Sistemas embebidos

Lic. Sol Pedre

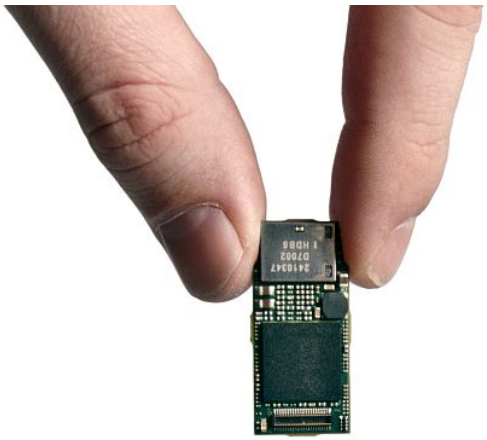
Laboratorio de Robótica y Sistemas Embebidos
Departamento de Computación - FCEN - UBA

12 de Octubre de 2012

El menú del dia

- ¿Qué es un Sistema Embebido?
- ¿Qué tienen adentro?
- ¿Cómo se diseña un sistema embebido? Un recorrido por algunos desarrollos en el LRSE:
  - Co-diseños usando ASICs y diseño de circuitos: El ExaBot.
  - Diseños en FPGA: Detección de hotspots en un UAV.
  - Co-diseños en FPGA: Detección de múltiples robots

¿Qué es un Sistema Embebido?



- El mundo está lleno de ellos.
- Se pueden definir como todo sistema que NO es una PC de escritorio, ni un servidor, ni una workstation, ni una supercomputadora, ni un cluster distribuido, en fin, NO es un sistema programable de propósito general.

Algunos ejemplos

Son sistemas diseñados para cumplir una o pocas funciones dedicadas y que está embebido como parte de algún dispositivo de hardware completo. Por ejemplo:

- Embebidos en industria automotriz : navegador GPS, de la aviación: piloto automático, control de aterrizaje
- Embebidos en telecomunicaciones: routers, modems
- Embebidos en comunicaciones: teléfonos celulares
- Embebidos en el hogar: control de heladeras, microondas, robots que aspiran, cortan pasto.
- Etc, pero muchos etcéteras.

¿Qué tienen adentro?

Todos (o casi casi todos) están compuestos por software y hardware diseñados específicamente para la tarea que tienen que cumplir, e interactuando muy cercanamente.

- En realidad, aproximadamente el 0 % de los microprocesadores que se fabrican se usan en PCs de escritorio. El 100 % restante se usan en embebidos.
- En los últimos años alrededor de 500 millones de microprocesadores se usaron en PCs y 10 mil millones en embebidos.

¿Qué tienen adentro?

Elementos tipo CPU corriendo software...

- Procesadores.
- Microcontroladores.
- Digital Signal Processors (DSP).

... + Hardware

- Circuitos Integrados (ASIC).
- Lógica Programable (FPGA).
- Circuitos digitales y analógicos diseñados específicamente.

¿Qué tienen adentro?

Elementos tipo CPU corriendo software...

- Procesadores.
- Microcontroladores.
- Digital Signal Processors (DSP).

... + Hardware

- Circuitos Integrados (ASIC).
- Lógica Programable (FPGA).
- Circuitos digitales y analógicos diseñados específicamente.

El ExaBot

Un sistema embebido de control con procesadores embebidos, microcontroladores, ASICs y diseño de placas.

- Software:
  - Un procesador embebido
  - Tres microcontroladores.
- Hardware:
  - Varios ASICs
  - Circuitos digitales y analógicos desarrollados particularmente para el robot.

Metodología de co-diseño

- Objetivo
- Definición de locomoción, sensores, cuerpo, capacidad de procesamiento.
- Partición en subsistemas.
- Refinamiento de esos subsistemas: hardware, software y prototipado.
- Integración de los subsistemas: comunicación, etapa de power, diseño final de la placa integrando todos los subsistemas, software final.
- Montado.

Metodología de co-diseño

- Objetivo
- Definición de locomoción, sensores, cuerpo, capacidad de procesamiento.
- Partición en subsistemas.
- Refinamiento de esos subsistemas: hardware, software y prototipado.
- Integración de los subsistemas: comunicación, etapa de power, diseño final de la placa integrando todos los subsistemas, software final.
- Montado.

Metodología de co-diseño

- Objetivo
- Definición de locomoción, sensores, cuerpo, capacidad de procesamiento.
- Partición en subsistemas.
- Refinamiento de esos subsistemas: hardware, software y prototipado.
- Integración de los subsistemas: comunicación, etapa de power, diseño final de la placa integrando todos los subsistemas, software final.
- Montado.

Metodología de co-diseño

- Objetivo
- Definición de locomoción, sensores, cuerpo, capacidad de procesamiento.
- Partición en subsistemas.
- Refinamiento de esos subsistemas: hardware, software y prototipado.
- Integración de los subsistemas: comunicación, etapa de power, diseño final de la placa integrando todos los subsistemas, software final.
- Montado.

Metodología de co-diseño

- Objetivo
- Definición de locomoción, sensores, cuerpo, capacidad de procesamiento.
- Partición en subsistemas.
- Refinamiento de esos subsistemas: hardware, software y prototipado.
- Integración de los subsistemas: comunicación, etapa de power, diseño final de la placa integrando todos los subsistemas, software final.
- Montado.

Metodología de co-diseño

- Objetivo
- Definición de locomoción, sensores, cuerpo, capacidad de procesamiento.
- Partición en subsistemas.
- Refinamiento de esos subsistemas: hardware, software y prototipado.
- Integración de los subsistemas: comunicación, etapa de power, diseño final de la placa integrando todos los subsistemas, software final.
- Montado.

Metodología de co-diseño

- Objetivo
- Definición de locomoción, sensores, cuerpo, capacidad de procesamiento.
- Partición en subsistemas.
- Refinamiento de esos subsistemas: hardware, software y prototipado.
- Integración de los subsistemas: comunicación, etapa de power, diseño final de la placa integrando todos los subsistemas, software final.
- Montado.

Objetivo

Tener una única plataforma robótica que permita hacer

1. *Investigación:* centradas en navegación autónoma (sobre todo adentro).
2. *Extensión:* actividades basadas en programar comportamientos simples en el robot (evadir obstáculos, seguir línea, etc).
3. *Educación en la Facu:* Visión en Robótica, otras materias.
4. *Bajo costo* Comprado con otros similares.
5. *Y que esté hecho acá!*

Requerimiento

Dado que el robot tiene un amplio espectro de aplicaciones el mayor requerimiento es:

Fácilmente reconfigurable: tiene que soportar diferentes sensores y unidades de procesamiento y ser fácilmente reconfigurable con un subconjunto de los mismos para una actividad particular

Locomoción y cuerpo

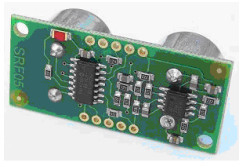
Dado que el estudio de la locomoción y los temas mecánicos NO son un objetivo del Exa, podemos seguir el principio de ingeniería **K**eep **I**t **S**imple **S**tupid: La solución más sencilla que cumple con los requerimientos es la adecuada.

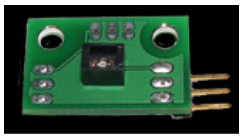
Figura : El Traxster kit

Incluye motores de corriente continua de 7,2V y 2Amp con encoders.

Sensores

Muchos y de distinto tipo para poder realizar experiencias variadas en sus múltiples objetivos.





- Anillo de 8 telémetros infrarrojos - 6 a 30 cm.
- 1 Sonar Devanatech SRF05 - 4 mm a 4 mts.
- 2 Line-following.
- 2 Contacto.
- Web-Cam para visión estéreo.
- Sensor de bateria.
- Para control de motores: encoders + sensor de consumo
- ... y puertos de expansión!!

Capacidad de procesamiento

Dado que los diferentes objetivos del Exa generan requerimientos de capacidad de cómputo muy diferentes, volvemos al requerimiento de reconfigurabilidad. Dividimos en dos niveles: procesamiento de bajo nivel para control de los motores y los sensores, y de alto nivel para algoritmos mas complejos. La unidad de procesamiento de alto nivel se pensó para que pudiera ser fácilmente removida o reemplazada.

Partición en subsistemas

- Tres básicos: control de sensores, de motores, procesamiento de alto-nivel.
- Unidades de procesamiento:
  - Para sensores y motores: microcontroladores. Familia Microchip PIC18F.
  - Para alto nivel: procesador embebido. PC104 - ARM de 200 Mhz.
- Comunicación:
  - ¿Qué tienen los PIC y la PC104? SPI, (I^2C), USART.
  - Usamos el más simple que cumple los requerimientos: SPI!
- En total: 7 subsistemas (sensores, 2 motores, procesamiento, comunicación, power, programación)

Partición en subsistemas

- Tres básicos: control de sensores, de motores, procesamiento de alto-nivel.
- Unidades de procesamiento:
  - Para sensores y motores: microcontroladores. Familia Microchip PIC18F.
  - Para alto nivel: procesador embebido. PC104 - ARM de 200 Mhz.
- Comunicación:
  - ¿Qué tienen los PIC y la PC104? SPI, (I^2C), USART.
  - Usamos el más simple que cumple los requerimientos: SPI!
- En total: 7 subsistemas (sensores, 2 motores, procesamiento, comunicación, power, programación)

Partición en subsistemas

- Tres básicos: control de sensores, de motores, procesamiento de alto-nivel.
- Unidades de procesamiento:
  - Para sensores y motores: microcontroladores. Familia Microchip PIC18F.
  - Para alto nivel: procesador embebido. PC104 - ARM de 200 Mhz.
- Comunicación:
  - ¿Qué tienen los PIC y la PC104? SPI, (I^2C), USART.
  - Usamos el más simple que cumple los requerimientos: SPI!
- En total: 7 subsistemas (sensores, 2 motores, procesamiento, comunicación, power, programación)

Partición en subsistemas

- Tres básicos: control de sensores, de motores, procesamiento de alto-nivel.
- Unidades de procesamiento:
  - Para sensores y motores: microcontroladores. Familia Microchip PIC18F.
  - Para alto nivel: procesador embebido. PC104 - ARM de 200 Mhz.
- Comunicación:
  - ¿Qué tienen los PIC y la PC104? SPI, (I^2C), USART.
  - Usamos el más simple que cumple los requerimientos: SPI!
- En total: 7 subsistemas (sensores, 2 motores, procesamiento, comunicación, power, programación)

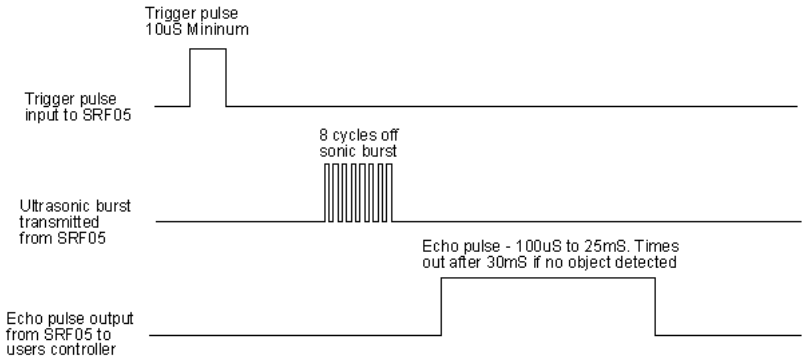
Partición en subsistemas

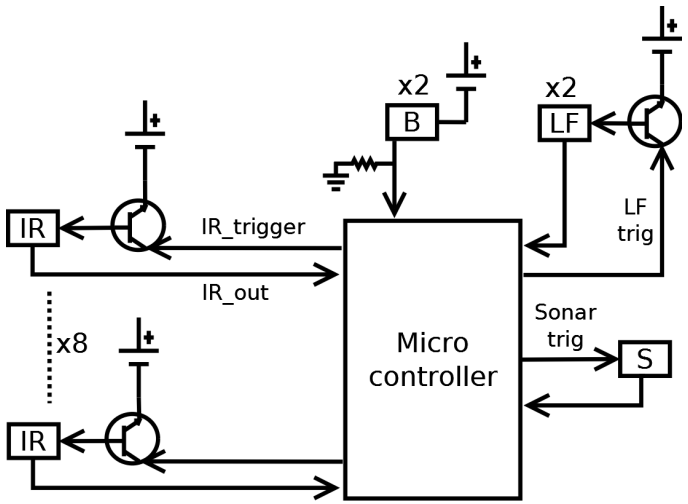
- Tres básicos: control de sensores, de motores, procesamiento de alto-nivel.
- Unidades de procesamiento:
  - Para sensores y motores: microcontroladores. Familia Microchip PIC18F.
  - Para alto nivel: procesador embebido. PC104 - ARM de 200 Mhz.
- Comunicación:
  - ¿Qué tienen los PIC y la PC104? SPI, (I^2C), USART.
  - Usamos el más simple que cumple los requerimientos: SPI!
- En total: 7 subsistemas (sensores, 2 motores, procesamiento, comunicación, power, programación)

Refinamiento de cada subsistema: sensores

Objetivo: controlar 8 IRs, 1 sonar, 2 bumpers, 2 linefollowing.

Hardware



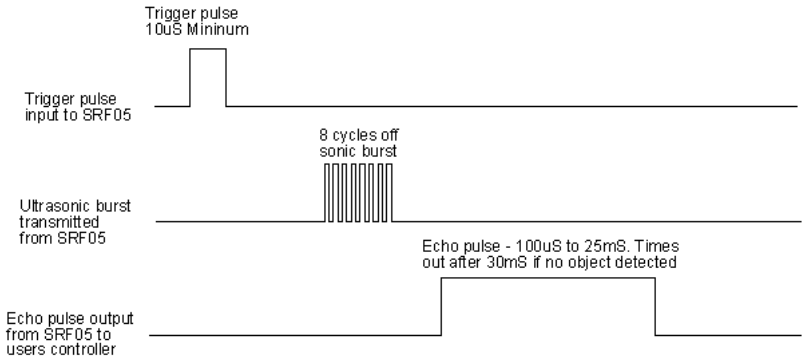


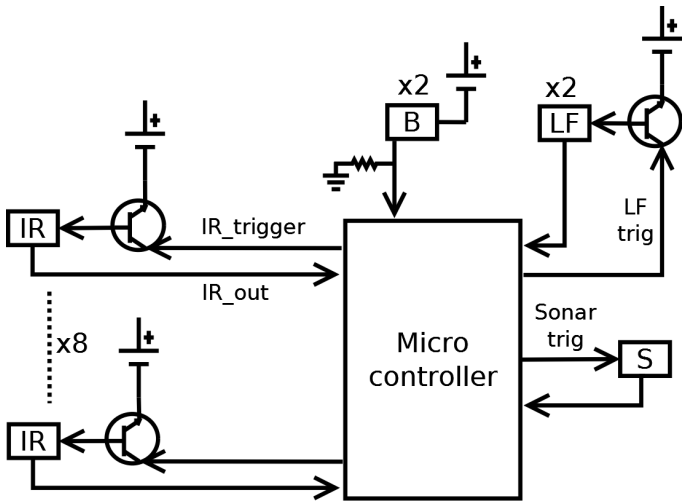
- ¿Qué hardware necesitas para controlar cada sensor? VER DATASHEETS!!!!
- Por ej, el sonar: necesitás un pin digital para triger, uno para output y un módulo CCP.
- En total, se requieren para el microcontrolador (solo para controlar los sensores): 8 pins AN, 16 pins Dig, 2 timers, un módulo ADC y un módulo CCP.

Refinamiento de cada subsistema: sensores

Objetivo: controlar 8 IRs, 1 sonar, 2 bumpers, 2 linefollowing.

Hardware



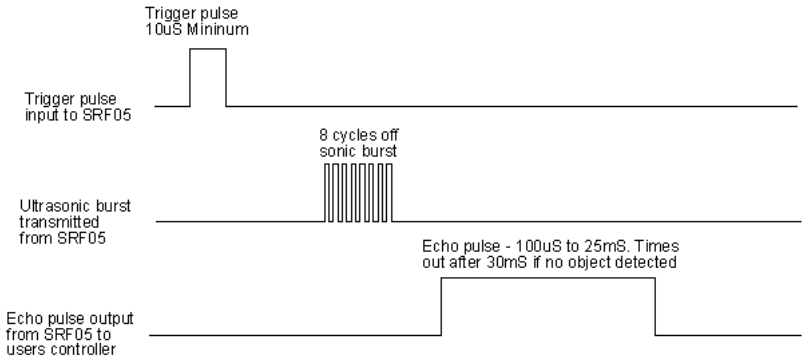


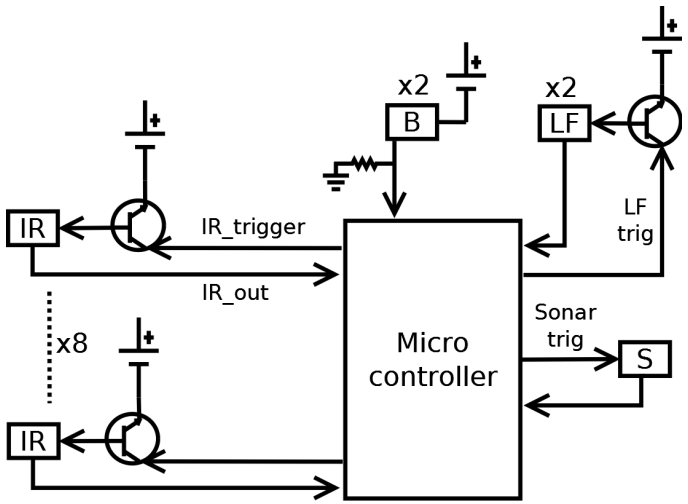
- ¿Qué hardware necesitas para controlar cada sensor? VER DATASHEETS!!!!
- Por ej, el sonar: necesitás un pin digital para triger, uno para output y un módulo CCP.
- En total, se requieren para el microcontrolador (solo para controlar los sensores): 8 pins AN, 16 pins Dig, 2 timers, un módulo ADC y un módulo CCP.

Refinamiento de cada subsistema: sensores

Objetivo: controlar 8 IRs, 1 sonar, 2 bumpers, 2 linefollowing.

Hardware



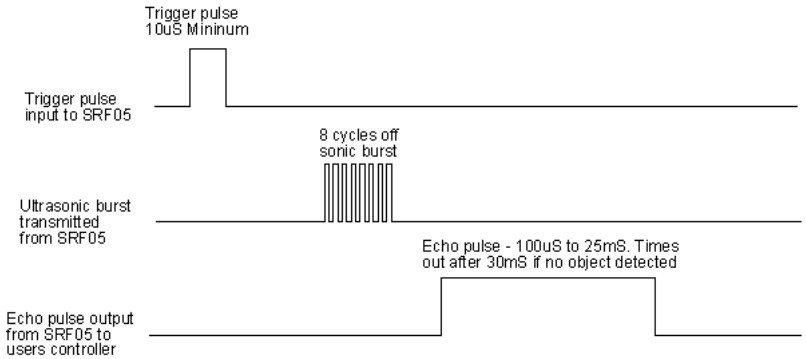


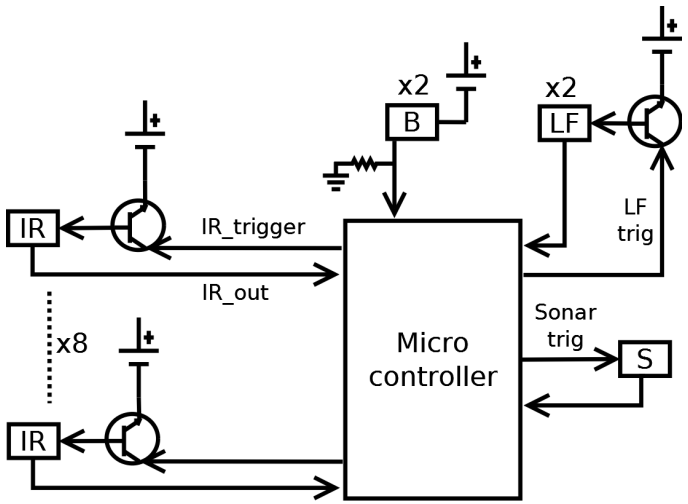
- ¿Qué hardware necesitas para controlar cada sensor? VER DATASHEETS!!!!
- Por ej, el sonar: necesitás un pin digital para triger, uno para output y un módulo CCP.
- En total, se requieren para el microcontrolador (solo para controlar los sensores): 8 pins AN, 16 pins Dig, 2 timers, un módulo ADC y un módulo CCP.

Refinamiento de cada subsistema: sensores

Objetivo: controlar 8 IRs, 1 sonar, 2 bumpers, 2 linefollowing.

Hardware





- ¿Qué hardware necesitas para controlar cada sensor? VER DATASHEETS!!!!
- Por ej, el sonar: necesitás un pin digital para triger, uno para output y un módulo CCP.
- En total, se requieren para el microcontrolador (solo para controlar los sensores): 8 pins AN, 16 pins Dig, 2 timers, un módulo ADC y un módulo CCP.

Refinamiento: Tips para Software

- En los robots, el control en tiempo real y a intervalos conocidos de sensores y actuadores es *fundamental*, ya que se mueven en el mundo real.
- Para lograr esto tenemos:
  - Modulos de Hardware que actuan en paralelo, realizan parte del procesamiento e interrumpen cuando tienen datos (tanto en PIC's como en PC104).
  - Programación con varios procesos en la PC104.
  - Programación en módulos y con tiempos controlados mediante medidos.
- Herramientas: MPLAB, C-18 compiler. Cross-compiler para PC104.

Refinamiento: Tips para Software

- En los robots, el control en tiempo real y a intervalos conocidos de sensores y actuadores es *fundamental*, ya que se mueven en el mundo real.
- Para lograr esto tenemos:
  - Modulos de Hardware que actuan en paralelo, realizan parte del procesamiento e interrumpen cuando tienen datos (tanto en PICs como en PC104)
  - Programación con varios procesos en la PC104
  - Programación en módulos y con tiempos controlados mediante
- Herramientas: MPLAB, C-18 compiler. Cross-compiler para PC104.

Refinamiento: Tips para Software

- En los robots, el control en tiempo real y a intervalos conocidos de sensores y actuadores es *fundamental*, ya que se mueven en el mundo real.
- Para lograr esto tenemos:
  - Modulos de Hardware que actuan en paralelo, realizan parte del procesamiento e interrumpen cuando tienen datos (tanto en PICs como en PC104)
  - Programación con varios procesos en la PC104.
  - Programación en módulos y con tiempos ESTRICTAMENTE medidos.
- Herramientas: MPLAB, C-18 compiler. Cross-compiler para PC104.

Refinamiento: Tips para Software

- En los robots, el control en tiempo real y a intervalos conocidos de sensores y actuadores es *fundamental*, ya que se mueven en el mundo real.
- Para lograr esto tenemos:
  - Modulos de Hardware que actuan en paralelo, realizan parte del procesamiento e interrumpen cuando tienen datos (tanto en PICs como en PC104)
  - Programación con varios procesos en la PC104.
  - Programación en módulos y con tiempos ESTRICTAMENTE medidos.
- Herramientas: MPLAB, C-18 compiler. Cross-compiler para PC104.

Refinamiento: Tips para Software

- En los robots, el control en tiempo real y a intervalos conocidos de sensores y actuadores es *fundamental*, ya que se mueven en el mundo real.
- Para lograr esto tenemos:
  - Modulos de Hardware que actuan en paralelo, realizan parte del procesamiento e interrumpen cuando tienen datos (tanto en PICs como en PC104)
  - Programación con varios procesos en la PC104.
  - Programación en módulos y con tiempos ESTRICTAMENTE medidos.
- Herramientas: MPLAB, C-18 compiler. Cross-compiler para PC104.

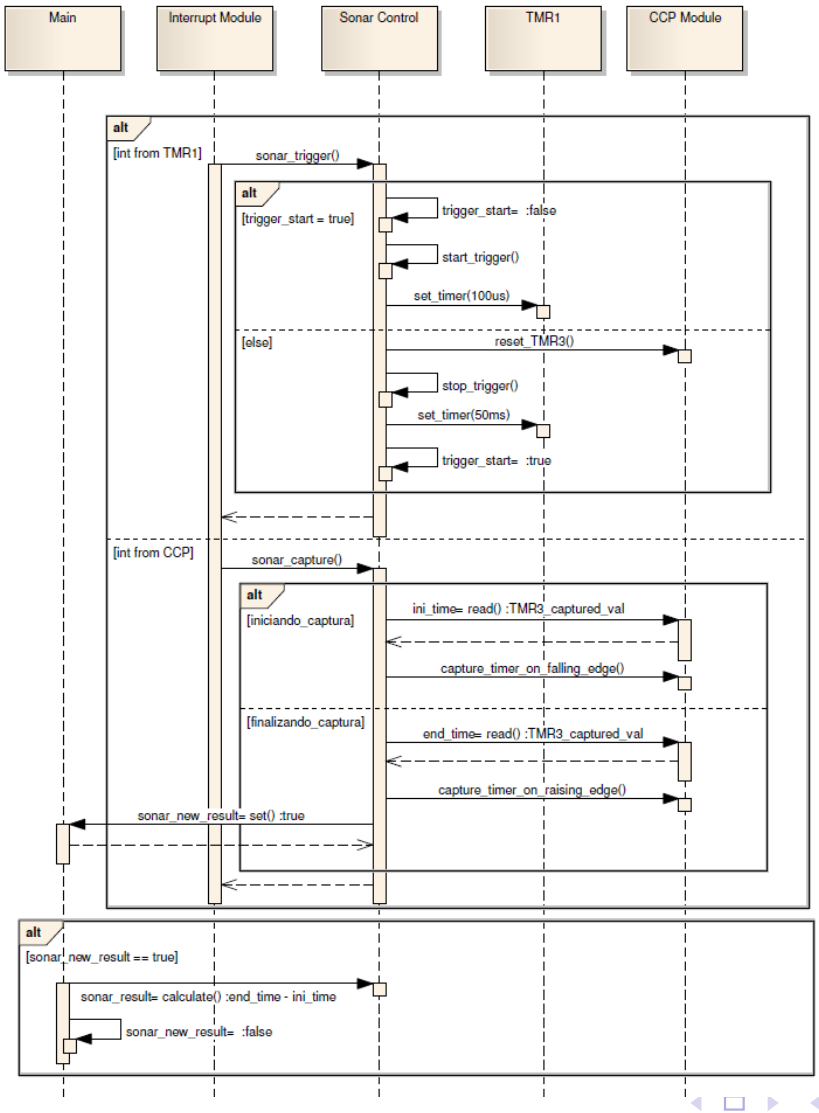
Refinamiento: Tips para Software

- En los robots, el control en tiempo real y a intervalos conocidos de sensores y actuadores es *fundamental*, ya que se mueven en el mundo real.
- Para lograr esto tenemos:
  - Modulos de Hardware que actuan en paralelo, realizan parte del procesamiento e interrumpen cuando tienen datos (tanto en PICs como en PC104)
  - Programación con varios procesos en la PC104.
  - Programación en módulos y con tiempos ESTRICTAMENTE medidos.
- Herramientas: MPLAB, C-18 compiler. Cross-compiler para PC104.

Refinamiento: Tips para Software

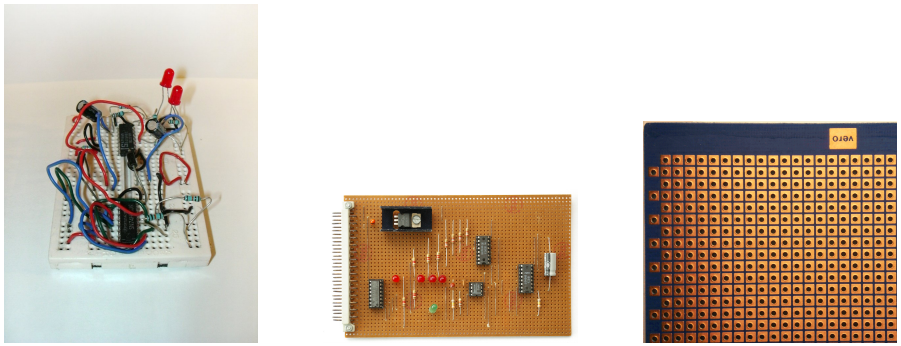
- En los robots, el control en tiempo real y a intervalos conocidos de sensores y actuadores es *fundamental*, ya que se mueven en el mundo real.
- Para lograr esto tenemos:
  - Modulos de Hardware que actuan en paralelo, realizan parte del procesamiento e interrumpen cuando tienen datos (tanto en PICs como en PC104)
  - Programación con varios procesos en la PC104.
  - Programación en módulos y con tiempos ESTRICTAMENTE medidos.
- Herramientas: MPLAB, C-18 compiler. Cross-compiler para PC104.

Refinamiento: Software



Refinamiento: Prototipado

Se testean en una placa prototipo subconjuntos de funcionalidades: para eso se testea el hardware + el software.

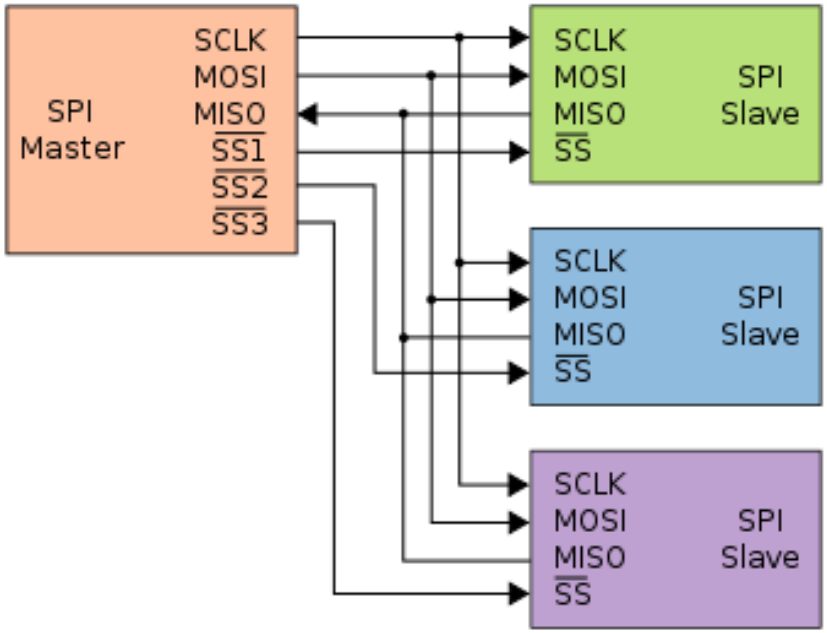


a b c

Figura : Prototyping boards: **a)** Breadboard, **b)** Stripboard, **c)** Perfboard

Herramientas de testing: Debugger de MPLAB, osciloscopio, fuente regulada, tester.

Integración: Comunicación

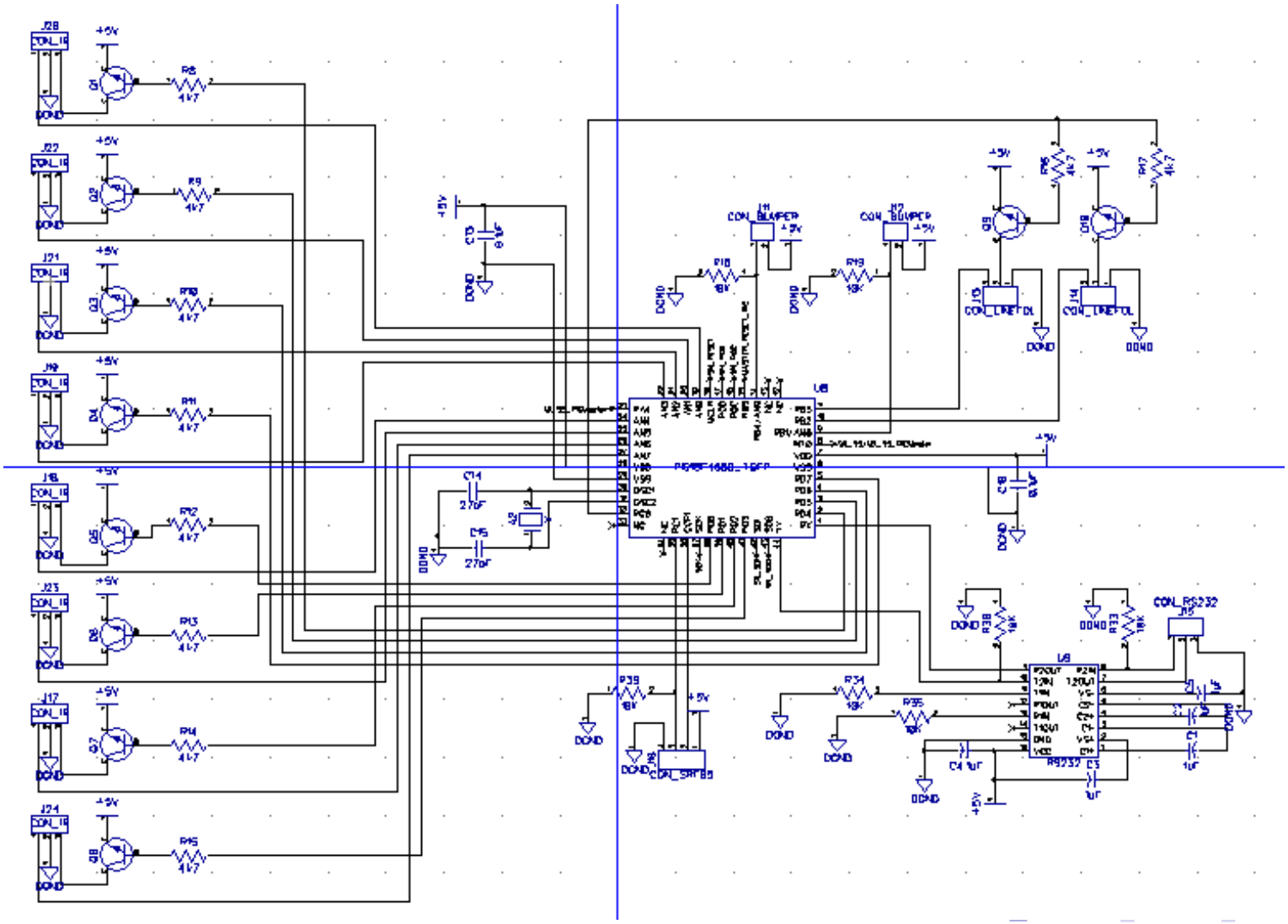


- Dos configuraciones: con y sin la PC104. El código del sistema de comunicación está distribuido en los PICs y PC104.
- Implementa un modelo OSI con las siguientes capas: física, link, red, aplicación. En los slaves sin capa de red.

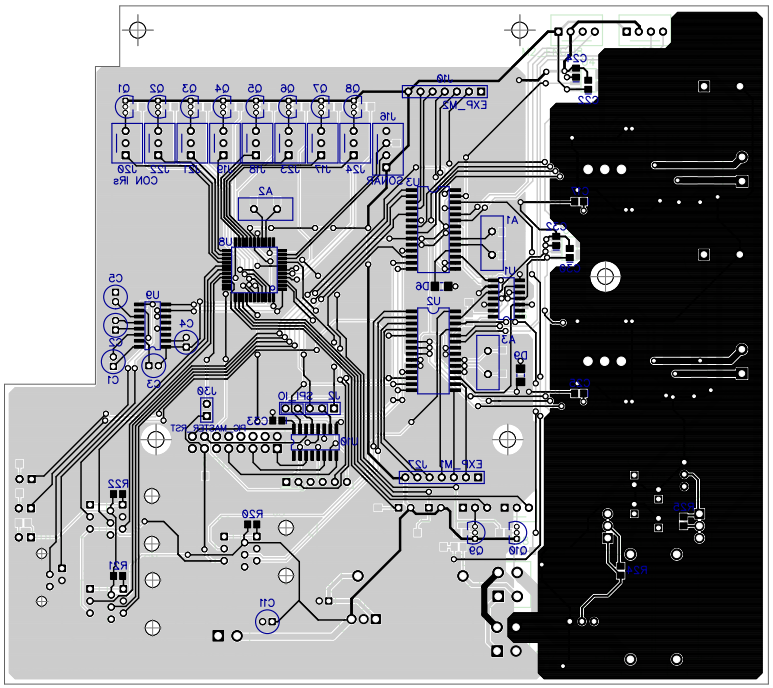
Integración: Power

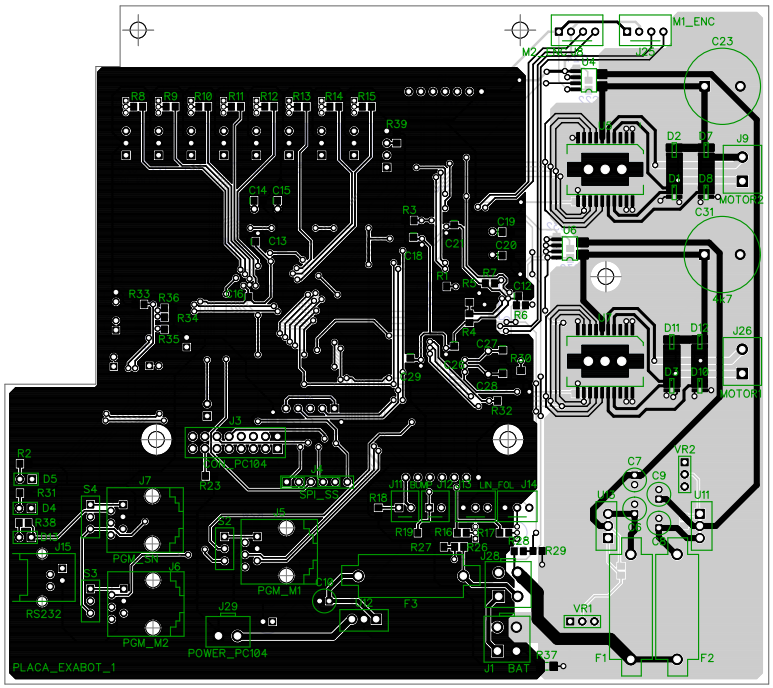
- Una vez que cada sensor, actuador, IC, y circuitos están definidos, se pueden calcular los requerimientos exactos para las baterias.
- Dos subsistemas de power:
  - para los motores (7,2V regulados).
  - para los ICs y los sensores (5V regulados)
- Celdas de Ion-Litio (3,6V a 4,2V, 1.9 Amp/h)

Integración: Esquemático y PCB final



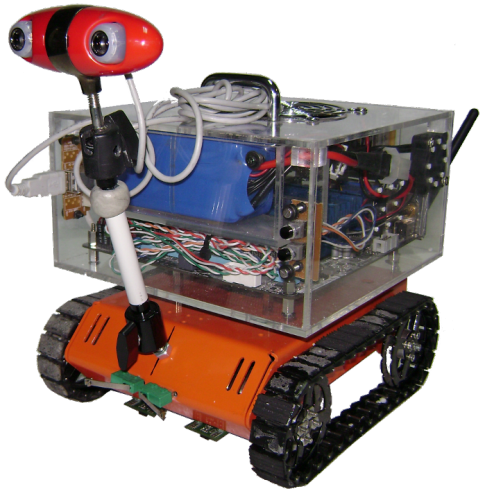
Integración: Esquemático y PCB final





Herramientas: DipTrace o similar.

Montado

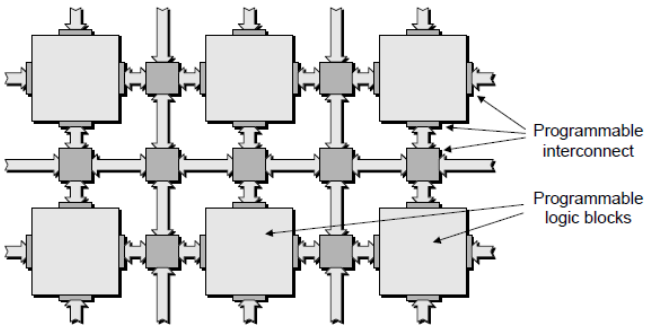


Hasta ahora y lo que viene

- Hasta ahora vimos el co-diseño y desarrollo de un sistema embebido de contol usando procesadores embebidos y microcontroladores corriendo software junto con ASICs y diseño de circuitos (análogicos y digitales).
- Ahora entramos en el mundo de la lógica programable, utilizada ampliamente en sistemas embebidos de procesamiento de señales.
- Vamos a ver por un lado un sistema completamente diseñado en lógica y otro co-diseñado.

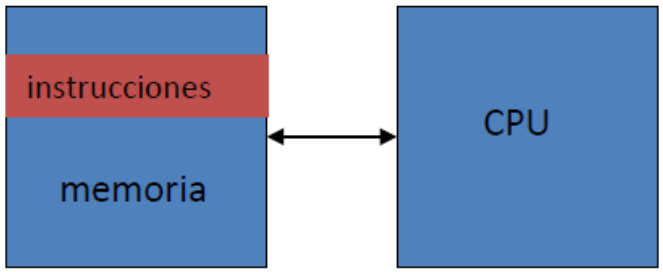
¿Qué es una FPGA?

Field Programmable Gate Array



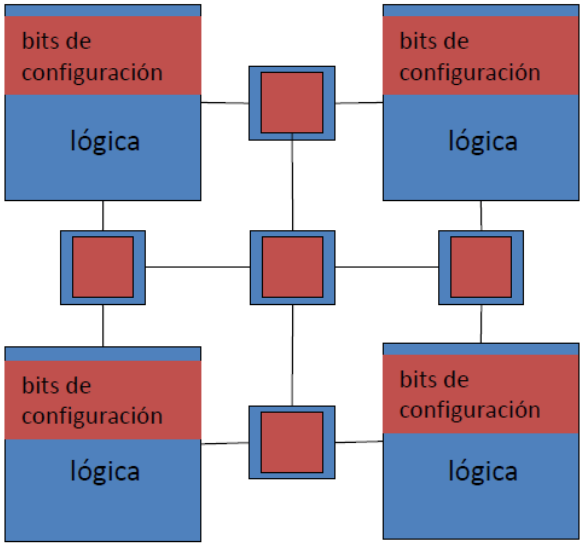
- ICs que contienen bloques configurables de lógica y conexiones configurables.
- Se configuran para realizar distintas tareas. Algunas son programados una sola vez(OTP), otras muchas
- Los FPGA se programan "in the field": no los programa el fabricante, sino el desarrollador.
- ¿Porqué Gate Array? Es el nombre de un tipo de tecnología de fabricación de los ASIC.

Microprocesadores y FPGA



Programación CPU:

- Las instrucciones se obtienen de la memoria
- Las instrucciones seleccionan operaciones complejas

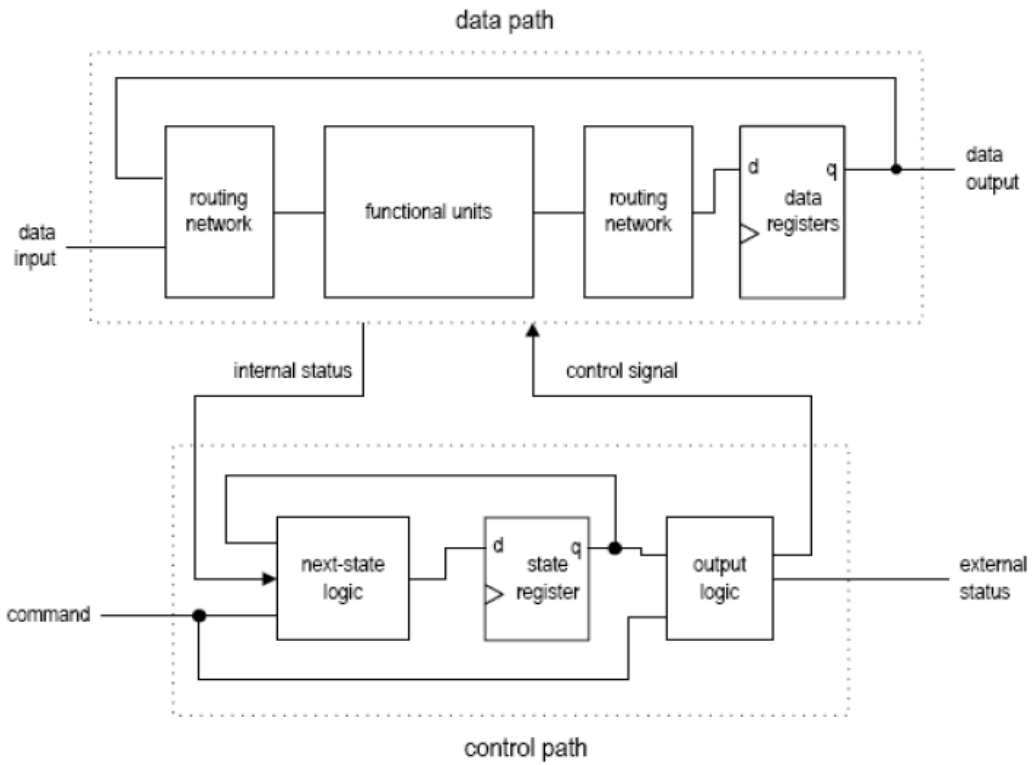


Configuración FPGA:

- Bits permanecen en el dispositivo que programan
- Un bit de configuración controla un switch o un bit de lógica

Metodología de Diseño

FSMD: Finite State Machine con Datapath



- Control Path, FSM con:
  - Registro de estado
  - Salida
  - Siguiente Estado
- Data Path contiene
  - Registros para los datos
  - Unidades funcionales donde se computan las operaciones
  - Ruteo entre los datos y las unidades funcionales
- Ambos controlados por el mismo reloj.

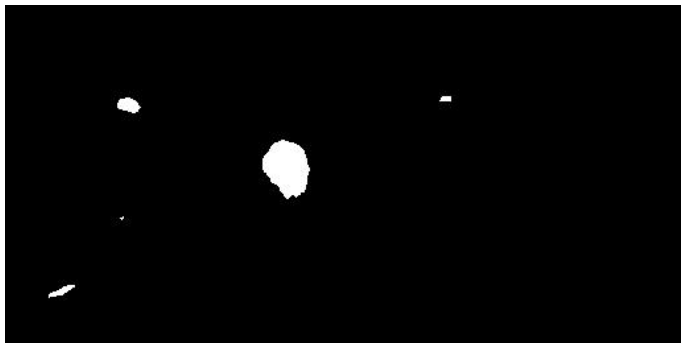
¿Cómo se programan?

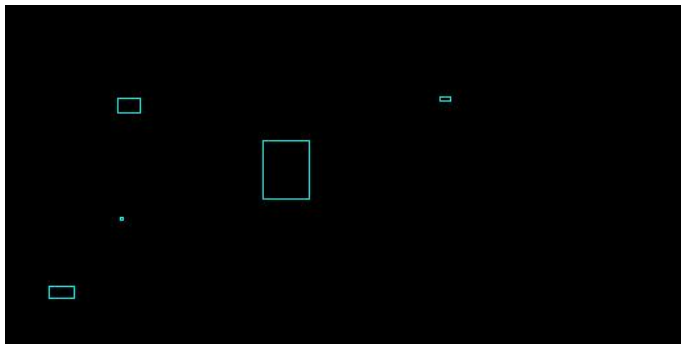
- Hardware Description Languages (VHDL, Verilog).
- Herramientas propietarias del desarrollador de FPGAs particular. Los dos grandes: Xilinx y Altera. Nosotros usamos Xilinx, asi que es el ISE.
- Testing? Simulación! Simuladores como QuestaSim. Otras herramientas: ChipScope, Osciloscopio.
- Verificación... eso ya son palabras mayores!!!!

Detección de HotSpots en tiempo real utilizando FPGA

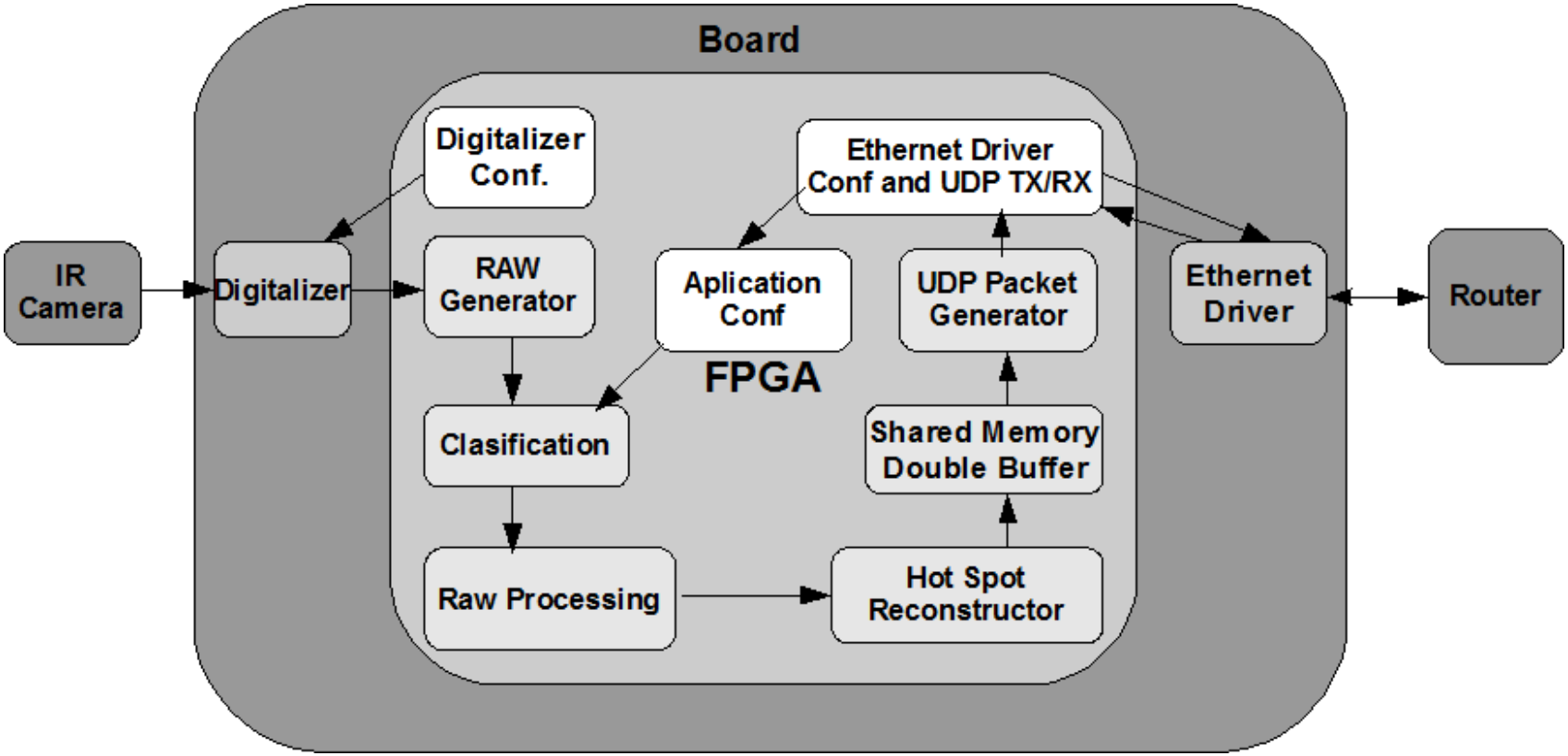
Diseño completo en hardware para sensado remoto con restricciones de espacio, tiempo real y consumo

- **Problema**: identificar zonas calientes en imágenes capturadas por la cámara infrarroja de un UAV en tiempo real.
- Desarrollamos un algoritmo de segmentación para identificar la posición y caraterísticas espaciales del hot spot.
- Esta información se transmite a bomberos combatiendo un incendio forestal.





Implementación en FPGA



Diseño completo de la placa, incluyendo el chip digitalizador para la cámara infrarroja y el módulo Ethernet por hardware para la transmisión de los resultados.

Resultados

- Segmentamos de manera exitosa la imagen mientras es capturada, con un delay total de procesamiento igual al tiempo que lleva capturar un pixel.
- Los resultados de la imagen previa se transmiten en paralelo con el procesamiento de la imagen actual para no agregar delays innecesarios.
- El delay de procesamiento es independiente del tamaño de la imagen. Presentamos además fórmulas para calcular el tamaño de la FPGA necesario para una determinada aplicación.
- La solución es independiente de la cámara IR particular, sin necesidad de memoria extra para almacenamiento de la imagen.
- La implementación se realizó completamente en lógica programable, desde la adquisición de la imagen hasta la comunicación de los resultados utilizando UDP, diseñando a tal efecto mas de 40 módulos VHDL.

¿Porqué Co-Diseño?

- ¿Porqué no todo (o casi) en un procesador?
  - Simplemente porque en muchísimas aplicaciones los requerimientos de tiempo real y consumo de potencia no dan para usar procesadores.
- ¿Porqué no todo en lógica/hardware?
  - Porque es muy pero muy difícil de diseñar y de verificar!!
- Y porque no tiene sentido: que cada parte haga lo que mejor le sale!
  - CPU - control (generalmente muy secuencial)
  - Lógica - procesamiento de los datos (generalmente paralelizable)
- En general, la mayor parte del tiempo se consume en muy pocas funciones!

¿Porqué Co-Diseño?

- ¿Porqué no todo (o casi) en un procesador?
  - Simplemente porque en muchísimas aplicaciones los requerimientos de tiempo real y consumo de potencia no dan para usar procesadores.
- ¿Porqué no todo en lógica/hardware?
  - Porque es muy pero muy difícil de diseñar y de verificar!!!
- Y porque no tiene sentido: que cada parte haga lo que mejor le sale!
  - CPU - control (generalmente muy secuencial)
  - Lógica - procesamiento de los datos (generalmente paralelizable)
- En general, la mayor parte del tiempo se consume en muy pocas funciones!

¿Porqué Co-Diseño?

- ¿Porqué no todo (o casi) en un procesador?
  - Simplemente porque en muchísimas aplicaciones los requerimientos de tiempo real y consumo de potencia no dan para usar procesadores.
- ¿Porqué no todo en lógica/hardware?
  - Porque es muy pero muy difícil de diseñar y de verificar!!!
- Y porque no tiene sentido: que cada parte haga lo que mejor le sale!
  - CPU - control (generalmente muy secuencial)
  - Lógica - procesamiento de los datos (generalmente paralelizable)
- En general, la mayor parte del tiempo se consume en muy pocas funciones!

¿Porqué Co-Diseño?

- ¿Porqué no todo (o casi) en un procesador?
  - Simplemente porque en muchísimas aplicaciones los requerimientos de tiempo real y consumo de potencia no dan para usar procesadores.
- ¿Porqué no todo en lógica/hardware?
  - Porque es muy pero muy dificil de diseñar y de verificar!!!
- Y porque no tiene sentido: que cada parte haga lo que mejor le sale!
  - CPU - control (generalmente muy secuencial)
  - Lógica - procesamiento de los datos (generalmente paralelizable)
- En general, la mayor parte del tiempo se consume en muy pocas funciones!

¿Porqué Co-Diseño?

- ¿Porqué no todo (o casi) en un procesador?
  - Simplemente porque en muchísimas aplicaciones los requerimientos de tiempo real y consumo de potencia no dan para usar procesadores.
- ¿Porqué no todo en lógica/hardware?
  - Porque es muy pero muy dificil de diseñar y de verificar!!!
- Y porque no tiene sentido: que cada parte haga lo que mejor le sale!
  - CPU - control (generalmente muy secuencial)
  - Lógica - procesamiento de los datos (generalmente paralelizable)
- En general, la mayor parte del tiempo se consume en muy pocas funciones!

¿Porqué Co-Diseño?

- ¿Porqué no todo (o casi) en un procesador?
  - Simplemente porque en muchísimas aplicaciones los requerimientos de tiempo real y consumo de potencia no dan para usar procesadores.
- ¿Porqué no todo en lógica/hardware?
  - Porque es muy pero muy dificil de diseñar y de verificar!!!
- Y porque no tiene sentido: que cada parte haga lo que mejor le sale!
  - CPU - control (generalmente muy secuencial)
  - Lógica - procesamiento de los datos (generalmente paralelizable)
- En general, la mayor parte del tiempo se consume en muy pocas funciones!

¿Porqué Co-Diseño?

- ¿Porqué no todo (o casi) en un procesador?
  - Simplemente porque en muchísimas aplicaciones los requerimientos de tiempo real y consumo de potencia no dan para usar procesadores.
- ¿Porqué no todo en lógica/hardware?
  - Porque es muy pero muy dificil de diseñar y de verificar!!!
- Y porque no tiene sentido: que cada parte haga lo que mejor le sale!
  - CPU - control (generalmente muy secuencial)
  - Lógica - procesamiento de los datos (generalmente paralelizable)
- En general, la mayor parte del tiempo se consume en muy pocas funciones!

Metodología de co-diseño

- **Motivación**: muchas aplicaciones embebidas necesita tiempo real y al mismo tiempo tienen restricciones de consumo, tamaño y peso. Para estos desarrollos son comunes soluciones basadas en FPGA, pero su mayor obstáculo es que los tiempos de desarrollo son largos (y la verificación compleja).
- **Objetivo**: Lograr soluciones embebidas con aceleración por hardware que cumplan con los requerimentos de tiempo real, pero con tiempos de desarrollo similares a los de proyectos de software.
- **A continuación**:
  1. Presentamos una *metodología de co-diseño* para sistemas embebidos centrados en procesador con aceleración por hardware usando FPGA.
  2. La aplicamos a un proyecto de imágenes para batallas de robots en una arena.

Metodología de co-diseño

- **Motivación**: muchas aplicaciones embebidas necesita tiempo real y al mismo tiempo tienen restricciones de consumo, tamaño y peso. Para estos desarrollos son comunes soluciones basadas en FPGA, pero su mayor obstáculo es que los tiempos de desarrollo son largos (y la verificación compleja).
- **Objetivo**: Lograr soluciones embebidas con aceleración por hardware que cumplan con los requerimentos de tiempo real, pero con tiempos de desarrollo similares a los de proyectos de software.
- **A continuación**:
  - Presentamos una *metodología de co-diseño para sistemas embebidos centrados en procesador con aceleración por hardware usando FPGA.*
  - La aplicamos a un proceso de imágenes para localizar múltiples robots en una arena

Metodología de co-diseño

- **Motivación**: muchas aplicaciones embebidas necesita tiempo real y al mismo tiempo tienen restricciones de consumo, tamaño y peso. Para estos desarrollos son comunes soluciones basadas en FPGA, pero su mayor obstáculo es que los tiempos de desarrollo son largos (y la verificación compleja).
- **Objetivo**: Lograr soluciones embebidas con aceleración por hardware que cumplan con los requerimentos de tiempo real, pero con tiempos de desarrollo similares a los de proyectos de software.
- **A continuación**:
  - Presentamos una *metodología de co-diseño para sistemas embebidos centrados en procesador con aceleración por hardware usando FPGA*.
  - La aplicamos a un par de ejemplos para resolver problemas reales en una arena.

Metodología de co-diseño

- **Motivación**: muchas aplicaciones embebidas necesita tiempo real y al mismo tiempo tienen restricciones de consumo, tamaño y peso. Para estos desarrollos son comunes soluciones basadas en FPGA, pero su mayor obstáculo es que los tiempos de desarrollo son largos (y la verificación compleja).
- **Objetivo**: Lograr soluciones embebidas con aceleración por hardware que cumplan con los requerimentos de tiempo real, pero con tiempos de desarrollo similares a los de proyectos de software.
- **A continuación**:
  - Presentamos una *metodologia de co-diseño para sistemas embebidos centrados en procesador con aceleración por hardware usando FPGA*.
  - La aplicamos a un proceso de imágenes para localizar múltiples robots en una arena.

Metodología de co-diseño

- **Motivación**: muchas aplicaciones embebidas necesita tiempo real y al mismo tiempo tienen restricciones de consumo, tamaño y peso. Para estos desarrollos son comunes soluciones basadas en FPGA, pero su mayor obstáculo es que los tiempos de desarrollo son largos (y la verificación compleja).
- **Objetivo**: Lograr soluciones embebidas con aceleración por hardware que cumplan con los requerimentos de tiempo real, pero con tiempos de desarrollo similares a los de proyectos de software.
- **A continuación**:
  - Presentamos una *metodologia de co-diseño para sistemas embebidos centrados en procesador con aceleración por hardware usando FPGA*.
  - La aplicamos a un proceso de imágenes para localizar múltiples robots en una arena.

Metodología de co-diseño

- **Motivación**: muchas aplicaciones embebidas necesita tiempo real y al mismo tiempo tienen restricciones de consumo, tamaño y peso. Para estos desarrollos son comunes soluciones basadas en FPGA, pero su mayor obstáculo es que los tiempos de desarrollo son largos (y la verificación compleja).
- **Objetivo**: Lograr soluciones embebidas con aceleración por hardware que cumplan con los requerimentos de tiempo real, pero con tiempos de desarrollo similares a los de proyectos de software.
- **A continuación**:
  - Presentamos una *metodologia de co-diseño para sistemas embebidos centrados en procesador con aceleración por hardware usando FPGA*.
  - La aplicamos a un proceso de imágenes para localizar múltiples robots en una arena.

Metodología de co-diseño utilizada

- **Diseño** del sistema completo siguiendo un approach OOP, expresado en UML. (*Herramientas* Enterprise Architect)
- **Implementación y Test funcional** en C++ en un procesador de propósito general. (*Herramientas* Eclipse, gcc, gdb, valgrind.)
- **Partición HW/SW**
  - *Migrar la solución de SW al procesador embebido*. Para eso, caracterizar qué hardware es necesario para correr el procesador. Generar la plataforma de HW y de SW.
  - *Profiling* optimizaciones de soft + identificar qué metodos necesitan ser acelerados por HW.
  - *Herramientas* Xilinx Embedded Development Kit: XPS, SDK, gprof.
- **Traducción a HW**: cada módulo de hardware se implementa, testea e integra
  - *Herramientas de traducción semi-automática*. AutoESL, ROCCC, CatapultC.
  - *Guidelines de traducción* two-process. Proyecto de la ESA
  - *Herramientas* Xilinx EDK, ISE, Simuladores.

Metodología de co-diseño utilizada

- **Diseño** del sistema completo siguiendo un approach OOP, expresado en UML. (*Herramientas* Enterprise Architect)
- **Implementación y Test funcional** en C++ en un procesador de propósito general. (*Herramientas* Eclipse, gcc, gdb, valgrind.)
- **Partición HW/SW**
  - *Migrar la solución de SW al procesador embebido*. Para eso, caracterizar qué hardware es necesario para correr el procesador. Generar la plataforma de HW y de SW.
  - *Profiling* optimizaciones de soft + identificar qué metodos necesitan ser acelerados por HW.
  - *Herramientas* Xilinx Embedded Development Kit: XPS, SDK, gprof.
- **Traducción a HW**: cada módulo de hardware se implementa, testea e integra
  - *Herramientas de traducción semi-automática*. AutoESL, ROCCC, CatapultC.
  - *Guidelines de traducción* two-process. Proyecto de la ESA
  - *Herramientas* Xilinx EDK, ISE, Simuladores.

Metodología de co-diseño utilizada

- **Diseño** del sistema completo siguiendo un approach OOP, expresado en UML. (*Herramientas* Enterprise Architect)
- **Implementación y Test funcional** en C++ en un procesador de propósito general. (*Herramientas* Eclipse, gcc, gdb, valgrind.)
- **Partición HW/SW**
  - *Migrar la solución de SW al procesador embebido*. Para eso, caracterizar qué hardware es necesario para correr el procesador. Generar la plataforma de HW y de SW.
  - *Profiling* optimizaciones de soft + identificar qué metodos necesitan ser acelerados por HW.
  - *Herramientas* Xilinx Embedded Development Kit: XPS, SDK, gprof.
- **Traducción a HW**: cada módulo de hardware se implementa, testea e integra
  - *Herramientas de traducción semi-automática*. AutoESL, ROCCC, CatapultC.
  - *Guidelines de traducción* two-process. Proyecto de la ESA
  - *Herramientas* Xilinx EDK, ISE, Simuladores.

Metodología de co-diseño utilizada

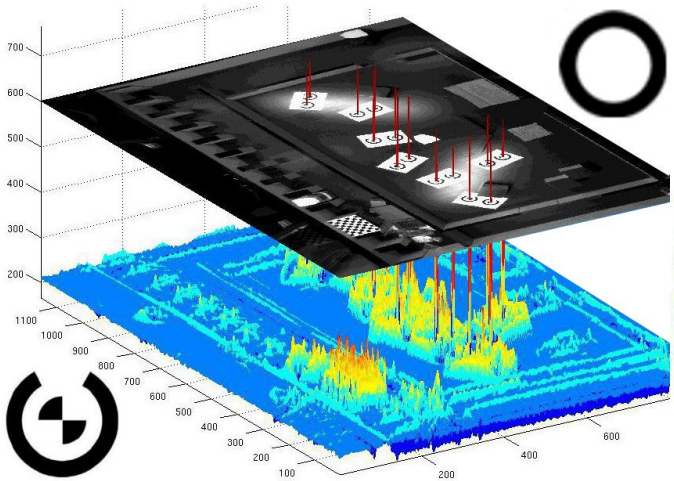
- **Diseño** del sistema completo siguiendo un approach OOP, expresado en UML. (*Herramientas* Enterprise Architect)
- **Implementación y Test funcional** en C++ en un procesador de propósito general. (*Herramientas* Eclipse, gcc, gdb, valgrind.)
- **Partición HW/SW**
  - *Migrar la solución de SW al procesador embebido*. Para eso, caracterizar qué hardware es necesario para correr el procesador. Generar la plataforma de HW y de SW.
  - *Profiling* optimizaciones de soft + identificar qué metodos necesitan ser acelerados por HW.
  - *Herramientas* Xilinx Embedded Development Kit: XPS, SDK, gprof.
- **Traducción a HW**: cada módulo de hardware se implementa, testea e integra
  - *Herramientas de traducción semi-automática*. AutoESL, ROCCC, CatapultC.
  - *Guidelines de traducción* two-process. Proyecto de la ESA
  - *Herramientas* Xilinx EDK, ISE, Simuladores.

Metodología de co-diseño utilizada

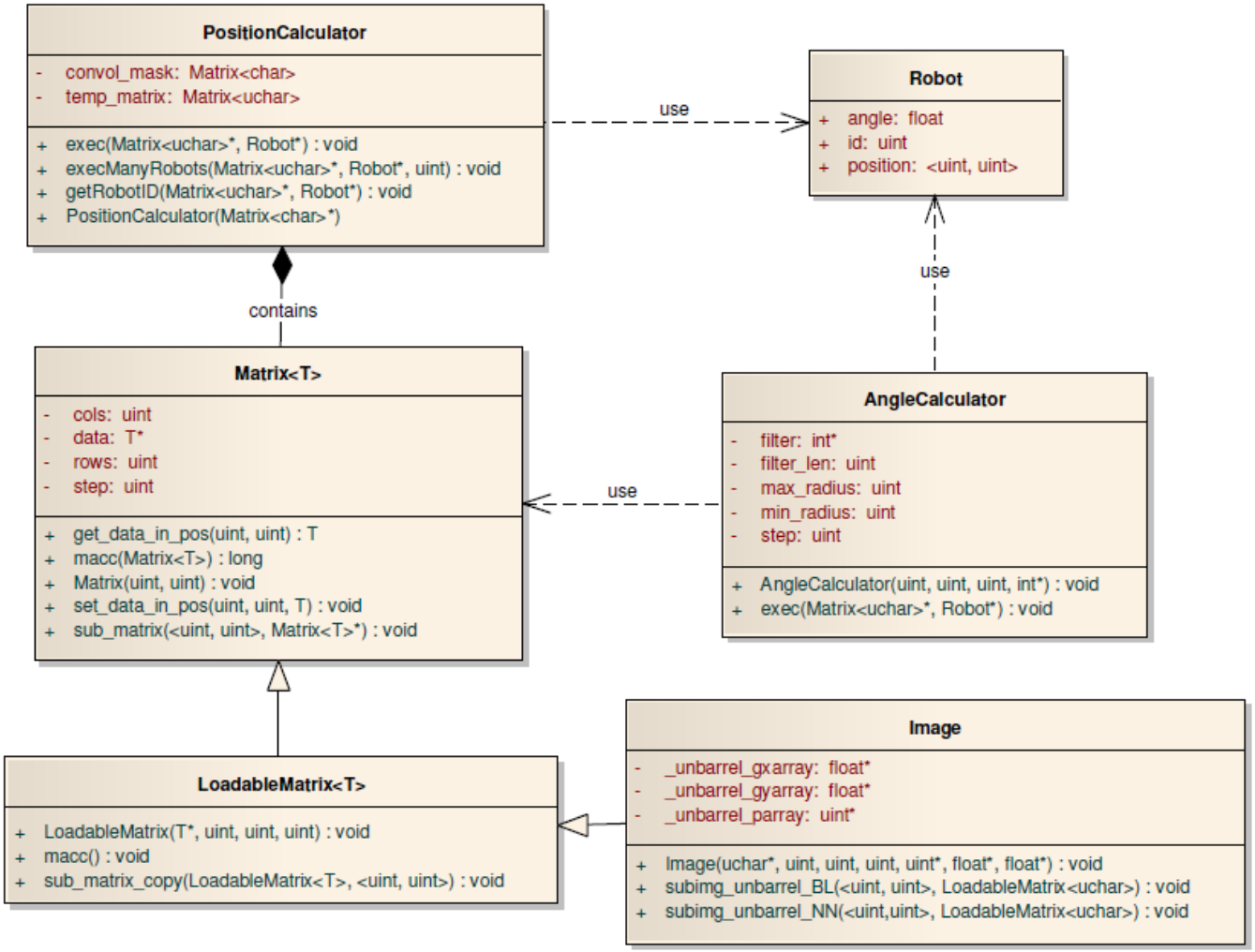
- **Diseño** del sistema completo siguiendo un approach OOP, expresado en UML. (*Herramientas* Enterprise Architect)
- **Implementación y Test funcional** en C++ en un procesador de propósito general. (*Herramientas* Eclipse, gcc, gdb, valgrind.)
- **Partición HW/SW**
  - *Migrar la solución de SW al procesador embebido*. Para eso, caracterizar qué hardware es necesario para correr el procesador. Generar la plataforma de HW y de SW.
  - *Profiling* optimizaciones de soft + identificar qué metodos necesitan ser acelerados por HW.
  - *Herramientas* Xilinx Embedded Development Kit: XPS, SDK, gprof.
- **Traducción a HW**: cada módulo de hardware se implementa, testea e integra
  - *Herramientas de traducción semi-automática*. AutoESL, ROCCC, CatapultC.
  - *Guidelines de traducción* two-process. Proyecto de la ESA
  - *Herramientas* Xilinx EDK, ISE, Simuladores.

Caso de Estudio: Localización de múltiples robots usando visión global

- SyRoTek es un laboratorio robótico virtual en desarrollo por el grupo de Robótica Móvil de la Universidad Técnica Checa.
- **Problema**: localizar la posición y el ángulo de cada robot en la arena en tiempo real.
- En conjunto con el laboratorio checo desarrollamos un algoritmo de reconocimiento de patrones y lo implementamos utilizando una FPGA con un procesador embebido.



Diseño e Implementación



Partición HW/SW

Table 1: Profiling results. All times in miliseconds.

	PPC405@300 MHz					i5@2.67GHz
	orig. code	cos_mask	fixed pt.	unbarrel_NI	angle_NI	all opt.
Matrix::macc	117.7	117.7	117.7	117.7	117.7	28.69
angleCalc::exec	246.9	48.3	17.7	17.7	7.3	0.84
Image::unbarrel	120.0	120.0	120.0	4.5	4.5	0.72
complete solution	*489.5*	*295.6*	*264.0*	*141.7*	*130.2*	*30.74*

- Plataforma: Avnet Virtex4-FX12 con PowerPC405 (que no tiene FPU).
- HW y SW necesario. El HW requerido es: Memorias: Flash, BRAMs, SDRAM, caches. Bus PLB para conectarlas al PPC. Modulos para debugging y profiling. Plataforma de Software: standalone.
- *Los unicos cambios en el SW son en el manejo de las imagenes e interfaces*

Implementación en Hardware y Resultados

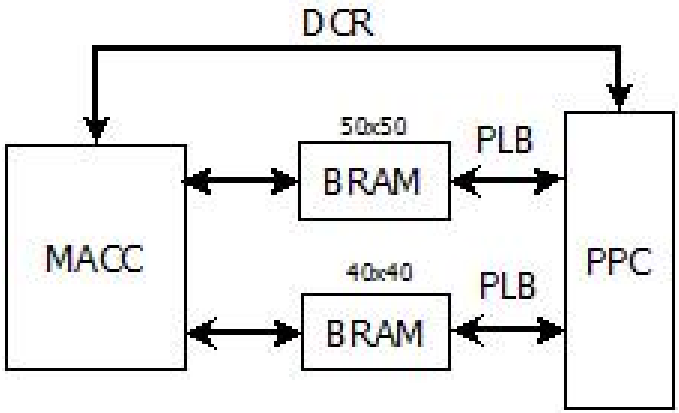


Table 2: Solution comparison in the Virtex4 FPGA

		all software	hard accelerated
Area	Slices	3,575	3,718
	BRAM	13	15
	DSP48	0	1
Time (ms)	Matrix::macc	117.7	22.4
	angleCalc::exec	246.9	7.3
	Image::unbarrel	120.0	4.5
	complete solution	*489.5*	*38.7*

	ROCCC	Two-Process
Slices	652	59
Slice Flip Flops	779	107
4 input LUTs	1099	112
FIFO16/RAMB16s	2	0
GCLKs	4	1
DSP48s	1	1
Max Freq Mhz	125.98	216.29
lines of code	1467	170

TABLE III
ROCCC AND TWO-PROCESS COMPARISON FOR VECTOR MACC

Conclusiones y trabajo en curso

La metodología:

- Reduce el esfuerzo de diseño subiendo el nivel de abstracción.
- Reduce el esfuerzo de codificar software reusando el golden model hecho en C++ como el código de la solución embebida final.
- Reduce el esfuerzo de codificación de HW ya que un buen diseño OOP permite encontrar exactamente qué métodos necesitan acelerarse por HW, previniendo traducciones de mas, y proponiendo el uso de guías o herramientas semi-automáticas para traducir
- Caso de Estudio: La solución embebida co-diseñada final procesa imágenes de 1600x1200 pixeles a 25 fps, logrando una aceleración de 12.6x de la solución de soft original.

Trabajo en curso: agregar paralelismo explícito en SW y mas unidades de HW.... Multithreading!

Materias

- Diseño de Sistemas con FPGA. Profesora: Dra. Patricia Borensztejn.
- Co-diseño Hardware Software utilizando FPGA. Profesora: Dra. Patricia Borensztejn.
- Visión en Robótica. Profesora: Dra. Marta Mejail
- Introducción a la Robótica ... en gestación!

Laboratorio de Robótica y Sistemas Embebidos

Pagina: http://robotica.dc.uba.ar
e-mail: robotica@dc.uba.ar
Integrantes:

- Profesores: Dra. Marta Mejail, Dra. Patricia Borensztejn, Dr. Julio Jacobo
- Estudiantes de Doctorado: Lic. Pablo de Cristóforis, Lic. Matias Nistche, Lic. Tahiú Piré, Lic. Sol Pedre.
- Estudiantes de grado: Sergio Gonzalez, Emiliano González, Facundo Pessacg, Javier Caccavelli, Carlos di Pietro, Thomas Fischer, Kevin Allekote, Maximiliano Urso.

Muchas gracias

Muchas gracias!