Celoxica

BAE SYSTEMS

MediusTech

Eagle Harbor Holdings, LLC

Exploiting FPGAs for Sensor Fusion

Steve Chappell,

Director Applications Engineering, Celoxica Ltd

2004 MAPLD International Conference
Ronald Reagan Building and International Trade Center
Washington, D.C.
September 8-10, 2004

Agenda

- Automotive Sensor Boresighting
- System Overview
  - Architecture
  - Sensors
  - FPGA System
- Design
  - Design Flow
  - Affine Transformations
  - 32bit Soft Core Processor
- Testing, Results and Summary

Automotive Sensor Boresighting

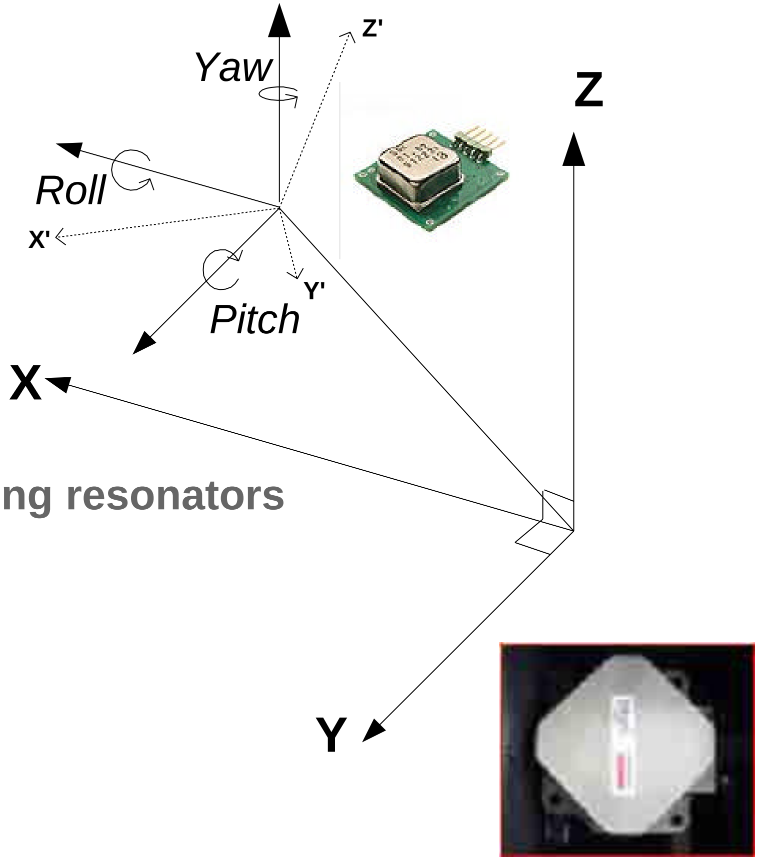
- Why?
  - Next generation automotive systems:
    - Lane Departure Warning, Collision Avoidance, Blind Spot Detection or Adaptive Cruise Control
    - Require “fusion” of data from sensors: video, radar, laser, global positioning systems and inertial measurement devices.
  - Sensors need accurate alignment with the vehicle platform
  - Significant cost/complexity for manufacturing process
  - Sensors may be displaced/offset after production
  - “Soft correction” of the sensor data desirable

RC200

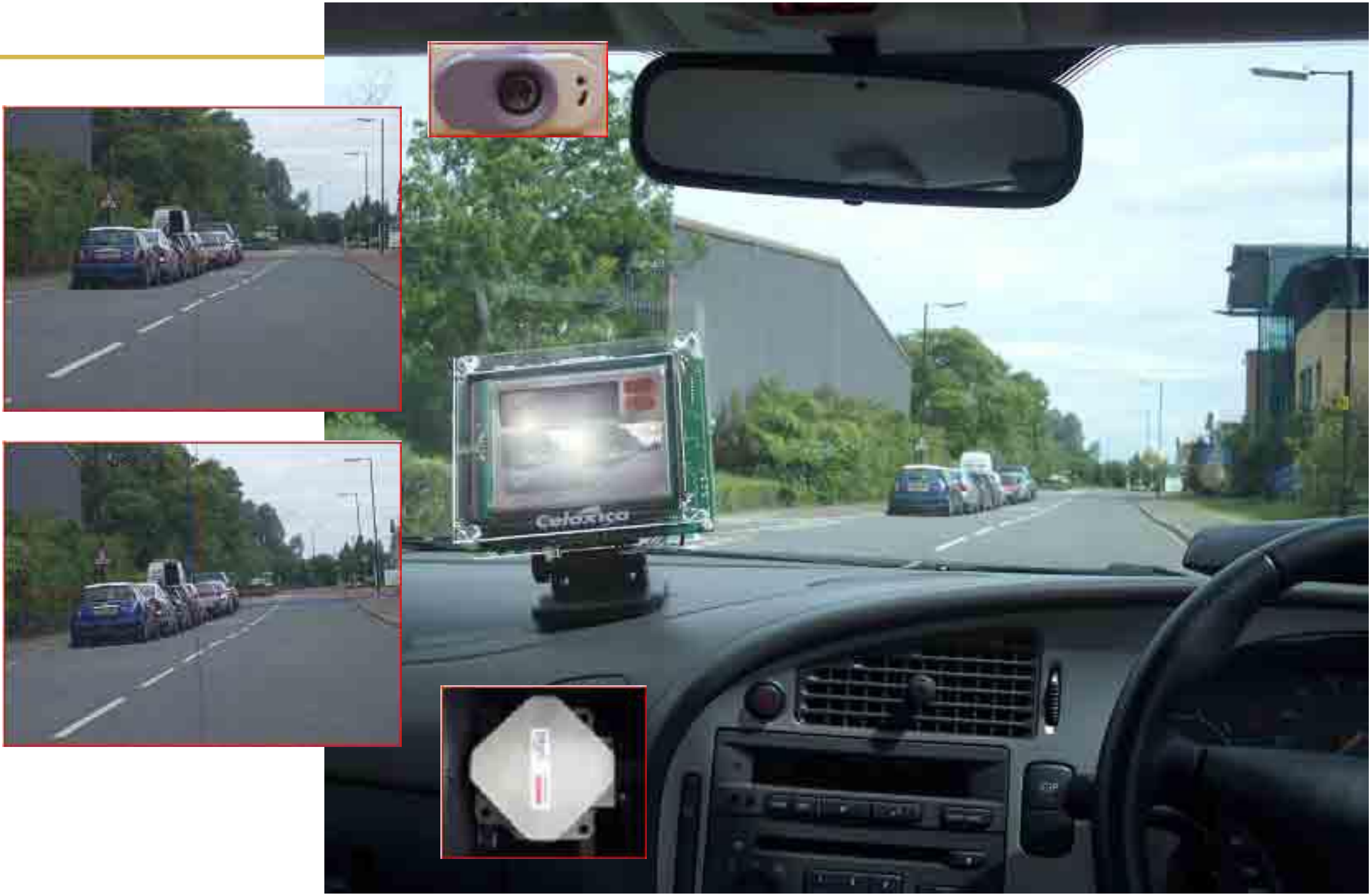
- Proposed solution
  - IMU from BAE Systems, (6-DOF)
  - Sensor Fusion Engine from Medius Inc
  - 3rd party accelerometers (off the shelf)
  - Celoxica Integration (RC200)

Sensor Reference Frames

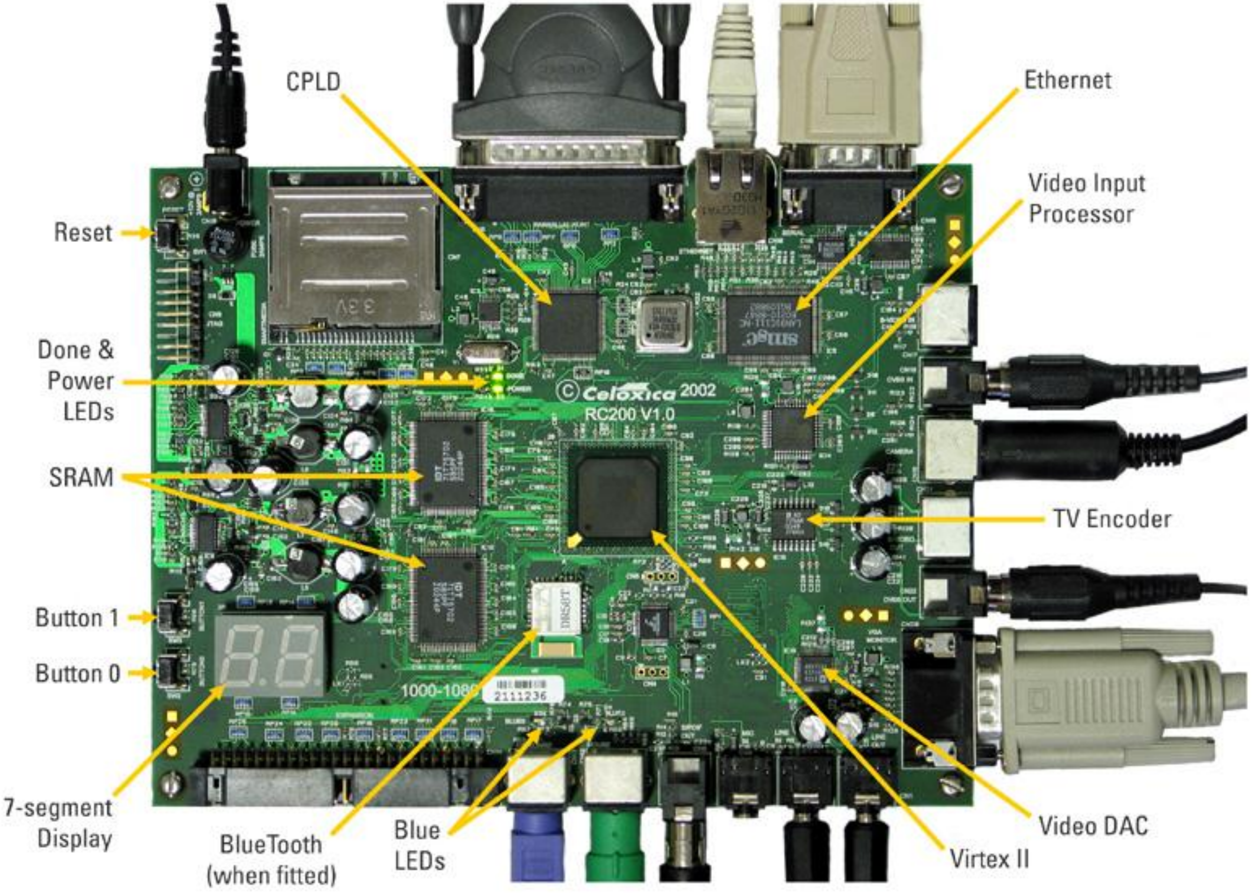
- Sensor Fusion Algorithm generates
  - Roll, Pitch, Yaw
  - Confidence Estimates
- MEMS Sensors
  - Acceleration
    - ☐ Capacitance of moving plates
  - Rotation
    - ☐ Coriolis force indiced vibrations in ring resonators
  - 6 DOF IMU (DMU)
  - 2 Axis Accelerometer (ACC)



In Car

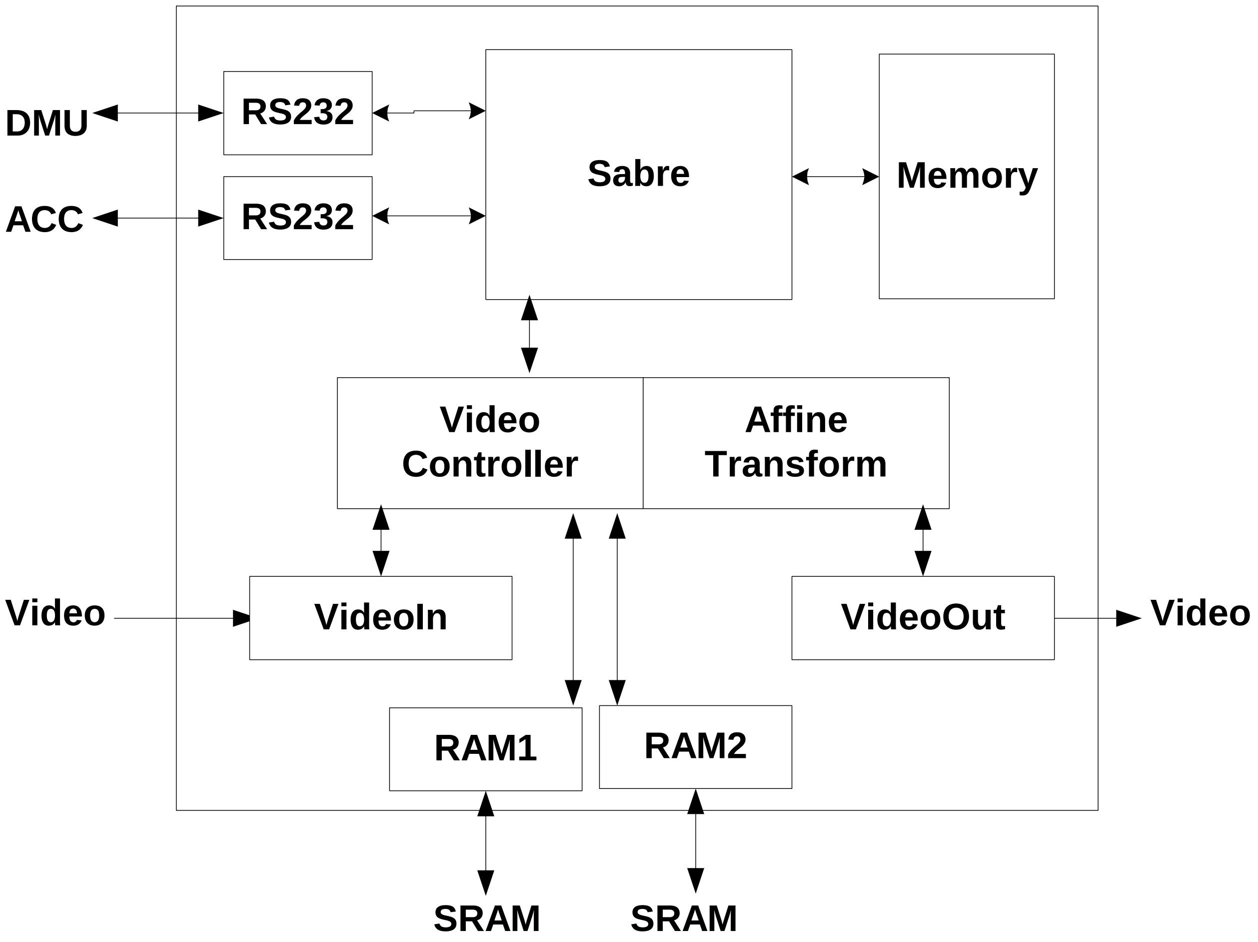


RC200 board

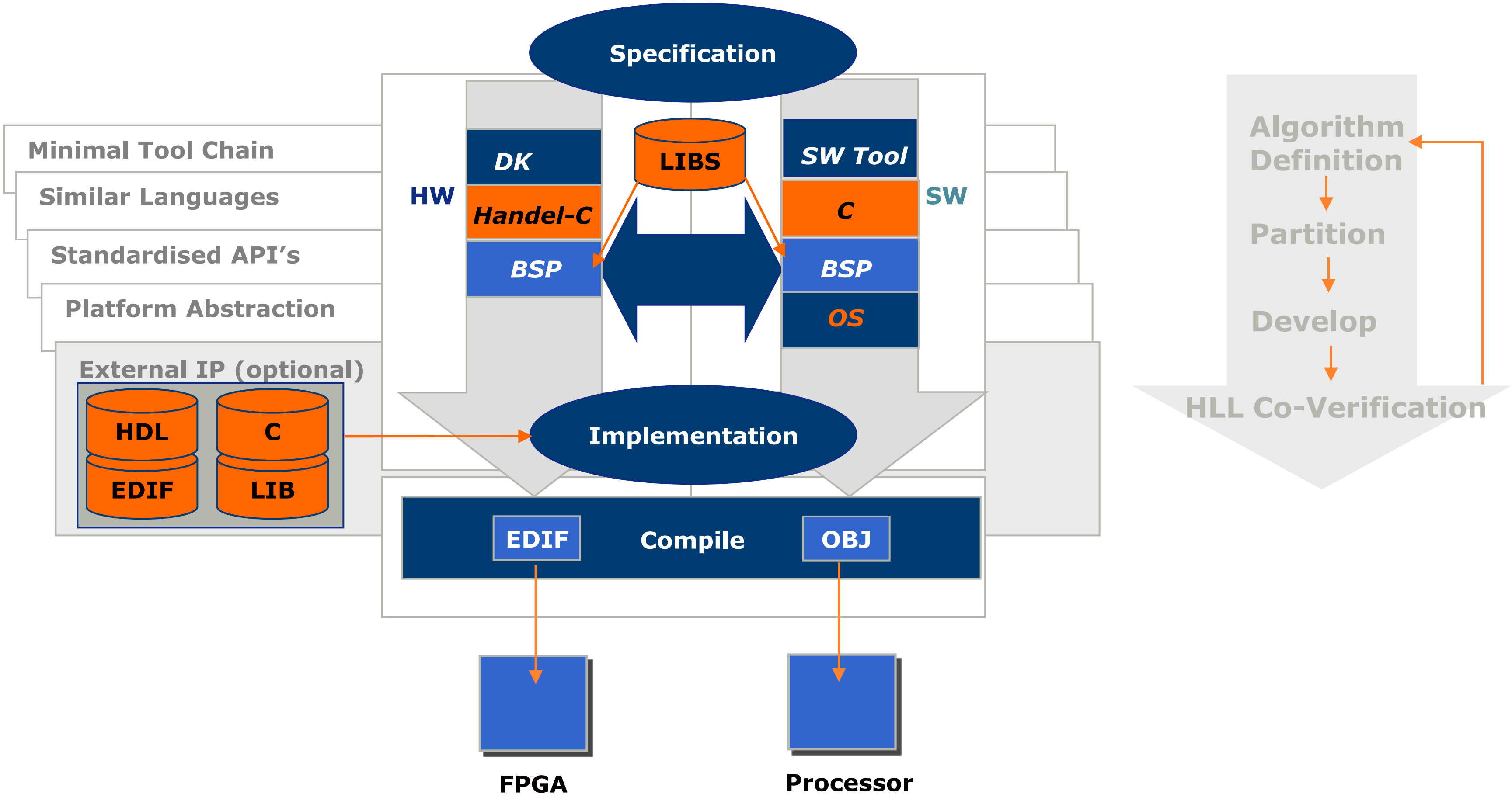


- Virtex II (1M Gate)
- Ethernet 10/100 MAC
- RS232
- Memory 8MB in 2 banks
- Smart media
- Video In/Out including T.V. out
- Audio In / Out
- Bluetooth
- TFT/Touch screen
- Expansion I/O

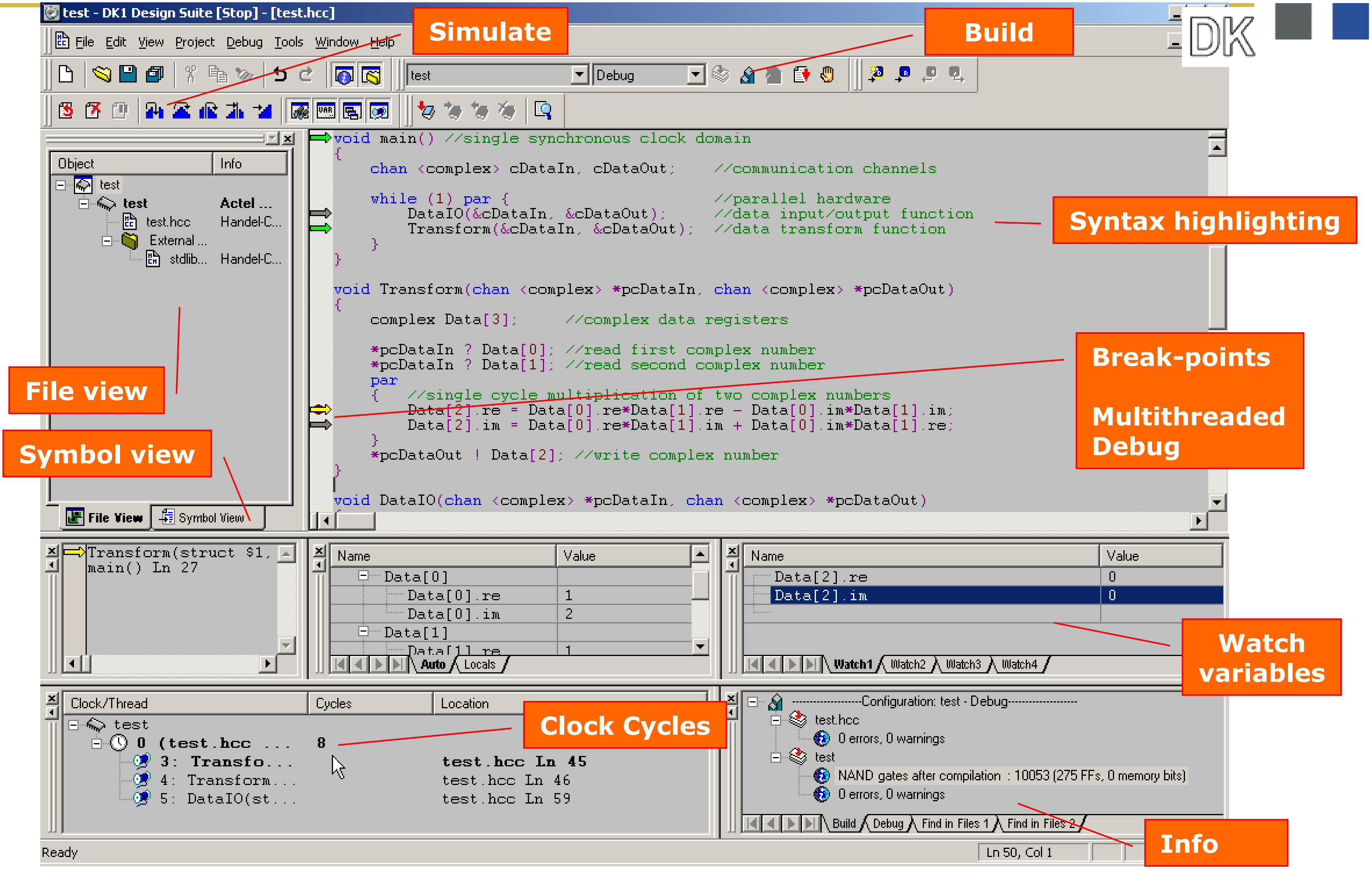
FPGA System


DMU
ACC
RS232
RS232
Sabre
Memory
Video Controller
Affine Transform
Video
VideoIn
VideoOut
Video
RAM1
RAM2
SRAM
SRAM

Design Flow


Specification
Minimal Tool Chain
Similar Languages
Standardised API's
Platform Abstraction
External IP (optional)
HDL
C
EDIF
LIB
HW
DK
Handel-C
BSP
LIBS
SW Tool
C
BSP
OS
SW
Implementation
EDIF
Compile
OBJ
FPGA
Processor
Algorithm Definition
Partition
Develop
HLL Co-Verification

DK - Software Compiled System Design



```
void main() //single synchronous clock domain
{
    chan <complex> cDataIn, cDataOut;     //communication channels

    while (1) par {                       //parallel hardware
        DataIO(&cDataIn, &cDataOut);      //data input/output function
        Transform(&cDataIn, &cDataOut);   //data transform function
    }
}

void Transform(chan <complex> *pcDataIn, chan <complex> *pcDataOut)
{
    complex Data[3];      //complex data registers

    *pcDataIn ? Data[0]; //read first complex number
    *pcDataIn ? Data[1]; //read second complex number
    par
    {   //single cycle multiplication of two complex numbers
        Data[2].re = Data[0].re*Data[1].re - Data[0].im*Data[1].im;
        Data[2].im = Data[0].re*Data[1].im + Data[0].im*Data[1].re;
    }
    *pcDataOut ! Data[2]; //write complex number
}

void DataIO(chan <complex> *pcDataIn, chan <complex> *pcDataOut)
```

```
Transform(struct $1,
main() Ln 27
```

Name	Value
Data[0]	
Data[0].re	1
Data[0].im	2
Data[1]	
Data[1].re	1

Name	Value
Data[2].re	0
Data[2].im	0

Clock/Thread	Cycles	Location
test		
0 (test.hcc ...	8	
3: Transfo...		test.hcc Ln 45
4: Transform...		test.hcc Ln 46
5: DataIO(st...		test.hcc Ln 59

```
-----------------Configuration: test - Debug-----------------
test.hcc
    0 errors, 0 warnings
test
    NAND gates after compilation : 10053 (275 FFs, 0 memory bits)
    0 errors, 0 warnings
```

Simulate

Build

Syntax highlighting

Break-points

Multithreaded Debug

File view

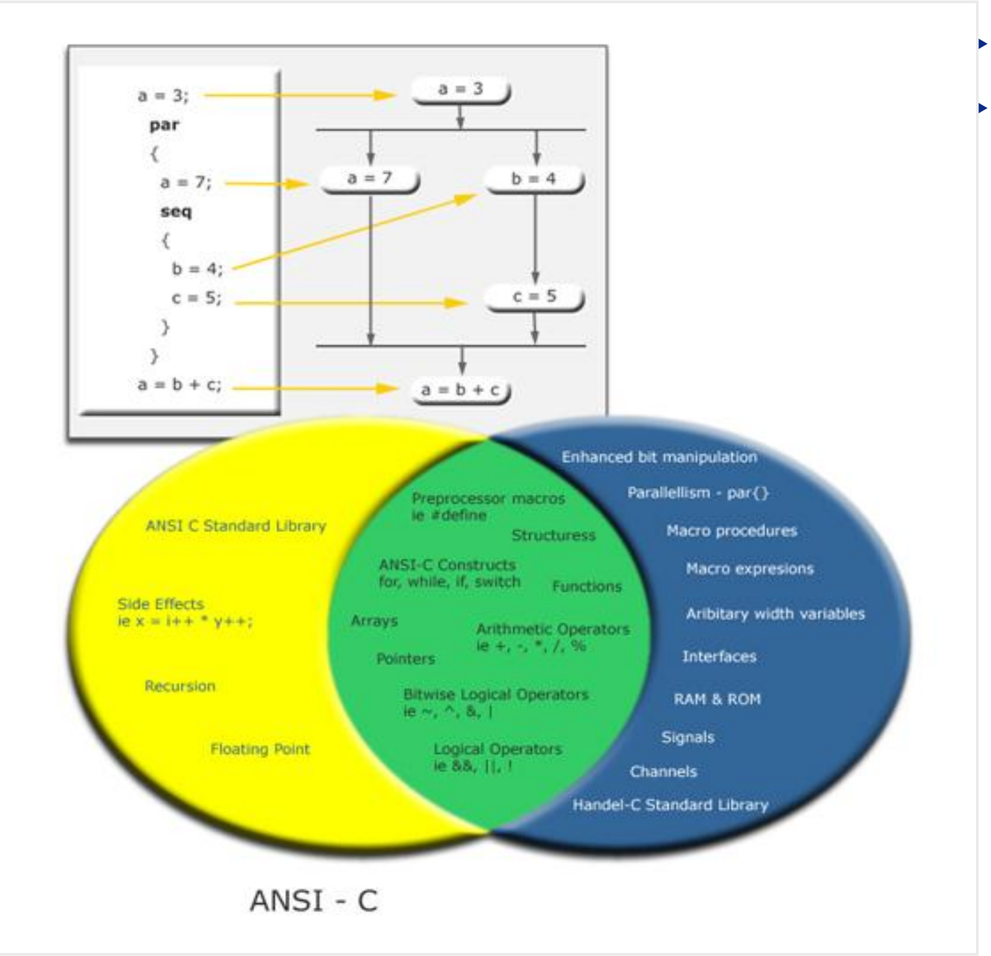
Symbol view

Watch variables

Clock Cycles

Info

Synthesizable ANSI-C for hardware



- **ANSI-C blocks are by default sequential**
- **`par{…}` executes statements in parallel**
- **par block completes when all statements complete**
  - **Time for block is time for longest statement**
  - **Can nest sequential blocks in par blocks**

Sequential Block

```
// 3 Clock Cycles
{
    a=1;
    b=2;
    c=3;
}
```

Parallel Block

```
// 1 Clock Cycle
   par{
          a=1;
          b=2;
          c=3;
    }
```

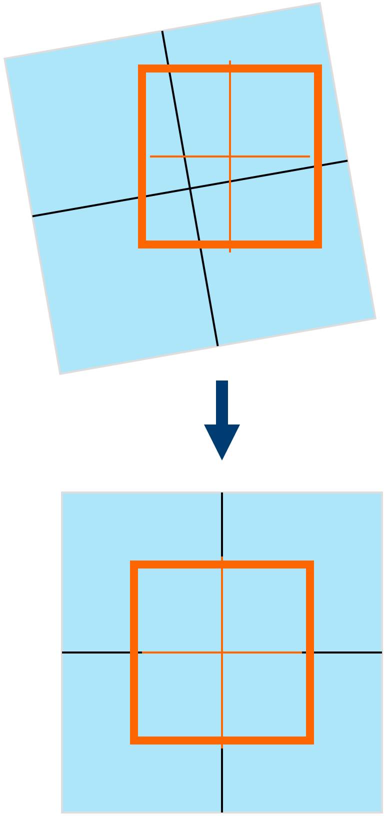
FPGA System Handel-C Code

```
void main (void)
{    …
    // Run everything
    par{ // Run Hardware Components
        SabreRun(&MyBus);     // 32-bit Processor
        RAMRun(RAM1);         // RAM Framebuffer
        RAMRun(RAM2);         // RAM Framebuffer
        VideoInRun (VideoIn); // Video Input Stream
        VideoOutRun(VideoOut);// Video Output Stream
        seq{
          par{ // Enables on Startup
               RAMEnable(RAM1);
               RAMEnable(RAM2);
               VideoInEnable (VideoIn);
               VideoOutEnable(VideoOut);
          }

          seq{ // main control loop
               WaitForSabre(); // Wait for Kalman Result
               par{
                   VideoInProcess (VideoIn); // Capture Video
                   VideoOutProcess(VideoOut); //Affine Transform/Output
               }
          }
        }
     }
}
```

Affine Transformations

- For the Transformation of Video



$$r' = Ar + B\,,$$

where A is the coordinate rotation matrix for angle θ about the z axis

$$A = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

and B is the linear transformation vector for corrections b_x and b_y in x and y respectively

$$B = \begin{pmatrix} b_x & 0 \\ 0 & b_y \end{pmatrix}$$

Affine Transformation Handel-C Code

```
static macro proc RotateCoordinates(theta,InX,InY,OutX,OutY)
{     …
     par{
          // Pipeline step 1
          GenerateSine(theta,Sin);
          GenerateCos(theta,Cos);

          //Pipeline step 2
          mapX = InX - CentreOfRotation[0];
          mapY = InY - CentreOfRotation[1];
          temp[0] = Int2fixed(mapX);
          temp[1] = Int2fixed(mapY);

          // Pipeline step 3
          FixedMult(temp[1], -Sin, temp[2]);
          FixedMult(temp[0], Cos, temp[3]);
          FixedMult(temp[0], Sin, temp[4]);
          FixedMult(temp[1], Cos, temp[5]);

          //Pipeline step 4
          mapXback = fixed2Int(temp[2]+temp[3]);
          mapYback = fixed2Int(temp[4]+temp[5]);

          //Pipeline step 5
          OutX = mapXback + CentreOfRotation[0];
          OutY = mapYback + CentreOfRotation[1];
     }
}
```

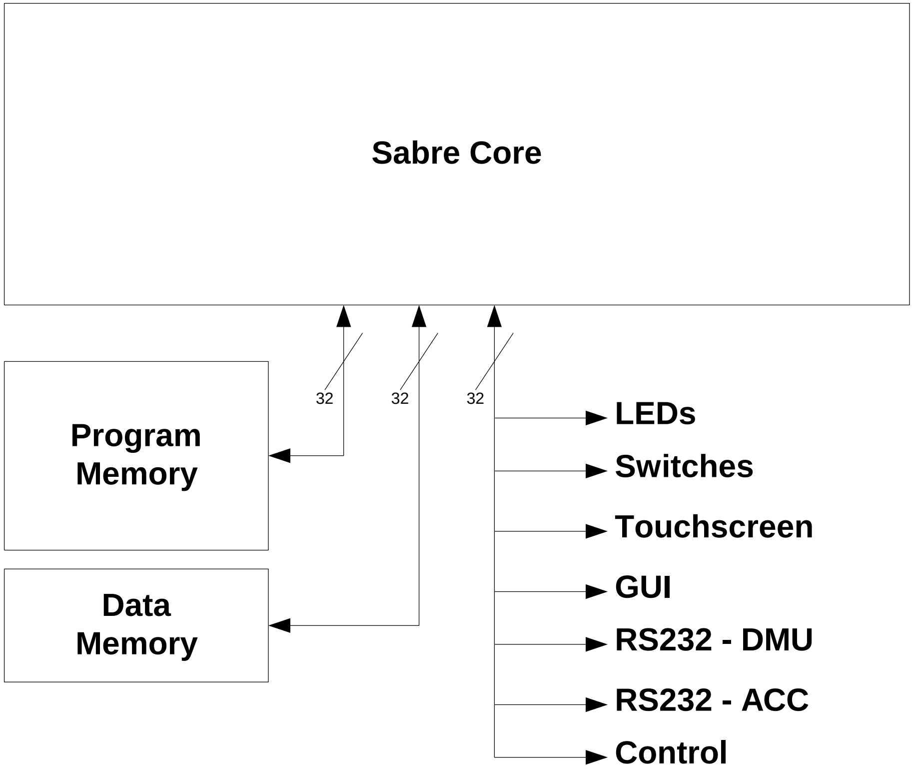
```
OutX = InX.cos(theta)-

       InY.sin(theta)

OutY = InY.cos(theta)+

       InX.sin(theta)
```

Sabre Processor System Architecture



Sabre Processor System Handel-C Code

```
void SabreRun (BusPtr)
{   …
   par{

    /* Core components */
    SabreRun           (BusPtr, DATA_MEMORY, PROGRAM_MEMORY);
    SabreBusRun        (BusPtr);
    SabreBusMemoryRun  (BusPtr, BUS_BASE_ADDRESS);

    /* User defined Peripherals */
    //LEDs
    SabreBusLEDsRun        (BusPtr, LEDS_BASE_ADDRESS);
    //Switches
    SabreBusSwitchesRun    (BusPtr, SWITCHES_BASE_ADDRESS);
    // TouchScreen
    SabreBusTouchScreenRun (BusPtr, TSCREEN_BASE_ADDRESS);
    // Graphical Output to Screen
    SabreGuiRun            (BusPtr, LINE_BASE_ADDRESS, …);
    // AMU Interface
    SabreRS232DMURun       (BusPtr, SERIAL1_BASE_ADDRESS);
    // DMU Interface
    SabreRS232ACCRun       (BusPtr, SERIAL2_BASE_ADDRESS);
    // Registers for Affine Transform
    SabreControlRun        (BusPtr, ANGLES_BASE_ADDRESS);
   }
}
```

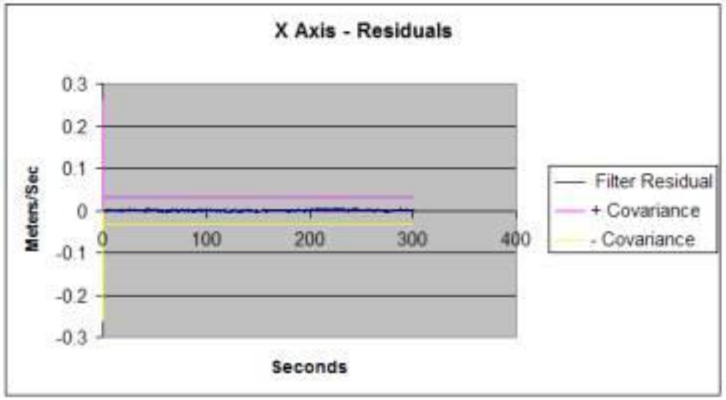
Testing and Results

- Static tests
    - Instruments calibrated using a level test platform.
    - Absolute misalignments measured directly using a laser attached to the boresighted sensor.
    - Static roll and yaw tests are more difficult to perform than the pitch tests since we must orientate the platform and use gravity to generate components of acceleration in the ACC and DMU accelerometers.

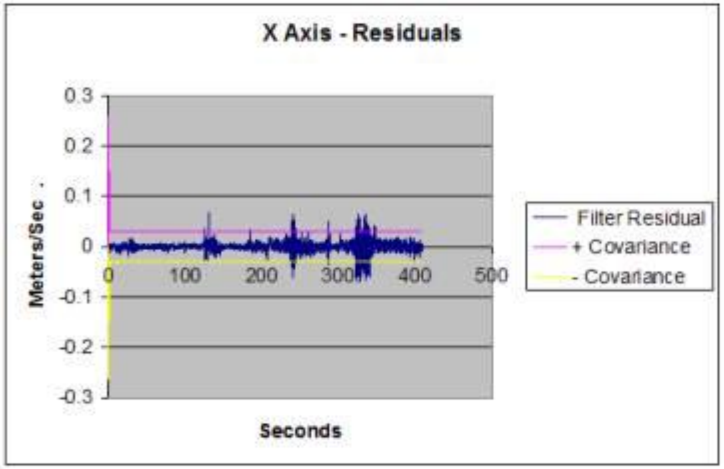
- Dynamic tests
    - Calibration
    - Misalign by a few degrees
    - Start Correction and run for 300s.

- Residuals used to tune the Kalman Filter (noise removal)
    - Static Tests : 0.003 to 0.01 m/s
    - Dynamic Tests: >0.015

Results: X Axis residuals

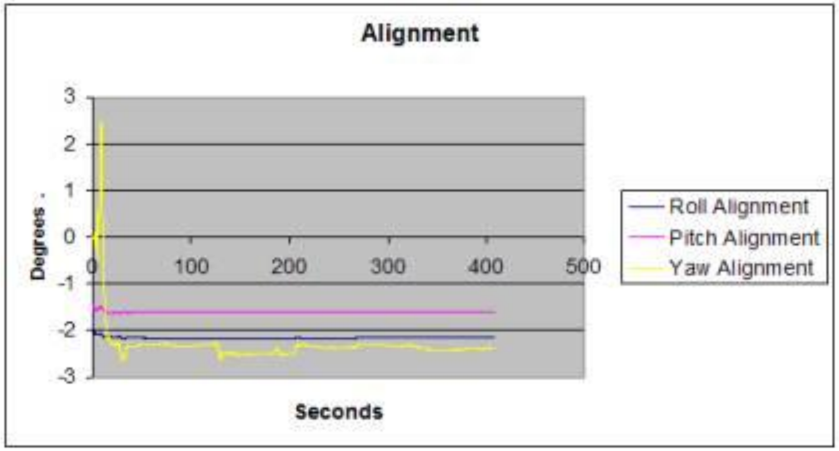


Static Test

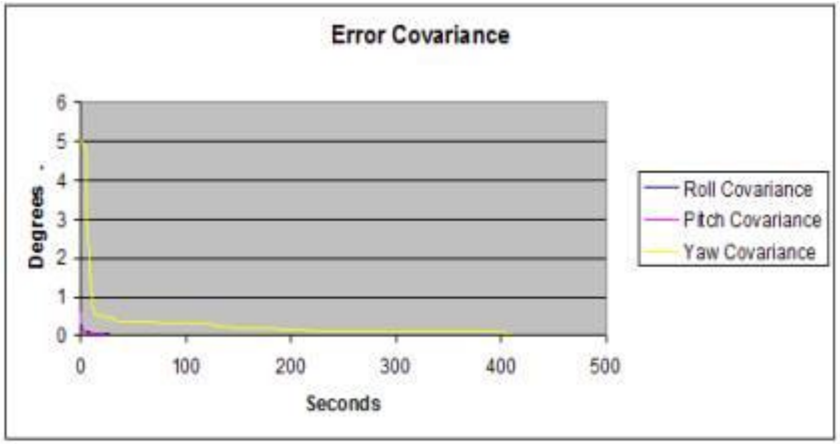


Dynamic Test

Results: Dynamic Test

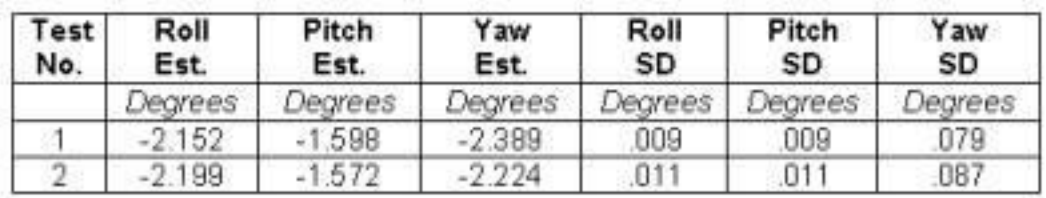


Alignment



Error

Results Summary



Test No.	Roll Est.	Pitch Est.	Yaw Est.	Roll SD	Pitch SD	Yaw SD
	Degrees	*Degrees*	*Degrees*	*Degrees*	*Degrees*	*Degrees*
1	-2.152	-1.598	-2.389	.009	.009	.079
2	-2.199	-1.572	-2.224	.011	.011	.087

Maneuver Performed	True Angle	Avg Estimated Angle	Filter Confidence
Pitch up	1 degree	.979 degrees	.011 degrees
Pitch down	1 degree	-1.002 degrees	.011 degrees
Roll left	2 degrees	-2.082 degrees	.011 degrees
Roll right	2 degrees	1.986 degrees	.011 degrees
Yaw left	1 degree	-1.005 degrees	.012 degrees
Yaw right	1 degree	1.073 degrees	.012 degrees

Summary

- Inexpensive accelerometers mounted on (or during assembly of) a sensor and an Inertial Measurement Unit (IMU) fixed to the vehicle can be used to compute the misalignment of the sensor to the IMU and thus vehicle.
- Sensor fusion techniques established in advanced aviation systems are applied to automotive vehicles with results exceeding typical industry requirements for sensor alignment.
- Manufacturing cost reduction and ROI
- COTS FPGA board and Celoxica DK Design Suite greatly simplified the task of creating a real-time proof of concept system.