORAS 3D

El Arduino MEGA2560 es compatible con la mayoria de los shield o tarjetas de aplicación/ampliación disponibles para las tarjetas Arduino UNO original.

Las características principales son:

- Microprocesador ATmega2560
- Tensión de alimentación (recomendado) 7-12V
- Integra regulación y estabilización de +5Vcc
- 54 líneas de Entradas/Salidas Digitales (14 de ellas se pueden utiliza como salidas PWM)
- 16 Entradas Analógicas
- Máxima corriente continua para las entradas: 40 mA
- Salida de alimentación a 3.3V con 50 mA
- Memoria de programa de 256Kb (el bootloader ocupa 8Kb)
- Memoria SRAM de 8Kb para datos y variables del programa
- Memoria EEPROM para datos y variables no volátiles
- Velocidad del reloj de trabajo de 16MHz
- Reducidas dimensiones de 100 x 50 mm

Alimentación

El Arduino Mega puede ser alimentado vía la conexión USB o con una fuente de alimentación externa. El origen de la alimentación se selecciona automáticamente.

Las fuentes de alimentación externas (no-USB) pueden ser tanto un transformador o una batería. El transformador se puede conectar usando un conector macho de 2.1mm con centro positivo en el conector hembra de la placa. Los cables de la

batería puede conectarse a los pines Gnd y Vin en los conectores de alimentación (POWER)

La placa puede trabajar con una alimentación externa de entre 6 a 20 voltios. Si el voltaje suministrado es inferior a 7V, el pin de 5V puede proporcionar menos de 5 Voltios y la placa puede volverse inestable; si se usan mas de 12V los reguladores de voltaje se pueden sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son los siguientes:

- VIN. La entrada de voltaje a la placa Arduino cando se está usando una fuente externa de alimentación (en opuesto a los 5 voltios de la conexión USB). Se puede proporcionar voltaje a través de este pin, o, si se está alimentando a través de la conexión de 2.1mm , acceder a ella a través de este pin.

- 5V. La fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa. Esta puede provenir de VIN a través de un regulador integrado en la placa, o proporcionada directamente por el USB u otra fuente estabilizada de 5V.

- 3V3. Una fuente de voltaje de 3.3 voltios generada por un regulador integrado en la placa. La corriente máxima soportada 50mA.

- GND. Pines de toma de tierra.

Memoria

El ATmega2560 tiene 256KB de memoria flash para almacenar código (8KB son usados para el arranque del sistema(bootloader). El ATmega2560 tiene 8 KB de memoria SRAM y 4KB de EEPROM, a la cual se puede acceder para leer o escribir con la librería EEPROM.

Entradas y Salidas

Cada uno de los 54 pines digitales en el Mega pueden utilizarse como entradas o como salidas usando las funciones pinMode(), digitalWrite(), y digitalRead(). Las E/S operan a 5 voltios. Cada pin puede proporcionar o recibir una intensidad

maxima de 40mA y tiene una resistencia interna de pull-up (desconectada por defecto) de 20-50kOhms. Además, algunos pines tienen funciones especializadas:

- Serie: 0 (RX) y 1 (TX),Serie 1: 19 (RX) y 18 (TX); Serie 2: 17 (RX) y 16 (TX); Serie 3: 15 (RX) y 14 (TX).Usados para recibir (RX) transmitir (TX) datos a través de puerto serie TTL. Los pines Serie: 0 (RX) y 1 (TX) estan conectados a los pines correspondientes del chip FTDI USB-to-TTL.

- Interrupciones Externas: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3), y 21 (interrupción 2). Estos pines se pueden configurar para lanzar una interrupción en un valor LOW(0V), en flancos de subida o bajada (cambio de LOW a HIGH(5V) o viceversa), o en cambios de valor. Ver la función attachInterrupt() para más detalles.

- PWM: de 0 a 13.Proporciona una salida PWM (Pulse Wave Modulation, modulación de onda por pulsos) de 8 bits de resolución (valores de 0 a 255) a traves de la función analogWrite().

- SPI: 50 (SS), 51 (MOSI), 52 (MISO), 53 (SCK). Estos pines proporcionan comunicación SPI, usando lalibrería SPI.

- LED: 13.Hay un LED integrado en la placa conectado al pin digital 13, cuando este pin tiene un valor HIGH(5V) el LED se enciende y cuando este tiene un valor LOW(0V) este se apaga.

El Mega tiene 16 entradas analógicas, y cada una de ellas proporciona una resolución de 10bits (1024 valores). Por defecto se mide desde 0V a 5V, aunque es posible cambiar la cota superior de este rango usando el pin AREF y la función analogReference().

- I2C: 20 (SDA) y 21 (SCL). Soporte para el protocolo de comunicaciones I2C (TWI) usando la librería Wire.

- AREF. Voltaje de referencia para la entradas analogicas. Usado por analogReference().

- Reset. Suministrar un valor LOW (0V) para reiniciar el microcontrolador. Típicamente usado para añadir un botón de reset a los shields que no dejan acceso a este botón en la placa.

Comunicaciones

EL Arduino Mega facilita en varios aspectos la comunicación con la PC, otro Arduino u otros microcontroladores. El ATmega2560proporciona cuatro puertos de comunicación vía serie UART TTL (5V). Un ATmega16U2 integrado en la placa canaliza esta comunicación serie a través del puerto USB y los drivers (incluidos en el software de Arduino) proporcionan un puerto serie virtual en el ordenador. El software incluye un monitor de puerto serie que permite enviar y recibir información textual de la placa Arduino. Los LEDS RX y TX de la placa parpadearán cuando se detecte comunicación transmitida través de la conexión USB (no parpadearán si se usa la comunicación serie a través de los pines 0 y 1).

La libreria SoftwareSerial permite comunicación serie por cualquier par de pines digitales del Mega.

El ATmega2560 también soporta la comunicación I2C (TWI) y SPI. El software de Arduino incluye una librería Wire para simplificar el uso el bus I2C. Para el uso de la comunicación SPI, ver la hoja de especificaciones (datasheet) del ATmega2560.

Programación

El Arduino Mega se puede programar con el software Arduino. El ATmega2560 en el Arduino Mega viene precargado con un gestor de arranque (bootloader) que permite cargar nuevo código sin necesidad de un programador por hardware externo. Se comunica utilizando el protocolo STK500 original (referencia, archivo de cabecera C).

También puede evitarse el gestor de arranque y programar directamente el microcontrolador a través del puerto ICSP (In Circuit Serial Programming)

Reinicio Automatico por Software

En vez de necesitar reiniciar presionando físicamente el botón de reset antes de cargar, el Arduino Mega esta diseñado de manera que es posible reiniciar por software desde el ordenador donde este conectado. Una de las lineas de control de flujo (DTR) del ATmega16U2 está conectada a la linea de reinicio del ATmega2560 a través de un condensador de 100 nanofaradios. Cuando la linea se pone a LOW(0V), la linea de reinicio también se pone a LOW el tiempo suficiente para reiniciar el chip. El software de Arduino utiliza esta característica para permitir cargar los sketches con solo apretar un botón del entorno. Dado que el gestor de arranque tiene un lapso de tiempo para ello, la activación del DTR y la carga del sketch se coordinan perfectamente.

Esta configuración tiene otras implicaciones. Cuando el Mega se conecta a un ordenador con Mac OS X o Linux, esto reinicia la placa cada vez que se realiza una conexión desde el software (via USB). El medio segundo aproximadamente posterior, el gestor de arranque se esta ejecutando. A pesar de estar programado para ignorar datos mal formateados (ej. cualquier cosa que la carga de un programa nuevo) intercepta los primeros bytes que se envían a la placa justo después de que se abra la conexión. Si un sketch ejecutándose en la placa recibe algún tipo de configuración inicial o otro tipo de información al inicio del programa, debe asegurarse de que el software con el cual se comunica espera un segundo después de abrir la conexión antes de enviar los datos.

El Mega contiene una pista que puede ser cortada para deshabilitar el auto-reset. Las terminaciones a cada lado pueden ser soldadas entre ellas para rehabilitarlo. Están etiquetadas con “RESET-EN”. También se puede deshabilitar el auto-reset conectando una resistencia de 110 ohms desde el pin 5V al pin de reset.

Protección contra sobrecorrientes en USB

El Arduino Mega tiene un multifusible reinicializable que protege la conexión USB del PC de cortocircuitos y sobretensiones. Aparte de que la mayoría de ordenadores proporcionan su propia protección interna, el fusible proporciona un capa extra de protección. Si más de 500mA son detectados en el puerto USB, el fusible automáticamente corta la conexión hasta que el cortocircuito o la sobretensión desaparece.

Características Físicas y Compatibilidad de Shields

La longitud y amplitud máxima de la placa Mega 2560 son de 4 y 2.1 pulgadas respectivamente, con el conector USB y la conexión de alimentación sobresaliendo de estas dimensiones. Tres agujeros para fijación con tornillos permiten colocar la placa en superficies y cajas. Tener en cuenta que la distancia entre los pines digitales 7 y 8 es 160 mil (0,16″), no es múltiplo de la separación de 100 mil entre los otros pines.

Estructura de un programa para Arduino

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes necesarias, o funciones, encierran bloques que contienen declaraciones, estamentos o instrucciones.

```
void setup() //Primera Parte

{

  estamentos;

}

void loop() //Segunda Parte

{

  estamentos;

}
```

En donde setup() es la parte encargada de recoger la configuración y loop() es la que contiene el programa que se ejecutará cíclicamente (de ahí el término loop –bucle-). Ambas funciones son necesarias para que el programa trabaje.

La función de configuración (setup) debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta sólo una vez, y se utiliza para configurar o inicializar pinMode (modo de trabajo de las E/S), configuración de la comunicación en serie y otras.

La función bucle (loop) siguiente contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc) Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

setup()

La función setup() se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pins, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar. Así mismo se puede utilizar para establecer el estado inicial de las salidas de la placa.

```
void setup()

{

  pinMode(pin, OUTPUT); // configura el 'pin' como salida

  digitalWrite(pin, HIGH);  // pone el ‘pin’ en estado HIGH
```

```
}
```

loop()

Después de llamar a setup(), la función loop() hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa esté respondiendo continuamente ante los eventos que se produzcan en la placa.

```
void loop()

{

digitalWrite(pin, HIGH); // pone en uno (on, 5v) el 'pin'

delay(1000);              // espera un segundo (1000 ms)

digitalWrite(pin, LOW); // pone en cero (off, 0v.) el 'pin'

delay(1000);

}
```

Funciones

Una función es un bloque de código que tiene un nombre y un conjunto de instrucciones que son ejecutadas cuando se llama a la función. Son funciones setup() y loop() de las que ya se ha hablado. Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Las funciones se declaran asociadas a un tipo de valor “type”. Este valor será el que devolverá la función, por ejemplo 'int' se utilizará cuando la función devuelve un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocará delante la palabra “void”, que significa “función vacía”. Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

```
type nombreFunción(parámetros)

{

instrucción;

}
```

La función siguiente devuelve un número entero, delayVal() se utiliza para poner un valor de retraso en un programa que lee una variable analógica de un potenciómetro conectado a una entrada de Arduino. Al principio se declara como una variable local, 'v' recoge el valor leído del potenciómetro que estará comprendido entre 0 y 1023, luego se divide el valor por 4 para ajustarlo a un margen comprendido entre 0 y 255, finalmente se devuelve el valor 'v' y se retornaría al programa principal. Esta función cuando se ejecuta devuelve el valor de tipo entero 'v'.

```
int delayVal()

{

int v;                    // crea una variable temporal 'v'

v= analogRead(pot);      // lee el valor del potenciómetro

v /= 4;                 // convierte 0-1023 a 0-255

return v;                // devuelve el valor final

}
```

{} entre llaves

Las llaves sirven para definir el principio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), if.., etc.

```
type funcion()

{

instrucciones;

}
```

Una llave de apertura “{“ siempre debe ir seguida de una llave de cierre “}”, si no es así el programa dará errores.

El entorno de programación de Arduino incluye una herramienta de gran utilidad para comprobar el total de llaves. Sólo tienes que hacer click en el punto de inserción de una llave abierta e inmediatamente se marca el correspondiente cierre de ese bloque (llave cerrada).

; punto y coma

El punto y coma “;” se utiliza para separar instrucciones en el lenguaje de programación de Arduino. También se utiliza para separar elementos en una instrucción de tipo “bucle for”.

```
int x = 13;   /* declara la variable 'x' como tipo entero de valor 13 */
```

Nota: No poner fin a una línea con un punto y coma o se producirá en un error de compilación. El texto de error puede ser obvio, y se referirá a la falta de una coma, o puede que no. Si se produce un error raro y de difícil detección lo primero que debemos hacer es comprobar que los puntos y comas están colocados al final de las instrucciones.

/*… */ bloque de comentarios

Los bloques de comentarios, o comentarios multi-línea son áreas de texto ignorados por el programa que se utilizan para las descripciones del código o comentarios que ayudan a comprender el programa. Comienzan con / * y terminan con * / y pueden abarcar varias líneas.

```
/* esto es un bloque de comentario no se debe olvidar

cerrar los comentarios estos deben estar equilibrados */
```

Debido a que los comentarios son ignorados por el compilador y no ocupan espacio en la memoria de Arduino pueden ser utilizados con generosidad. También pueden utilizarse para "comentar" bloques de código con el propósito de anotar informaciones para depuración y hacerlo mas comprensible para cualquiera.

Nota: Dentro de una misma línea de un bloque de comentarios NO se puede escribir otro bloque de comentarios (usando /*..*/).

// línea de comentarios

Una línea de comentario empieza con // y terminan con la siguiente línea de código. Al igual que los comentarios de bloque, los de línea son ignoradas por el programa y no ocupan espacio en la memoria.

```
// esto es un comentario
```

Una línea de comentario se utiliza a menudo después de una instrucción, para proporcionar más información acerca de lo que hace ésta o para recordarla más adelante.

Anexo B

IMPLEMENTACION DE MODBUS RTU EN ARDUINO

```
/*****************************************************************************
 * EMPEZAR FUNCIONES ESCLAVO MODBUS RTU
 *****************************************************************************/

/* Variables globales */
unsigned int Txenpin = 0; /* Habilitar pin de transmisión, que se utiliza en las redes
SERIALES*/


/*  Enumeración de códigos de función Modbus compatibles. Si implementa una nueva, colocar
código de función aquí! */
enum {
        FC_READ_REGS  = 0x03,   //Lectura de multiples registros de almacenamiento - FUNCION
03
        FC_WRITE_REG  = 0x06,   //Escritura de un simple registro - FUNCION 06
        FC_WRITE_REGS = 0x10,   //Escritura de multiples registros - FUNCION 16
        FC_READ_DEV_ID = 0x2B   //Lectura de la informacion del dispositivo - FUNCION 43
};

/*   Funciones soportadas. Si se implementa una nueva, colocar código de función en este
conjunto! */
const   unsigned   char   fsupported[]   =   {   FC_READ_REGS,   FC_WRITE_REG,   FC_WRITE_REGS,
FC_READ_DEV_ID };

/* Constantes */
enum {
        MAX_READ_REGS = 0x7D, //125
        MAX_WRITE_REGS = 0x7B, //123
        MAX_MESSAGE_LENGTH = 256
};


enum {
        RESPONSE_SIZE = 6,
        EXCEPTION_SIZE = 3,
        CHECKSUM_SIZE = 2
};

/* Código de excepciones */
enum {
        NO_REPLY = -1,
        EXC_FUNC_CODE = 1,
        EXC_ADDR_RANGE = 2,
        EXC_REGS_QUANT = 3,
        EXC_EXECUTE = 4
};

/* Posiciones dentro de la consulta / respuesta matriz */
enum {
        SLAVE = 0,
        FUNC,
        START_H,
        START_L,
        REGS_H,
        REGS_L,
        BYTE_CNT
};
/*
```

```
CRC

 ENTRADAS:
   buf   ->  Array que contiene el mensaje que se enviará al controlador.
  start ->   Inicio del contador bucle de crc, generalmente 0.
  cnt   ->  Cantidad de bytes en el mensaje que se envía al controlador
 SALIDAS:
  temp  ->  Devuelve el byte CRC para el mensaje.
 COMENTARIOS:
        Esta rutina calcula el byte alto y bajo del CRC de un mensaje.
  Tenga en cuenta que este CRC sólo se utiliza para Modbus, no Modbus + etc.
 ************************************************************************/

unsigned int crc(unsigned char *buf, unsigned char start,
unsigned char cnt)
{
        unsigned char i, j;
        unsigned temp, temp2, flag;

        temp = 0xFFFF;

        for (i = start; i < cnt; i++) {
                temp = temp ^ buf[i];

                for (j = 1; j <= 8; j++) {
                        flag = temp & 0x0001;
                        temp = temp >> 1;
                        if (flag)
                                temp = temp ^ 0xA001;
                }
        }

        /* Invertir orden del byte. */
        temp2 = temp >> 8;
        temp = (temp << 8) | temp2;
        temp &= 0xFFFF;

        return (temp);
}


/************************************************************************
 *
 *  Las funciones siguientes construyen la consulta requerida en
 *      un paquete de consulta Modbus.
 *
 ************************************************************************/

/*
 * Inicio del paquete de una respuesta read_holding_register - FUNCION 03
 */
void build_read_packet(unsigned char slave, unsigned char function,
unsigned char count, unsigned char *packet)
{
        packet[SLAVE] = slave;
        packet[FUNC] = function;
        packet[2] = count * 2;
}
/*
 *  Inicio del paquete de una respuesta preset_multiple_register - FUNCION 16
 */
void build_write_packet(unsigned char slave, unsigned char function,
unsigned int start_addr,
unsigned char count,
unsigned char *packet)
{
        packet[SLAVE] = slave;
        packet[FUNC] = function;
```

```
        packet[START_H] = start_addr >> 8;
        packet[START_L] = start_addr & 0x00ff;
        packet[REGS_H] = 0x00;
        packet[REGS_L] = count;
}

/*
 *  Inicio del paquete de una respuesta write_single_register - FUNCION 06
 */
void build_write_single_packet(unsigned char slave, unsigned char function,
        unsigned int write_addr, unsigned int reg_val, unsigned char* packet)
{
        packet[SLAVE] = slave;
        packet[FUNC] = function;
        packet[START_H] = write_addr >> 8;
        packet[START_L] = write_addr & 0x00ff;
        packet[REGS_H] = reg_val >> 8;
        packet[REGS_L] = reg_val & 0x00ff;
}

/*
 * Inicio del paquete de una respuesta de excepción
 */
void build_error_packet(unsigned char slave, unsigned char function,
unsigned char exception, unsigned char *packet)
{
        packet[SLAVE] = slave;
        packet[FUNC] = function + 0x80;
        packet[2] = exception;
}


/*************************************************************************
 *
 * * Modbus_query (paquete, longitud)
 *
 * Función para añadir una suma de comprobación para el final de un paquete.
 * Tener en cuenta que el conjunto de paquetes debe ser de al menos 2 campos
   más que el string_length.
 *************************************************************************/

void modbus_reply(unsigned char *packet, unsigned char string_length)
{
        int temp_crc;

        temp_crc = crc(packet, 0, string_length);
        packet[string_length] = temp_crc >> 8;
        string_length++;
        packet[string_length] = temp_crc & 0x00FF;
}

/*************************************************************************
 *
 * send_reply( query_string, query_length )
 *
 *  Función para enviar una respuesta a un maestro Modbus.
 *  Devuelve: número total de caracteres enviado
 *************************************************************************/

int send_reply(unsigned char *query, unsigned char string_length)
{
        unsigned char i;

        if (Txenpin > 1) { // Seleccionar comunicacion serial en modo escucha
                UCSR0A=UCSR0A |(1 << TXC0);
                digitalWrite( Txenpin, HIGH);
                delay(1);
        }
```

```
        modbus_reply(query, string_length);
        string_length += 2;

        for (i = 0; i < string_length; i++) {
                Serial.write(query[i]);
        }

        if (Txenpin > 1) {      // Seleccionar comunicacion serial en modo escucha
                while (!(UCSR0A & (1 << TXC0)));
                digitalWrite( Txenpin, LOW);
        }

        return i;     /* Esto no significa que la escritura tuvo éxito */
}

/*************************************************************************
 *
 *  receive_request( array_for_data )
 *
 * Función para vigilar a una petición del maestro Modbus.
 *
 * Devuelve: Número total de caracteres recibidos, si OK.
 *           0 si no hay ninguna solicitud.
 *           Un código de error negativo en caso de fallo.
 *************************************************************************/

int receive_request(unsigned char *received_string)
{
        int bytes_received = 0;

        /* FIXME: hace que Serial.available espere 1.5T o 3.5T antes de salir del bucle */
        while (Serial.available()) {
                received_string[bytes_received] = Serial.read();
                bytes_received++;
                if (bytes_received >= MAX_MESSAGE_LENGTH)
                        return NO_REPLY;  /*  Error de puerto  */
        }

        return (bytes_received);
}

/*************************************************************************
 *
 *  modbus_request(slave_id, request_data_array)
 *
 * Funcion para la solicitud correcta devuelta y que la suma de comprobación
 * es correcta.
 *
 * Devuelve:  string_length si OK.
 *    0 si falla.
 *    Menor que 0 para errores de excepción.
 *
 *   Nota: Todas las funciones que se utilizan para enviar o recibir datos a través de
 *             Modbus devuelven estos valores de retorno.
 *
 *************************************************************************/

int modbus_request(unsigned char slave, unsigned char *data)
{
        int response_length;
        unsigned int crc_calc = 0;
        unsigned int crc_received = 0;
        unsigned char recv_crc_hi;
        unsigned char recv_crc_lo;

        response_length = receive_request(data);

        if (response_length > 0) {
```

```
                crc_calc = crc(data, 0, response_length - 2);
                recv_crc_hi = (unsigned) data[response_length - 2];
                recv_crc_lo = (unsigned) data[response_length - 1];
                crc_received = data[response_length - 2];
                crc_received = (unsigned) crc_received << 8;
                crc_received =
                        crc_received | (unsigned) data[response_length - 1];

                /*********** Comprobar CRC de la respuesta  ************/

                if (crc_calc != crc_received) {
                        return NO_REPLY;
                }

                /* Comprobar Identificación del esclavo */

                if (slave != data[SLAVE]) {
                        return NO_REPLY;
                }
        }
        return (response_length);
}

/*********************************************************************
 *
 * validate_request(request_data_array, request_length, available_regs)
 *
 * Función para comprobar que la solicitud pueda ser procesada por el esclavo.
 *
 * Devuelve: 0 si es OK
 *           Un código de excepción negativo en caso de error
 *
 *********************************************************************/

int validate_request(unsigned char *data, unsigned char length,
unsigned int regs_size)
{
        int i, fcnt = 0;
        unsigned int regs_num = 0;
        unsigned int start_addr = 0;
        unsigned char max_regs_num;

        /* Chequear código de función */

        for (i = 0; i < sizeof(fsupported); i++) {
                if (fsupported[i] == data[FUNC]) {
                        fcnt = 1;
                        break;
                }
        }
        if (0 == fcnt)
                return EXC_FUNC_CODE;

        if (FC_WRITE_REG == data[FUNC]) {
                /*  Para la función de escribir un solo registro, este es el registro de
destino. */
                regs_num = ((int) data[START_H] << 8) + (int) data[START_L];
                if (regs_num >= regs_size)
                        return EXC_ADDR_RANGE;
                return 0;
        }

        if (FC_READ_DEV_ID == data[FUNC]) {
                                                /* Para función de información de dispositivo.
*/
                if (data[3] != 1)             /* Sólo identificación del dispositivo básico
*/
                  return EXC_REGS_QUANT;
```

```
                return 0;
        }

            /* Para las funciones de registro de lectura / escritura, este es el rango */

        regs_num = ((int) data[REGS_H] << 8) + (int) data[REGS_L];

        /* Comprobar la cantidad de registros  */

        if (FC_READ_REGS == data[FUNC])
                max_regs_num = MAX_READ_REGS;
        else if (FC_WRITE_REGS == data[FUNC])
                max_regs_num = MAX_WRITE_REGS;

        if ((regs_num < 1) || (regs_num > max_regs_num))
                return EXC_REGS_QUANT;

        /* Comprobar rango de registro, direccion de inicio es 0 */

        start_addr = ((int) data[START_H] << 8) + (int) data[START_L];
        if ((start_addr + regs_num) > regs_size)
                return EXC_ADDR_RANGE;

        return 0;     /* OK, no es una excepción */
}

/**************************************************************************
 *
 *  write_regs(first_register, data_array, registers_array)
 *
 *  escribe dentro del holding registers del esclavo los datos de la consulta,
 *      a partir de las start_addr.
 *
 *      Devuelve: el número de registros escritos
 **************************************************************************/

int write_regs(unsigned int start_addr, unsigned char *query, int *regs)
{
        int temp;
        unsigned int i;

        for (i = 0; i < query[REGS_L]; i++) {
                /* desplazar el registro hi_byte a temp */
                temp = (int) query[(BYTE_CNT + 1) + i * 2] << 8;
                /* OR con lo_byte */
                temp = temp | (int) query[(BYTE_CNT + 2) + i * 2];

                regs[start_addr + i] = temp;
        }
        return i;
}

/**************************************************************************
 *
 *  preset_multiple_registers(slave_id, first_register, number_of_registers,
 *      data_array, registers_array)
 *
 *  Escriba los datos de una matriz en el holding registers del esclavo.
 *
 **************************************************************************/

int preset_multiple_registers(unsigned char slave,
unsigned int start_addr,
unsigned char count,
unsigned char *query,
int *regs)
{
        unsigned char function = FC_WRITE_REGS; /* Preseleccionar Multiple Registros */
```

```
        int status = 0;
        unsigned char packet[RESPONSE_SIZE + CHECKSUM_SIZE];

        build_write_packet(slave, function, start_addr, count, packet);

        if (write_regs(start_addr, query, regs)) {
                status = send_reply(packet, RESPONSE_SIZE);
        }

        return (status);
}


/**********************************************************************
 *
 * write_single_register(slave_id, write_addr, data_array, registers_array)
 *
 * Escribe un único int val en un single holding register del esclavo.
 *
 **********************************************************************/

int write_single_register(unsigned char slave,
        unsigned int write_addr, unsigned char *query, int *regs)
{
        unsigned char function = FC_WRITE_REG; /*  Función: Escribe Registro Único */
        int status = 0;
        unsigned int reg_val;
        unsigned char packet[RESPONSE_SIZE + CHECKSUM_SIZE];

        reg_val = query[REGS_H] << 8 | query[REGS_L];
        build_write_single_packet(slave, function, write_addr, reg_val, packet);
        regs[write_addr] = (int) reg_val;
/*
        written.start_addr=write_addr;
        written.num_regs=1;
*/
        status = send_reply(packet, RESPONSE_SIZE);

        return (status);
}


/**********************************************************************
 *
 *  read_holding_registers(slave_id, first_register, number_of_registers,
 *      registers_array)
 *
 *      lee los holdings registers del esclavo y los envía al maestro Modbus.
 *
 **********************************************************************/

int read_holding_registers(unsigned char slave, unsigned int start_addr,

unsigned char reg_count, int *regs)
{
        unsigned char function = 0x03;  /* Función 03: Lectura de registros  */
        int packet_size = 3;
        int status;
        unsigned int i;
        unsigned char packet[MAX_MESSAGE_LENGTH];

        build_read_packet(slave, function, reg_count, packet);

        for (i = start_addr; i < (start_addr + (unsigned int) reg_count);
         i++) {
                        packet[packet_size] = regs[i] >> 8;
                packet_size++;
                packet[packet_size] = regs[i] & 0x00FF;
```

```
                packet_size++;
        }


        status = send_reply(packet, packet_size);

        return (status);
}

/*************************************************************************
 *
 *     read_device_id(slave_id, query , length)
 *
 *     Lee la identificación del dispositivo y lo envía al maestro Modbus
 *
 *************************************************************************/

int read_device_id(unsigned char slave, unsigned char *query, unsigned char lenght)
{
        unsigned char function = 0x2B;  /* Funcion 43: Leer identificacion del dispositivo
*/
        unsigned char MEIType = 0x0E;   /* MEI Type */
        unsigned char ReadDeviceIDCode = 0x01; /*   Leer Código de Identificación del
dispositivo  */
        unsigned char ConformityLevel = 0x01; /* Nivel de Conformidad */
        unsigned char MoreFollows = 0x0; /* lo que sigue */
        unsigned char NextObjectID = 0x0; /* Siguiente ID del objeto */
        unsigned char NumObjects = 0x03; /* Número de objetos  */
        int packet_size = 3;
        int status;
        unsigned int i;
        unsigned char packet[MAX_MESSAGE_LENGTH];

        packet[SLAVE] = slave;
        packet[FUNC] = function;
        packet[2] = MEIType;
        packet[3] = ReadDeviceIDCode;
        packet[4] = ConformityLevel;
        packet[5] = MoreFollows;
        packet[6] = NextObjectID;

        unsigned int StartObject = 7;
        packet[StartObject] = NumObjects;

        /* ID del objeto 0 */

        String company ="ARDUINO";
        int length=company.length();
        packet[StartObject+1] = 0;       /* ID del objeto = 0 */
        packet[StartObject+2] = length;
        for (i = 0; i < length; i++ ) {
          packet[StartObject + 3 + i] = company.charAt(i);
        }
        StartObject = StartObject + 2 + length;

        /*  ID del objeto 1 */

        String product ="MEGA";
        length=product.length();
        packet[StartObject+1] = 1;      /* ID del objeto = 1 */
        packet[StartObject+2] = length;
        for (i = 0; i < length; i++ ) {
          packet[StartObject + 3 + i] = product.charAt(i);
        }
        StartObject = StartObject + 2 + length;

        /*  ID del objeto 2 */
```

```
        String ver ="V1";
        length=ver.length();
        packet[StartObject+1] = 2;  /* ID del objeto = 2 */
        packet[StartObject+2] = length;
        for (i = 0; i < length; i++ ) {
          packet[StartObject + 3 + i] = ver.charAt(i);
        }
        StartObject = StartObject + 2 + length + 1;

        status = send_reply(packet, StartObject);

        return (status);
}


void configure_mb_slave(long baud, char parity, char txenpin)
{
        Serial.begin(baud);

        switch (parity) {
        case 'e': // 8E1 even parity
                UCSR0C |= ((1<<UPM01) | (1<<UCSZ01) | (1<<UCSZ00));
                //      UCSR0C &= ~((1<<UPM00) | (1<<UCSZ02) | (1<<USBS0));
                break;
        case 'o': // 8O1 odd parity
                UCSR0C |= ((1<<UPM01) | (1<<UPM00) | (1<<UCSZ01) | (1<<UCSZ00));
                //      UCSR0C &= ~((1<<UCSZ02) | (1<<USBS0));
                break;
        case 'n': // 8N1 no parity
                UCSR0C |= ((1<<UCSZ01) | (1<<UCSZ00));
                //      UCSR0C &= ~((1<<UPM01) | (1<<UPM00) | (1<<UCSZ02) | (1<<USBS0));
                break;
        default:
                break;
        }

        if (txenpin > 1) {                // pin 0 y el pin 1 se reservan para RX / TX
                Txenpin = txenpin;       /* Establecer variable global */
                pinMode(Txenpin, OUTPUT);
                digitalWrite(Txenpin, LOW);
        }

        return;
}

/*
 * update_mb_slave(slave_id, holding_regs_array, number_of_regs)
 *
 * Comprueba si existe alguna solicitud válida del maestro Modbus. Si hay,
 * realiza la acción solicitada
 */

unsigned long Nowdt = 0;
unsigned int lastBytesReceived;
const unsigned long T35 = 5;

int update_mb_slave(unsigned char slave, int *regs,
unsigned int regs_size)
{
        unsigned char query[MAX_MESSAGE_LENGTH];
        unsigned char errpacket[EXCEPTION_SIZE + CHECKSUM_SIZE];
        unsigned int start_addr;
        int exception;
        int length = Serial.available();
        unsigned long now = millis();
        unsigned char MEIType = 0x0E;   /* MEI Type para identificación del dispositivo */

        if (length == 0) {
```

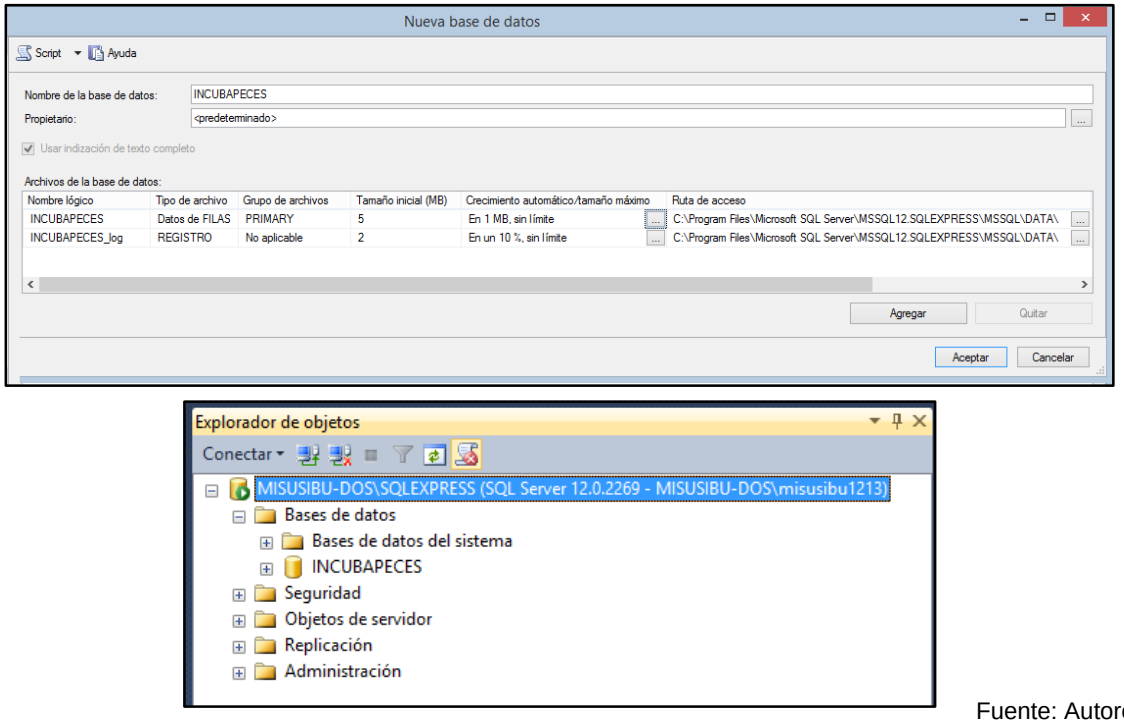
```
                lastBytesReceived = 0;
                return 0;
        }

        if (lastBytesReceived != length) {
                lastBytesReceived = length;
                Nowdt = now + T35;
                return 0;
        }
        if (now < Nowdt)
                return 0;

        lastBytesReceived = 0;

        length = modbus_request(slave, query);
        if (length < 1)
                return length;


        exception = validate_request(query, length, regs_size);
        if (exception) {
                        build_error_packet(slave, query[FUNC], exception,
                        errpacket);
                        send_reply(errpacket, EXCEPTION_SIZE);
                        return (exception);
        }


        start_addr = ((int) query[START_H] << 8) +
                      (int) query[START_L];
        switch (query[FUNC]) {
                case FC_READ_REGS:
                        return read_holding_registers(slave,
                        start_addr,
                        query[REGS_L],
                        regs);
                break;
                case FC_WRITE_REGS:
                        return preset_multiple_registers(slave,
                        start_addr,
                        query[REGS_L],
                        query,
                        regs);
                break;
                case FC_WRITE_REG:
                        write_single_register(slave,
                        start_addr,
                        query,
                        regs);
                break;
                case FC_READ_DEV_ID:
                        if (query[START_H]==MEIType) {
                            return read_device_id(slave, query, length);
                        }
                break;
        }
}
```

Anexo C

CREACION BASE DE DATOS

En la figura 96 se visualiza la ventana donde se crea la base de datos para guardar la información desde el SCADA.

Figura 96. Creación de la bases de datos en SQLEXPRESS



Fuente: Autores

CODIGO TABLA GUARDAR VARIABLES

```
USE [INCUBAPECES]
GO

/****** Object:  Table [dbo].[Variables_Incubadora]    Script Date: 02/12/2015
11:41:14 p. m. ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
```

```
GO

CREATE TABLE [dbo].[Variables_Incubadora](
	[Fecha] [datetime] NULL,
	[Temperatura] [decimal](3, 1) NULL,
	[Nivel_Agua] [int] NULL,
	[pH] [float] NULL,
	[Oxigeno_Disuelto] [float] NULL
) ON [PRIMARY]

GO
```

CODIGO TABLA GUARDAR ALARMAS

```
USE [INCUBAPECES]
GO

/****** Object:  Table [dbo].[Alarmas]    Script Date: 03/12/2015 12:09:50 p. m. ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo]. [Alarmas] (
	[ID] [int] IDENTITY (1, 1) NOT NULL,
	[Servidor] [varchar] (50) NULL,
	[Grupo] [varchar] (50) NULL ,
	[Variable] [varchar] (50) NULL ,
	[TextoAlarma] [varchar] (250) NULL ,
	[Accion] [varchar] (50) NULL ,
	[Valor] [varchar] (50) NULL ,
	[Usuario] [varchar] (50) NULL ,
	[FechaHora] [datetime] NULL

) ON [PRIMARY]
GO
ALTER TABLE [Alarmas] WITH NOCHECK ADD
	CONSTRAINT [PK_Alarmas] PRIMARY KEY  CLUSTERED
	(
	[ID]
	)  ON [PRIMARY]
GO

CREATE  INDEX  [IX_Alarmas]  ON  [Alarmas]([Servidor],  [Grupo],  [Variable], [TextoAlarma]) ON [PRIMARY]
 GO
   CREATE  INDEX [IX_Alarmas_1] ON [Alarmas]([FechaHora]) ON [PRIMARY]
GO
```

Anexo D

CONFIGURACION MODULO BLUETOOTH

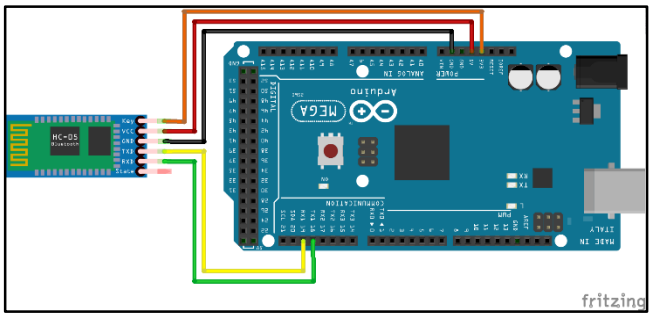
El módulo de bluetooth HC-05 es el que ofrece una mejor relación de precio y características, ya que es un módulo Maestro-Esclavo, quiere decir que además de recibir conexiones desde una PC o Tablet, también es capaz de generar conexiones hacia otros dispositivos bluetooth. Esto nos permite por ejemplo, conectar dos módulos de bluetooth y formar una conexión punto a punto para transmitir datos entre dos microcontroladores o dispositivos.

El HC-05 tiene un **modo de comandos AT** que debe activarse mediante un estado **alto en el PIN34 mientras se enciende (o se resetea) el módulo**. En las versiones para protoboard este pin viene marcado como **"Key"**. Una vez que estamos en el modo de comandos AT, podemos configurar el módulo bluetooth y cambiar parámetros como el nombre del dispositivo, password, modo maestro/esclavo, etc.

Para comunicarnos con el módulo y configurarlo, es necesario tener acceso al módulo mediante una interfaz serial. Podemos usar un arduino con un par de cables (aprovechando el puente USB-Serial del Arduino).

Las conexiones para realizar con arduino son bastante sencillas. Ver figura 97. Solamente requerimos colocar como mínimo la alimentación y conectar los pines de transmisión y recepción serial (TX y RX). Hay que recordar que en este caso los pines se debe conectar cruzados TX Bluetooth -> RX de Arduino y RX Bluetooth -> TX de Arduino.

Figura 97. Conexión módulo a Arduino Mega



Fuente: Autores

A continuación se carga el siguiente código al Arduino para realizar la configuración del módulo por comandos AT.

```
void setup()
{
Serial1.flush();
delay(500);
Serial1.begin(38400);
Serial.begin(9600);
Serial.println("Preparado para enviar comandos AT:");

Serial1.print("ATrn");

delay(100);

}

void loop()
{

if (Serial1.available())
Serial.write(Serial1.read());

if (Serial.available())
Serial1.write(Serial.read());

}
```

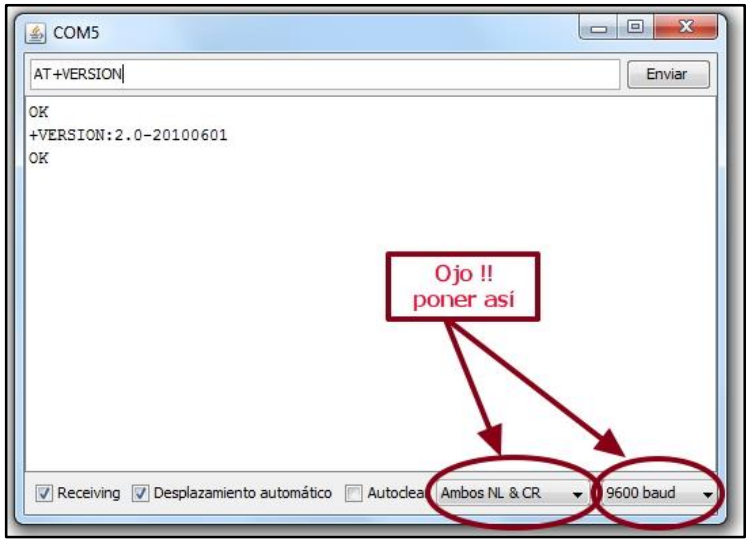
Ahora se arranca en modo master, es decir que admita códigos AT.

Solo se necesita:

1. Desconectar el pin GND o Vcc de alimentación del módulo bluetooth.
2. Apretar el botón y dejarlo pulsado.
3. Poner de nuevo el pin de alimentación del paso 1 **SIN** soltar el botón del paso 2.
4. Cuando se vea un parpadeo lento, unos 3 segundos. Se suelta el botón. Los parpadeos son mucho más lentos.

Se debe abrir el Terminal de Arduino IDE, seleccionar **9600 Baudios** como velocidad de transferencia y habilitar la opción "**Ambos NL & CR**". Seguido de esto se envía el comando **"AT**" automáticamente se debe recibir como respuesta "**OK**" si esto fue exitoso está todo bien. Ver figura 98.

Figura 98. Monitor serial Arduino para enviar códigos AT



Fuente: Autores

Ahora para saber a qué velocidad de transmisión está trabajando el **Bluetooth HC-05**, se escribe el comando **"AT+UART"** se debe recibir como respuesta "**+UART : 9600,0,0 / OK**". Si se desea cambiarla solo se ingresa: "**AT+UART = 115200,0,0**" Enter y se recibe como respuesta "**OK**", para verificarlo se repite el procedimiento anterior. Para el caso del proyecto de deja en 9600 bps.

Para cambiar el nombre al dispositivo se escribe el comando "**AT+NAME = "INCUBAPECES"**" y para verificarlo se vuelve a escribir el comando "**AT+NAME**".

Para cambiar la contraseña a 1234: **AT+PSWD:1234.**

Anexo E

CONTROLADOR HMI PARA ARDUINO

Los widgets

Se puede encontrar seis componentes diferentes (widgets) para interactuar con el Arduino, que se dividen en dos grupos. El primer grupo son los widgets de entrada que son conformados por:

1. Button: Se utiliza con las variables pin-out y virtual-booleana-in. En el cuadro de configuración de la aplicación, sólo se requiere una etiqueta con su nombre y la variable que va a ser asociado.

2. Switch: Se utiliza las mismas variables como el widget de botón y tener el mismo cuadro de configuración.

3. Slider: Cambia una variable de tipo int desde un valor mínimo a un valor máximo, actúa como un potenciómetro. Es compatible sólo con la variable virtual int-in. En el cuadro de configuración, que se requiere para poner la etiqueta con su nombre, el valor mínimo, el valor máximo y la variable a asociar.

El segundo grupo de widgets son los widgets de salida, estos son:

1. LED: Este widget de enciende o apaga según el estado de la variable asociada. Las variables a asociado son: pin-out, pin-in y virtual-booleana-out. Sólo se requiere la variable de asociación y una etiqueta con su nombre.

2. Display de 7 segmentos: Se muestran un número entero de 0 a 9999, la variable asociada es lo virtual-int-out. Sólo se requiere la variable de asociación y una etiqueta con su nombre.

3. Barra Indicadora: Este widget de muestra el estado de una variable desde un mínimo a un valor máximo, la variable asociada es lo virtual-int-out. En el cuadro de configuración, que se requiere para poner la etiqueta con su nombre, el valor mínimo, el valor máximo y la variable a asociar.

Nota: Los widgets de entrada envía datos a la placa Arduino y los widgets de salida lee datos de la placa Arduino.

Tipos de variables

El HMI Controller utiliza seis tipos diferentes de variables para la interacción entre la aplicación y la placa Arduino, estas variables son:

1. Pin-in: Esta variable representa un pin de entrada física de la placa Arduino. Esta variable se declara automáticamente cuando se utiliza la función de Arduino pinMode en el *sketch*.

2. Pin-out: Representa un pin de salida física de la placa Arduino. Esta variable se declara automáticamente cuando se utiliza la función de Arduino pinMode en el *sketch*.

3. Virtual-boolean-in: Es una variable booleana utilizada como un dato "IN" de la aplicación para el Arduino. Puede ser declarada en el sistema HMI mediante el uso de la función attachBooleanIn en el *sketch*.

4. Virtual boolean-out: Es una variable booleana utilizada como un dato "OUT" de la placa Arduino a la aplicación. Puede ser declarada en el sistema HMI mediante el uso de la función attachBooleanOut en el *sketch*.

5. Virtual int-in: Es una variable int utilizado como un dato "IN" de la aplicación para el Arduino. Puede ser declarada en el sistema HMI mediante el uso de la función attachIntIn en el *sketch*.

6. Virtual int-out: Es una variable int utilizado como un dato "OUT" de la placa Arduino a la aplicación. Puede ser declarada en el sistema HMI mediante el uso de la función attachIntOut en el *sketch*.

Funciones para Arduino

Una de las grandes ventajas de la esta aplicación es que la programación de Arduino para hacer que la comunicación es muy fácil y simple, hay una serie de funciones que hacen que esto sea posible, esos son:

- *Hmi.attachIntIn* (variable int, letra char);

El *attachIntIn* le permite adjuntar una variable de tipo int para el sistema HMI, esta variable se debe ya declaró en el *sketch* Arduino y se utiliza para leer datos desde el dispositivo Android. El argumento letter es una etiqueta de tipo char para identificar la variable int dentro de la aplicación HMI Controller.

- *Hmi.attachIntOut* (variable int, char letter);

Esta función adjuntar una variable Int al sistema HMI, esta variable se debe ya declaró en el *sketch* Arduino y se utiliza para escribir datos en el dispositivo

Android. El argumento letter es una etiqueta de tipo char para identificar la variable int dentro de la aplicación HMI Controller.

- *Hmi.attachBooleanIn* (variable booleana, char letter);

El *attachBooleanIn* se utiliza para conectar una variable de tipo booleano para el sistema HMI, esta variable se debe ya declaró en el *sketch* Arduino y se utiliza para leer datos desde el dispositivo Android. El argumento letter es una etiqueta de tipo char para identificar la variable booleana dentro de la aplicación HMI Controller.

- *Hmi.attachBooleanOut* (variable booleana, char letter);

La función *attachBooleanOut* adjuntar una variable booleana para el sistema HMI, esta variable se debe ya declaró en el *sketch* Arduino y se utiliza para escribir datos en el dispositivo Android. El argumento letter es un char usado como una etiqueta para identificar la variable booleana dentro de la aplicación HMI Controller.

Para utilizar el HMI Controller con tu placa Arduino, debe implementar una de las siguientes funciones al final de su *sketch*:

- *Hmi.softSerial* (SoftwareSerial mySerial): se utiliza cuando se está implementando una conexión Bluetooth con la biblioteca Arduino SoftwareSerial, esto se recomienda para ser usado en la aplicación de la Arduino UNO y Arduino micro.

- *Hmi.hardSerial* (HardwareSerial mySerial): se utiliza cuando se está implementando una conexión Bluetooth con la biblioteca Arduino HardwareSerial, esto se recomienda que sea el uso al aplicar la Arduino Mega.

- *Hmi.n Ethernet* (Server myServer): Se utiliza cuando se está implementando un Arduino Ethernet shield.

Esta función transmite los datos entre los dos dispositivos, es decir, que permite la conexión de la placa Arduino y su dispositivo Android.