Universidad Autónoma de Querétaro
Facultad de Informática
Maestría en Ingeniería de Software Distribuido

ANÁLISIS ESTÁTICO EN TIEMPO DE CODIFICACIÓN CON ECLIPSE CDT

TESIS

Que como parte de los requisitos para obtener el TÍTULO de:

Maestro en Ingeniería de Software Distribuido

Presenta:
Felipe Antonio Martínez Figueroa

Dirigido por:
Dr. Marco Antonio Aceves Fernández

SINODALES

Dr. Marco Antonio Aceves Fernández
Presidente — Firma

Dr. Saúl Tovar Arriaga
Secretario — Firma

Dr. Jesús Carlos Pedraza Ortega
Vocal — Firma

Dr. Juan Manuel Ramos Arreguín
Suplente — Firma

Dr. Efrén Gorrostieta Hurtado
Suplente — Firma

M.C. Ruth Angélica Rico Hernández
Director de la Facultad

Dr. Irineo Torres Pacheco
Director de Investigación y Posgrado

Centro Universitario
Santiago de Querétaro, Qro.
Enero de 2012
México

RESUMEN

El longevo lenguaje de programación C es ampliamente utilizado en nuestros días para la implantación de soluciones Linux y de sistemas embebidos compactas y eficientes. Sin embargo incluso para los ingenieros de software más experimentados es difícil evitar los modos de falla que tienen que ver con lo no especificado y lo dependiente de la implementación del lenguaje. En el presente trabajo de investigación se utiliza un subconjunto del lenguaje C basado en mediciones para el desarrollo y evaluación de una solución de análisis estático con bajo nivel de ruido basada en Eclipse CDT. La solución desarrollada muestra tasas de falsos positivos menores al 10%.

(**Palabras clave**: Eclipse CDT, Análisis estático, Lenguaje de programación C)

SUMMARY

The ancient C programming language is widely used nowadays to implement compact and efficient Linux and embedded software solutions. However even for the most experienced software engineers it is really hard to avoid the failure modes related to the unspecified and implementation dependent language features. On this investigation effort a measurement based ISO C language subset is employed to develop and evaluate a low noise Eclipse CDT static analysis solution. The solution implemented shows a false positive rate less than 10%.

(**Key words**: Eclipse CDT, Static analysis, C programming language)

A mis padres por estar siempre a mi lado apoyándome y por haberme inculcado los mejores valores, el amor al trabajo y la ética profesional

Felipe Antonio Martínez Figueroa

AGRADECIMIENTOS

Para la evaluación de la solución varios ingenieros de software del Centro Técnico Querétaro aportaron su opinión, análisis y sugerencias de mejora. A todos ellos mi más sincero agradecimiento y todo mi afecto.

Agradecimiento en especial al **M.C.C. Luis Ricardo Corral Velázquez** y al **Dr. Marco Antonio Aceves Fernández** por su ayuda en la coordinación y dirección del presente trabajo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS **8**
ÍNDICE DE TABLAS **9**

1. ANTECEDENTES **2**
1.1. Planteamiento del problema 2
1.2. Marco teórico 5
1.2.1. Verificación de Software 8
1.2.2. Pruebas de Software 9
1.2.3. Validación de software 10
1.2.4. Análisis estático de software 10
1.2.4.1. Analizadores estáticos de software 11
1.2.4.2. ISO/IEC 9899 13
1.2.4.3. MISRA-C 14
1.2.4.4. EC-A 16
1.2.5. Eclipse 18
1.2.5.1. Eclipse CDT 19
1.2.6. El lenguaje C 22

2. ESTADO DEL ARTE **27**
2.1. Análisis estático 27
2.1.1. Reglas de codificación 28
2.1.1.1. CERT C 29
2.1.1.2. HIS 29
2.1.1.3. JPL 29
2.1.1.4. IPA/SEC C 32
2.1.1.5. Netrino C 33
2.1.2. Análisis estático automático 33
2.1.2.1. Desarrollos y soluciones existentes 34

3. PLANTEAMIENTO DE LA SOLUCIÓN **38**
3.1. Definición de la investigación 38
3.1.1. Justificación e importancia del tema 39
3.1.2. Preguntas de investigación 40
3.1.3. Propuesta de solución 40
3.1.3.1. Hipótesis 40
3.1.3.2. Estándar de codificación base 40
3.1.3.3. Plataforma de desarrollo base 40
3.1.3.4. Ventajas y desventajas potenciales 41
3.1.4. Variables de investigación 41
3.1.5. Objetivos específicos 41
3.1.6. Alcance 42
3.2. Método de desarrollo 42

4. DESARROLLO **47**

4.1. Reglas de verificación de análisis estático ... 47
4.2. Extensión de la plataforma Eclipse CDT ... 50
4.2.1. Matriz de rastreabilidad ... 52
4.3. Especificación de la extensión ... 54
4.3.1. Requisitos generales ... 54
4.3.2. Instalación ... 55
4.3.3. Configuración por defecto ... 55
4.3.4. Activación y desactivación de reglas ... 56
4.3.5. Patrones de inclusión y exclusión ... 57
4.3.6. Invocación ... 58
4.3.7. Reglas incluidas ... 59
4.3.7.1. S_GLOB – "Expression required for return statement" ... 59
4.3.7.2. E_PREC – "Equality precedence issue checker" ... 59
4.3.7.3. E_SIDE, E_COMM – "Statement has no effect" ... 60
4.3.7.4. E_UNEG – "Negativity comparison checker" ... 60
4.3.7.5. E_CSIGN – "ImplicitConversionsChecker" ... 60
4.3.7.6. E_REACH – "Unreachable statements" ... 61
4.3.7.7. E_EXTR – "Externs in nested blocks" ... 61
4.3.7.8. E_FEQL ... 61
4.3.7.9. E_FLOOP ... 62
4.3.7.10. E_BITF1 ... 62
4.3.7.11. E_NARRW – "PointersBeingCastToIntegerChecker" ... 62
4.3.7.12. E_CASE1 – "Empty switch or without a default case" ... 63
4.4. Evaluación ... 63
4.4.1. Medición sobre casos de prueba ... 64
4.4.1.1. Proporción de falsos positivos ... 64
4.4.1.2. Proporción de falsos negativos ... 65
4.4.2. Medición en campo ... 66
4.4.2.1. Proporción de falsos positivos ... 66
4.4.2.2. Proporción de falsos negativos ... 67

5. RESULTADOS ... **71**
5.1. Resultados generales ... 71
5.2. Ventajas y desventajas de la solución ... 74
5.3. Conclusiones generales ... 74
5.4. Trabajo futuro ... 75

BIBLIOGRAFÍA ... **76**

GLOSARIO DE TERMINOS ... **81**
ANEXOS ... **82**
ANEXO A ... **83**
ANEXO B ... **85**
HOJA DE DATOS DEL MICRO-CONTROLADOR V850 DX4 ... **90**

ÍNDICE DE FIGURAS

Figura 1. Arquitectura CDT. 20
Figura 2. Arquitectura CODAN 22
Figura 3. Método de desarrollo de software. 43
Figura 4. Paquetes de la extensión a Eclipse CDT. 50
Figura 5. Diagrama de clases. 51
Figura 6. Ventana de configuración de la extensión. 56
Figura 7. Desactivación de reglas. 57
Figura 8. Ventana de especificación de patrones de inclusión o exclusión. 58
Figura 9. Experiencia previa de los evaluadores con herramientas de análisis estático. 71
Figura 10. Opinión de los evaluadores sobre la configuración de la herramienta. 72
Figura 11. Opinión de los evaluadores respecto a la invocación de la herramienta. 72
Figura 12. Opinión de los evaluadores respecto los errores reportados por la herramienta. 72
Figura 13. Problemas enfrentados durante la evaluación de la herramienta. 73
Figura 14. Sugerencias de mejora a la herramienta. 73
Figura 15. Experiencia en general de la evaluación de la herramienta. 74

ÍNDICE DE TABLAS

Tabla 1. Áreas de proceso del CMMI 6
Tabla 2. Procesos de ingeniería SPICE 7
Tabla 3. Procesos de soporte SPICE 7
Tabla 4. Revisión de puntos de análisis estático automático 12
Tabla 5. Reglas EC-A 17
Tabla 6. Reglas JPL 32
Tabla 7. Subconjunto de reglas EC-A 49
Tabla 8. Requisitos de la extensión a Eclipse CDT 55
Tabla 9. Proyectos de software seleccionados 64
Tabla 10. Proporción de falsos positivos sobre casos de prueba 65
Tabla 11. Proporción de falsos negativos sobre casos de prueba 66
Tabla 12. Proporción de falsos positivos del uso en campo de la herramienta 67
Tabla 13. Proporción de falsos positivos del uso en campo de la herramienta 68
Tabla 14. Resultados generales de falsos positivos y de falsos negativos 71
Tabla 15. Herramientas existentes de análisis estático 89

PRIMER CAPÍTULO
ANTECEDEDENTES

1. ANTECEDENTES

1.1. Planteamiento del problema

El lenguaje C continúa siendo el lenguaje dominante en el campo rápidamente creciente del desarrollo de software embebido. C fue el lenguaje más usado para el desarrollo de software embebido entre 1997 y 2009 [Barr, 2009]. Por otra parte el lenguaje es adecuado para la programación de sistemas de bajo nivel y con su estandarización está disponible en una enorme variedad de plataformas. Linux, el sistema operativo libre líder, está escrito en C [Scott, 2006].

C y C++ tienen la mayor parte de las características que un grupo de desarrollo de software pudiera desear y en las manos adecuadas pueden utilizarse para escribir código bien organizado, estructurado y expresivo pero en las manos equivocadas esta flexibilidad puede llevar a código perverso y extremadamente difícil de entender [Humphreys, 2009]. Pero más allá de lo obscuro y enmarañado que este código pudiera ser es código que podría fallar muy probablemente si consideramos también que existen muchas áreas del lenguaje donde lo dependiente de la implementación y lo no especificado reinan. También donde las conversiones de tipos pueden no ser lo que los desarrolladores esperan. Algunos ejemplos de puntos de falla son:

- Una función que debe regresar algo pero no provee ninguna expresión de retorno.
- El signo de una variable entera cuando este no se especifica.
- La precedencia de los operadores de bit respecto a los operadores de igualdad.
- La pérdida de información en conversiones implícitas o cuando se asigna una variable de mayor tamaño a una más pequeña.
- Cuando se intercalan y se hacen operaciones entre variables enteras sin signo y variables enteras signadas.
- La pérdida de precisión por redondeo en variables flotantes.
- Estructuras muti-decisión (“switch”) sin un caso por defecto (“default”)
- Nombres reusados en diferentes ámbitos, etc.

Para detectar modos de falla como los anteriores comúnmente el código es analizado estáticamente; esto es, se analiza su forma y su estructura pero no se ejecuta.

Este análisis puede ser llevado a cabo de forma manual o automática. Los compiladores por ejemplo realizan análisis estático automático verificando gramática, sintaxis, compatibilidad de tipos, semántica, etc., con el fin de validar el código antes de llegar a generar código máquina. Una forma típica de análisis estático manual son las revisiones de código conocidas en inglés como "Code reviews" (revisiones de código) pero tales revisiones dependen de la experiencia de los revisores, mientras las técnicas automáticas se benefician del conocimiento experto codificado en las herramientas [Evans y Larochelle, 2002].

Las técnicas y las herramientas de análisis estático son usadas para asegurar la forma adecuada de los productos de software. Esto puede lograrse verificando la adherencia a estándares de codificación y llevando a cabo una revisión de interfases y tipos [Perry, 1999].

Por medio de las herramientas de análisis estático es posible automatizar la verificación de software respecto a un estándar específico, pero su uso no se ha extendido primordialmente porque su relación de falsos positivos puede variar del 30% al 100% [Kremenek et. al., 2004], porque toman frecuentemente tiempos prohibitivamente largos para ejecutarse y porque no están bien integradas a los ambientes de desarrollo de software [Kleidermacher, 2008], [Evans y Larochelle, 2002].

Además del ruido producido por el análisis estático está el ruido del conjunto de reglas. Para muchas de las reglas de los estándares de codificación no hay evidencia sustancial que las ligue a fallas reales [Boogerd y Moonen, 2009a].

La taxonomía[1] de las reglas de codificación consiste de reglas A (reglas de estilo), reglas B.1 (folklor del lenguaje pero no basadas en una falla objetiva) y B.2 (evidencia de falla objetiva) [Hatton, 2003a].

Un buen estándar de codificación debería por lo tanto incluir exclusivamente reglas B.2 ya que otro tipo de reglas no serían más que distracción para los desarrolladores, carga innecesaria para el análisis estático y por lo tanto ruido. Dado un estándar de codificación

[1] Taxonomía: Ordenación jerarquizada y sistemática [RAE, 2010]

con reglas ruidosas el forzar su cumplimiento puede incrementar las fallas de software [Boogerd y Moonen, 2009a] en lugar de disminuirlas. Esto se debe a que cualquier corrección de falla en el software tiene una probabilidad no cero de introducir una nueva falla, y si esta probabilidad excede la reducción que se logra arreglando la violación, el resultado es un incremento en la probabilidad de fallas [Adams, 1984].

El globalmente aceptado subconjunto seguro del lenguaje C, MISRA-C, el cual es utilizado no sólo en la industria automotriz sino en otras industrias en las cuales hay componentes de seguridad crítica tiene una relación entre fallas genuinas y desviaciones no relacionadas a algún modo de falla de cerca de 1/50 en su versión 1998 [Hatton, 2003a]. En MISRA-C 2004 hay una reducción general en los falsos positivos del 29% pero esa relación es todavía no mayor a 1/35, por lo que se concluye que a menos que los conceptos sean refinados para que se encuentren ligados a modos de falla o excluidos mediante una política de desviación ambas versiones de MISRA-C son demasiados ruidosas para ser de uso real [Hatton, 2007].

Concluyendo dado el uso que tiene y que va a seguir teniendo el lenguaje C en las aplicaciones Linux y de software embebido ¿será posible llegar a una solución de análisis estático que se encuentre bien integrada al ambiente de desarrollo, que su ejecución sea expedita, que se base en reglas de codificación con evidencia de falla objetiva y que su uso en campo arroje una relación de falsos positivos considerablemente menor al 30%?

1.2. Marco teórico

El término ingeniería de software fue acuñado por el grupo NATO en 1967. La aseveración respecto a que construir software es similar a otras actividades ingenieriles se le atribuye a la conferencia de ingeniería de software NATO llevada a cabo en 1968 en Garmish, Alemania. Una conclusión de los conferencistas fue que la ingeniería de software debía usar las ideas y los paradigmas de las disciplinas ingenieriles para resolver lo que llamaron "la crisis del software", término usado para referirse a la pobre calidad del software y a las fechas de entrega y presupuestos no alcanzados [Schach, 2007].

La IEEE [IEEE, 2002] define de manera muy acertada a la ingeniería de software como la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software, y al estudio de tales enfoques. Sommerville [Sommerville, 2007] define a la ingeniería de software como la disciplina ingenieril involucrada con todos los aspectos de la producción de software desde etapas tempranas (especificación del sistema) hasta mantener el sistema ya en uso. En términos generales ingeniería de software quiere decir aplicar ingeniería al software, con el objetivo de cumplir con los requisitos del cliente en tiempo y forma, y a largo plazo reducir los costos y tiempos de desarrollo empleando las mejores prácticas de ingeniería, estandarizando los procesos y obteniendo valor de las lecciones aprendidas. Su misión principal es eliminar los problemas de la calidad del software mediante el establecimiento y estandarización de técnicas y modelos que promueven el orden y la administración eficaz de proyectos.

Por otro lado las iniciativas para mejorar el proceso de software no son más que esfuerzos coordinados de estandarización que han sido llevado a cabo por organizaciones dedicadas a este campo o por asociaciones dedicadas a mejorar la administración del desarrollo de software. Entre las iniciativas más importantes encontramos al CMM, el CMMI, la serie de estándares ISO 9000 e ISO/IEC 15504, mejor conocido como SPICE [Schach, 2007].

Las áreas de proceso del CMMI agrupadas por categoría son:

Categoría	**Área de proceso**
Administración del proceso	Definición del proceso organizacional Enfoque del proceso organizacional Entrenamiento organizacional Desempeño del proceso organizacional Organización y despliegue organizacional
Administración de proyectos	Planeación de proyectos Monitoreo y control de proyectos Administración de contratos de proveedores Administración integrada de proyectos Administración de riesgos Integración de equipos Administración cuantitativa de proyectos
Ingeniería	Administración de requisitos Desarrollo de requisitos Soluciones técnicas Integración del producto ***Verificación*** ***Validación***
Soporte	Administración de la configuración Administración de la calidad del proceso y del producto Medición y análisis Análisis de decisiones y resoluciones Ambiente organizacional para la integración Análisis de causa y resolución

Tabla 1. Áreas de proceso del CMMI

El PAM (“Process Assesment Model”) automotriz de SPICE [Automotive SIG, 2010] define los procesos primarios de ingeniería como:

Identificación del proceso	***Nombre del proceso***
ENG.1	Obtención y licitación de requisitos
ENG.2	Análisis de requisitos del sistema
ENG.3	Diseño arquitectónico del sistema
ENG.4	Análisis de requisitos de software
ENG.5	Diseño de software
ENG.6	Construcción de software
ENG.7	Pruebas de integración de software
ENG.8	Pruebas de software
ENG.9	Pruebas de integración del sistema
ENG.10	Pruebas del sistema

Tabla 2. Procesos de ingeniería SPICE

Los procesos de soporte como:

Identificación del proceso	***Nombre del proceso***
SUP.1	Aseguramiento de calidad
SUP.2	Verificación
SUP.4	Revisiones colegiadas
SUP.7	Documentación
SUP.8	Administración de la configuración
SUP.9	Administración de resolución de problemas
SUP.10	Administración de cambios

Tabla 3. Procesos de soporte SPICE

Sin importar mucho el modelo o estándar de desarrollo de software que se utilice las actividades que típicamente influyen más directamente en la calidad del software son:

- la verificación de software
- las pruebas de software
- la validación de software

Sin embargo a pesar de que estas tareas son llevadas a cabo extensivamente para controlar y evitar problemas de calidad de software existe cierta confusión generalizada respecto a su alcance y sus diferencias. A continuación revisaremos estos conceptos con el propósito de aclararlos y de manejar una definición común de ellos.

1.2.1. Verificación de Software

El CMMI [CMMI Product Team, 2006] define la verificación como asegurar que los productos de trabajo seleccionados cumplen con sus requisitos específicos. En sus notas introductorias agrega también que la verificación es un proceso incremental porque ocurre durante todo el desarrollo del producto y de los productos de trabajo, desde la verificación de los requisitos progresando hasta culminar con la verificación del producto completado. Por su parte SPICE [Automotive SIG, 2010] define como propósito del proceso de verificación confirmar que cada producto de trabajo de un proceso o proyecto refleja propiamente los requisitos especificados.

Obsérvese que tanto la definición del CMMI como la de SPICE son definiciones muy generales de verificación. El motivo es que la verificación es aplicada a una gran variedad de productos de trabajo y no solamente a productos de trabajo de software, como a la documentación, especificaciones de diseño, planes de desarrollo, planes de pruebas, etc.

No obstante lo generales de las definiciones anteriores la definición de verificación de software no difiere mucho de ellas. La verificación de software involucra revisar que el software cumple con su especificación [Sommerville, 2007]. Dicho de otra manera la verificación de software no es más que verificación aplicada al software para confirmar que este cumple con sus requerimientos. En caso de detectar desviaciones durante el proceso de verificación, estas son normalmente tomadas en cuenta y el software puede ser cambiado para cumplir con la especificación tanto funcional como no funcional. Ejemplos típicos de desviaciones detectadas durante el proceso de verificación de software son, errores de

lógica del programa, código fuera de los estándares de codificación, software incorrectamente identificado, etc.

La forma quizás más típica de verificación de software que no sean pruebas de software es la inspección de software ó las inspecciones de software las cuales se utilizan durante todas las etapas del proceso y suelen complementarse mediante análisis estático no sólo de código fuente. Ambas técnicas de verificación son técnicas de análisis estático sobre el cual hablaremos más adelante.

Los métodos formales de desarrollo de software son también una técnica de inspección de software. Puede pensarse en ellos como lo último en técnicas de verificación estática. Los métodos formales tienen que ver primordialmente con un análisis matemático de la especificación no obstante debido a que requieren un análisis muy detallado de la especificación y del programa su uso es caro y consume mucho tiempo por lo que se encuentran confinados a los sistemas de software donde la seguridad es crítica [Sommerville, 2007].

1.2.2. Pruebas de Software

Las pruebas de software consisten en ejecutar un programa o una parte de este con entradas y salidas conocidas, predecibles y observables, con el propósito de encontrar defectos o desviaciones de los requisitos [Laplante, 2007]. Debido a que el código es ejercitado las pruebas de software son una técnica de verificación dinámica que requieren de una versión ejecutable del software para poder llevarse a cabo.

A pesar de que las inspecciones de software son utilizadas ampliamente ahora, las pruebas de software serán siempre la técnica de verificación y validación más importante [Sommerville, 2007].

En las pruebas de defectos de software para cierto conjunto de entradas conocidas se tiene un conjunto de salidas predecibles que son derivadas directamente de los requisitos de software. Cuando las salidas se desvían de los valores o de las condiciones esperadas se dice que se ha encontrado un defecto en el software. La reparación de ó de los errores encontrados durante las pruebas de software contribuye directamente a mejorar la calidad del mismo. Algunos ejemplos de pruebas de defectos de software son:

- Las pruebas de unidad

- Las pruebas de integración
- Las pruebas de regresión
- Las pruebas de fuerza bruta, etc.

1.2.3. Validación de software

Las pruebas de validación de software tienen el propósito de mostrar que el software es lo que el cliente quiere y que cumple con sus requisitos. Como parte de las pruebas de validación se pueden utilizar datos estadísticos para probar el desempeño y la confiabilidad del programa y checar cómo funciona este bajo condiciones reales de operación [Sommerville, 2007].

Debido a que este tipo de pruebas de software son en parte una demostración final de que el software cumple su cometido se aplican sobre versiones completas del software en condiciones reales de carga, operación e interacción. Para un dispositivo que controla el disparo de las bolsas de aire de un automóvil por ejemplo, algunas pruebas de validación podrían consistir en colocar el dispositivo en su empaquetado final en el vehículo bajo condiciones de temperatura y humedad reales y en un choque provocado, verificar que efectivamente las bolsas de aire esperadas son activadas en el tiempo esperado.

Algunos ejemplos de pruebas de validación de software son:

- Las pruebas estadísticas
- Las pruebas de estrés
- Las pruebas de envejecimiento, etc.

1.2.4. Análisis estático de software

Las técnicas de verificación suelen dividirse en dos grandes vertientes. Técnicas de verificación dinámica y técnicas de verificación estática.

En las técnicas de verificación dinámica (pruebas de software por ejemplo) el programa en cuestión es ejecutado y tanto sus salidas como su comportamiento son observados y evaluados.

Las técnicas de análisis estático nacieron de la tecnología de los compiladores y muchos de ellos tienen funcionalidades de análisis estático disponibles (frecuentemente no utilizadas) [Marciniak, 1994]. En este tipo de técnicas (inspecciones de software por

ejemplo) la forma y estructura del programa en cuestión son analizadas y evaluadas pero sin llegar a la ejecución del mismo. Es por ello que se les llama técnicas estáticas ya que no requieren de una versión ejecutable del software.

Por su parte la IEEE define el análisis estático [IEEE, 2002] como el proceso de evaluar un sistema o componente basado en su forma, estructura, contenido o documentación. Aunque el análisis estático de software no reemplaza al análisis dinámico, es una revisión de calidad útil que se lleva a cabo normalmente antes de que el análisis dinámico comience. Como regla general mientras más crítico sea un sistema mayor esfuerzo debe ser dedicado a las técnicas de verificación estática.

No hay duda de que para lenguajes como C el análisis estático es una técnica efectiva para descubrir errores de software. Este compensa las debilidades de diseño del lenguaje de programación. Para cierto tipo de errores y ciertos patrones comunes en errores de programación típicos (variables sin inicializar, variables sin usar, variables fuera de rango, etc), es posible automatizar el proceso de verificación lo cual ha generado el desarrollo de analizadores estáticos automáticos para distintos lenguajes de programación [Sommerville, 2007].

1.2.4.1. Analizadores estáticos de software

Los analizadores estáticos son piezas de software que escanean el código fuente de un programa y detectan posibles fallas y desviaciones de acuerdo a un conjunto de reglas o criterios dependientes del lenguaje de programación y de los estándares organizacionales aplicables.

La intención del análisis estático automático es llamar la atención respecto a anomalías encontradas en el programa como, variables sin inicialización, variables sin usar, o datos fuera de rango [Sommerville, 2007]. Algunas verificaciones que pueden llevarse a cabo mediante análisis estático son:

Tipo de fallas	***Verificación de análisis estático***
Fallas de datos	Variables utilizadas antes de inicializarse Variables declaradas pero no utilizadas

	Variables con un valor asignado pero sin uso Violación de los límites de un arreglo Variables sin declaración
Fallas de control de programa	Código inalcanzable Saltos incondicionales dentro de ciclos
Fallas de entrada-salida	Tipo de parámetro erróneo Número de parámetros erróneo Resultado de funciones sin uso Funciones y métodos sin llamar
Fallas de administración de información	Apuntadores sin asignar o inicializar Aritmética de apuntadores

Tabla 4. Revisión de puntos de análisis estático automático.

Las herramientas de análisis estático llevan a cabo varios tipos de análisis. No todas las herramientas realizan todas las funciones listadas. La siguiente lista es sólo una selección de algunos tipos de análisis disponibles [Marciniak, 1994]:

- Análisis de complejidad: medición de complejidad ciclomática por ejemplo.
- Análisis de flujo de datos: detección de variables no inicializadas, etc.
- Análisis de control de flujo: código inalcanzable o ciclos infinitos, etc.
- Análisis de flujo de información: dependencias entradas-salidas, etc.

Los analizadores estáticos son particularmente valiosos cuando un lenguaje de programación como C es utilizado [Sommerville, 2007]. C no tiene reglas estrictas de tipos y la revisión que el compilador puede hacer es limitada. Por lo tanto, es fácil que los programadores cometan errores, y las herramientas de análisis estático pueden descubrir automáticamente algunas de las fallas de software resultantes.

Las herramientas de análisis estático producen falsos positivos y falsos negativos respecto a las reglas o criterios utilizados como referencia. Los falsos positivos son violaciones erróneamente reportadas mientras los falsos negativos son las violaciones que

no son detectadas por las herramientas. Los falsos positivos requieren de esfuerzo adicional para determinar su no aplicabilidad y los falsos negativos también, pero para llevar a cabo su detección.

Usar herramientas avanzadas de análisis estático está convirtiéndose rápidamente en una buena práctica. El error más barato es aquél que se encuentra lo antes posible. Debido a que el análisis estático es un proceso que ocurre al tiempo de compilación, éste puede encontrar errores incluso antes de que se termine el software. Esto es usualmente menos caro que si se encontraran esos errores escribiendo un caso de prueba o depurando [Anderson, 2009].

1.2.4.2. ISO/IEC 9899

En 1982 era claro que C necesitaba estandarización formal. La mejor aproximación a un estándar, la primera edición de “The C Programming Language”, no describía más el uso actual del lenguaje. Esta edición era también insuficientemente precisa en muchos detalles del lenguaje, particularmente como referencia para los compiladores. El uso incipiente de C en proyectos sujetos a contratos comerciales y gubernamentales, reveló que la aprobación y publicación de un estándar oficial era importante, así que ANSI estableció el comité X3J11 bajo la dirección de CBEMA (“Computer and Business Equipment Manufacturers' Association”) en el verano de 1983 con el objetivo de producir un estándar del lenguaje. El X3J11 produjo su reporte a finales de 1989 y subsecuentemente este estándar fue aceptado por ISO como ISO/IEC 9899-1990 [Ritchie, 1993].

El también llamado “C90” intentaba ser no sólo una especificación actual del lenguaje y una referencia para los desarrolladores de compiladores, sino que deseaba ser un estándar claro y consistente que promoviera la portabilidad entre distintos ambientes.

Entre los cambios introducidos por X3J11 encontramos:

- Tipos para los parámetros de funciones
- Calificadores de tipo “const” y “volatile”
- Promociones de tipos ligeramente distintas.
- Librería estándar

El núcleo del lenguaje C quedó prácticamente intacto respecto al proceso de estandarización [Ritchie, 1993] y el estándar emergió más como una mejor y más cuidadosa

codificación que como una nueva invención. Los cambios más importantes ocurrieron para el preprocesador y para la librería estándar.

El “C90” ahora caduco consistía de lo siguiente:

- ISO/IEC 9899:1990 (“Information Technology – Programming Language C”)
- ISO/IEC 9899 AM1 (una enmienda aprobada en 1995).
- ISO/IEC 9899 TCOR1 (una primer corrección técnica aprobada en 1995)
- ISO/IEC 9899 TCOR2 (una segunda corrección técnica publicada en 1996)

El estándar actual para el lenguaje de programación C es ISO/IEC 9899:1999 o simplemente “C99” el cual fue publicado en 1999. Este estándar, como cualquier estándar de este tipo, puede ser adquirido por un miembro de ISO o IEC. La última versión disponible del estándar se compone de:

- ISO/IEC 9899:1999
- ISO/IEC 9899:1999 Cor. 1:2001(E) (primer corrección técnica publicada en 2001).
- ISO/IEC 9899:1999 Cor. 2:2004(E) (segunda corrección técnica publicada en 2004).
- ISO/IEC 9899:1999 Cor. 3:2007(E) (tercera corrección técnica publicada en 2007).

1.2.4.3. MISRA-C

MISRA (“The Motor Industry Software Reliability Association”) comenzó a principios de los años noventa como un proyecto del programa de seguridad de tecnologías de información del gobierno de Reino Unido [MIRA, 2009a]. Este programa creó proyectos en un amplio rango de industrias involucradas con sistemas electrónicos de seguridad. El proyecto MISRA fue concebido para desarrollar directrices para la construcción de software embebido en sistemas electrónicos de vehículos terrestres. En noviembre de 1994 se publicó “Development guidelines for vehicle based software”. Este documento representaba un consenso industrial significativo y fue también la primera interpretación industrial automotriz del emergente estándar IEC 61508.

Una vez que la fundación oficial se disolvió, los miembros de MISRA decidieron continuar trabajando de forma conjunta sobre una base informal. Uno de los primeros resultados de esta colaboración fue MISRA C, el cual nació cuando Ford y Rover decidieron combinar sus esfuerzos individuales para crear un subconjunto del lenguaje C. Desde entonces, MISRA C se ha convertido en el estándar de facto para la programación embebida en C en la mayoría de industrias relacionadas con la seguridad y es utilizado también para mejorar la calidad de software, incluso donde la seguridad no es la consideración primordial.

MISRA ha hecho mucho para promover los mejores lineamientos y prácticas para C y ahora para C++. En 1998, MISRA publicó su estándar de C para promover el uso seguro del lenguaje en la industria automotriz de Reino Unido, el cual ha sido actualizado y liberado como MISRA-C:2004. El globalmente aceptado subconjunto seguro del lenguaje C, aborda particularmente los temas realzados en el estándar ISO respecto a lo inespecífico e indefinido, y al comportamiento dependiente de la implementación. MISRA-C ha sido globalmente acogido por compañías como DENSO, Delphi, Paragon, Yaskawa, Ford, Continental, Actia, etc., como una base para motivar las buenas prácticas de programación [Humphreys, 2009].

Las directivas de MISRA han sido utilizadas extensivamente alrededor del mundo y no sólo en la industria automotriz sino en otras industrias, en las cuales hay componentes de seguridad crítica como en la aviación y en la industria médica. MISRA también ha sido traducido al japonés por un consorcio de miembros de la industria automotriz japonesa [Hatton, 2003a].

La cita de las reglas MISRA-C en una herramienta o en su documentación, o su utilización como base para la definición de un estándar interno, requiere del pago de una licencia por derechos de uso. Tomar MISRA-C como base para un estándar público de codificación o para un curso de entrenamiento que cite el texto de las reglas, requiere aprobación escrita por parte de MISRA y el cumplimiento de los términos que marcan las licencias MISRA-C para este tipo de aplicaciones [MIRA, 2009b]. Afortunadamente existen otros estándares como EC-A, que además de poder ser utilizados libremente, aportan beneficios adicionales como se explica a continuación.

1.2.4.4. EC-A

Aunque existe una gran necesidad por estandarizar el uso del lenguaje C, y aunque esta área ha sido explorada con anterioridad, el progreso ha sido ad hoc y ha sido llevado a cabo en la ausencia de mediciones de soporte [Hatton, 2003b].

En noviembre de 2003 Hatton define EC-A, un subconjunto de reglas de ISO C para evitar modos de falla conocidos, para los cuales hay mediciones de fallas publicadas. La idea central de este esfuerzo, fue descartar todas las reglas para las cuales no hay mediciones de soporte disponibles.

Lo que hace Hatton es relacionar las fallas (o posibles fallas) con la frecuencia con la que realmente fallan. Para este propósito, utiliza mediciones realizadas sobre la misma población en un intervalo de 4 años, sin importar que tan cruda sea la relación obtenida. Basta el hecho de que la falla (o falla potencial) haya efectivamente hecho fallar el software al menos una vez, para considerar su inclusión.

EC-A tiene a la enmienda C90 (ISO/IEC 9899 AM1) y a sus dos correcciones técnicas (ISO/IEC 9899 AM1 y ISO/IEC 9899 TCOR2) como sus estándares base, y donde la semántica es común con C99, se utiliza la redacción de este último. Las características principales de EC-A son:

- Cada regla se encuentra asociada a fallas reportadas que efectivamente han hecho fallar software.
- Cada regla cubre tantos modos de falla como es posible para reducir el número total de reglas.
- Cada regla es fácil de entender y tan clara como es posible.
- Cada regla es tan neutral como es posible para facilitar su aceptación.
- Tomando todas las reglas, se cubre la mayoría de fallas reportadas.

Las reglas EC-A no cubren el uso de la librería estándar de C. La razón es que las fallas basadas en la librería estándar tuvieron un impacto muy pequeño en los experimentos de medición. Por la misma razón, un número de reglas comúnmente citadas basadas en buenas prácticas como todas las funciones deben tener un solo punto de salida, o el uso del “goto”, fueron excluidas de este subconjunto ISO C.

Las reglas EC-A son:

Regla	***Texto***
S_GLOB	*"There shall be no dependence on any of the undefined features of ISO C"*
F_PROT	*"All function calls and definitions shall be preceded by a new-style function prototype"*
F_COMP	*"All function call arguments and function definition parameters shall be compatible with the corresponding arguments in the corresponding function declaration"*
E_PREC	*"A boolean-valued sub-expression shall not appear next to a bit operator, an assignment operator or a relational operator in a if or while expression"*
E_SIDE	*"Every expression statement shall have at least one side-effect for any execution path"*
E_COMM	*"The expression on the left hand side of a comma operator shall have at least one side-effect"*
E_UNEG	*"An unsigned expression shall not be compared for Negativity"*
E_CSIGN	*"No implicit conversion shall changed the signed nature of an object or reduce the number of bits in that object"*
E_REACH	*"Every non-null statement shall be reachable"*
E_HIDE	*"Local objects shall not hide objects with internal or external linkage"*
E_EXTR	*"The extern keyword shall not be used in a nested Block"*
E_FEQL	*"Floating point objects shall not be compared for equality or inequality"*
E_FLOOP	*"Floating point objects shall not be used in the initialisation, controlling or re-initialisation expressions of a for statement or the controlling expression of a while statement"*
E_BITF1	*"Every named bitfield of size 1 shall use the unsigned or signed keyword in its declaration"*
E_NARRW	*"No pointer shall be cast to a narrower integral Type"*
E_PCAST	*"Pointers shall not be cast to different pointer Types"*
E_CASE1	*"A switch statement shall have at least one 'case:' and shall have exactly one 'default:'"*
E_CFALL	*"No 'case:' or 'default:' shall be reachable from the previous 'case:' or 'default:' except by direct fall-through"*
P_ARGS2	*"No function-like macro shall use an argument more than once"*
P_PAREN	*"All macro arguments shall be parenthesized unless by doing so a syntax violation would be created"*

Tabla 5. Reglas EC-A

A diferencia de MISRA-C, EC-A no requiere de licencia o autorización alguna, además de que provee un conjunto de reglas que se encuentran respaldadas por mediciones de falla,

por lo que sólo aquellas reglas ligadas a evidencia de falla objetiva son incluidas. La primera parte favorece la utilización de EC-A tanto para el campo académico como para el campo industrial, mientras que la segunda reduce el ruido del estándar base de codificación.

1.2.5. Eclipse

El proyecto Eclipse fue creado originalmente por IBM en 2001 y soportado por un consorcio de vendedores de software (Borland, IBM, MERANT, Sistemas de Software QNX, Rational Software, RedHat, SuSE, TogetherSoft y Webgain). Para finales de 2003 el consorcio inicial ya se encontraba conformado por más de 80 miembros. En 2004 se creó la fundación Eclipse como una corporación no lucrativa independiente para que actuara como administradora de la comunidad Eclipse [The Eclipse foundation, 2011a].

Eclipse es una comunidad de software libre cuyos proyectos se enfocan en construir una plataforma de desarrollo extensible, ambientes de ejecución y marcos de trabajo para construir, liberar y administrar software a lo largo de todo el ciclo de desarrollo de software [The Eclipse foundation, 2011b].

El software Eclipse comenzó como un ambiente integrado de desarrollo o IDE para Java pero ha crecido enormemente. Los proyectos de Eclipse cubren ahora lenguajes estáticos y dinámicos, clientes, marcos de trabajo para servidores, modelado, negocios, embebidos, móviles, etc [The Eclipse foundation, 2011c].

Para los millones de desarrolladores, ingenieros y usuarios alrededor del mundo, Eclipse es una plataforma extensible para la integración de herramientas. Para los cientos de miles de clientes que utilizan la plataforma para desarrollar extensiones (“plug-ins”) o plataformas completas, Eclipse representa tecnología probada, confiable y escalable sobre la cual pueden diseñarse, desarrollarse y liberarse productos comerciales. Para los miles de estudiantes e investigadores, Eclipse representa una plataforma estable para la innovación, la libertad y la experimentación. Para todos estos individuos, grupos y organizaciones, Eclipse es una plataforma sin vendedor para la integración de herramientas, la cual es soportada por un diverso ecosistema [Clayber et. al., 2006].

Desde 2002, Eclipse ha sido reconocido mediante premios y menciones honoríficas, entre las cuales encontramos, el mejor producto de Java (Munich, Alemania, 2003), la mejor herramienta de desarrollo de aplicaciones (InfoWorld 2003), herramienta de

desarrollo del año (Developer.com 2005), mejor herramienta de software libre ("Software Development Magazine 2005"), etc [The Eclipse foundation, 2011d].

El campo del software embebido ha tenido tradicionalmente un mercado de herramientas muy fragmentado, pero en los últimos tres años el terreno de las herramientas embebidas ha cambiado dramáticamente. Eclipse ha permitido el acuerdo para trabajar en una infraestructura común y eso ha liberado recursos de tal manera que los proveedores se pueden enfocar en herramientas especializadas que añaden valor a sus clientes [Walters, 2005].

Los principales proveedores de herramientas para sistemas embebidos y vendedores de Linux soportan ahora Eclipse como ambiente de desarrollo. Entre estos proveedores encontramos a Wind River, Green Hills, LynuxWork, QNX, IBM, Intel, MontaVista, Novell SUSE, PalmSource, Tensilica, TI, TimeSys y Mentor Graphics [Whitbread, 2009].

1.2.5.1. Eclipse CDT

Entre la gran variedad de proyectos basados en la plataforma Eclipse, se encuentra uno específico para el desarrollo en C y C++. Este proyecto es conocido con el nombre de Eclipse CDT o simplemente CDT por sus siglas en inglés, que abrevian "Eclipse C/C++ development Tooling".

El CDT arrancó en julio de 2002 con el objetivo primordial de proveer un marco de trabajo para el desarrollo en C y C++. El CDT 1.0 fue liberado en diciembre de 2002 [Marineau-Mes, 2005].

El proyecto Eclipse CDT provee un IDE para C y C++ completamente funcional dentro de la plataforma Eclipse. Sus características incluyen creación de proyectos, compilación para varios conjuntos de herramientas, compilación por medio de make, navegación, jerarquía de tipos, árboles de llamadas, navegador de encabezados, navegador de definiciones de macros, editor de código con resaltado de sintaxis, agrupación de código, remanufacturado de código, generación de código, herramientas visuales de depuración, etc [The Eclipse foundation, 2011e].

Utilizando el CDT, es posible realizar introspección de código C o C++ [Recoskie et. al., 2007]. El conocimiento del código fuente es guardado en el DOM ("Document Object Model") del CDT, con el cual se construyen varios componentes de introspección:

- Scanner: reconoce y construye tokens para el parser.
- Parser: Reconoce la salida del scanner y construye ASTs (“Abstract syntax trees”).
- DOM ASTs (“Abstract syntax trees”): árboles que guardan información estructural detallada del código.
- Visitadores AST: visitan nodos en el AST para hacer minería de datos.
- Índex: provee la habilidad de ejecutar búsquedas rápidas por nombre de elemento (nombre de variable, nombre de tipo, etc).
- Visitadores de índex: visitan nodos en el índex para hacer minería de datos.

La arquitectura del parser CDT se muestra a continuación [Schorn, 2006]:

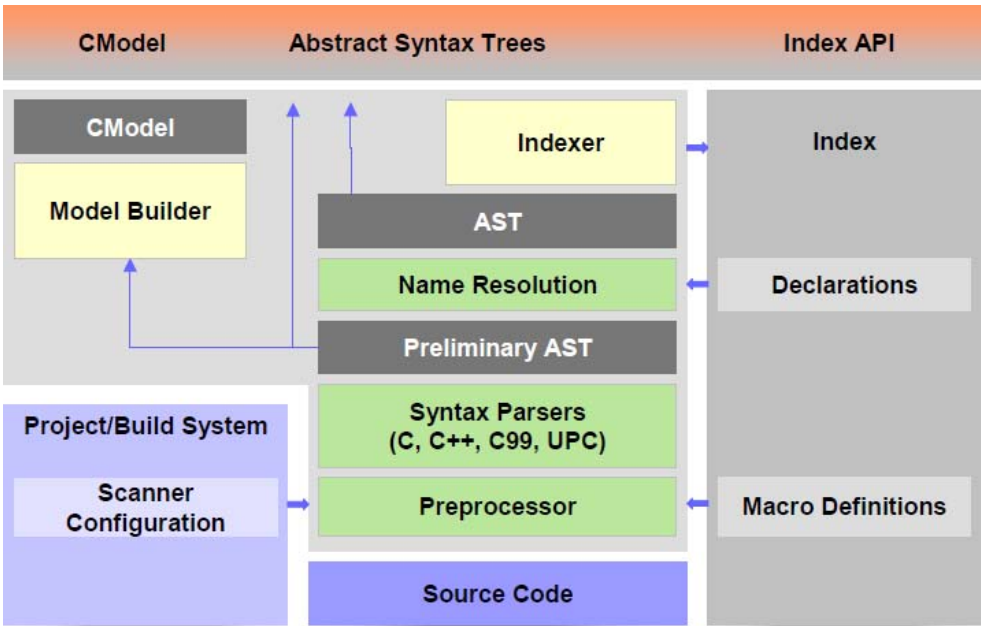

Figura 1. Arquitectura CDT.

En esta arquitectura el preprocesador lleva a cabo el análisis léxico del código fuente y la sustitución textual de acuerdo a los macros de configuración especificados para el proyecto y a aquéllos encontrados en el código fuente. Esta salida es tomada por el analizador sintáctico correspondiente de acuerdo al lenguaje de programación, y se construyen árboles de sintaxis abstracta preliminares, así como mapeos de referencias y de

símbolos que son guardados en el Índex. Con el índex y los árboles preliminares, se lleva a cabo la resolución de nombres, de este modo se tienen árboles de sintaxis abstracta totalmente resueltos, que son incluidos y jerarquizados en el modelo del proyecto. La capa superior es el API que permite acceder al modelo, a los árboles de sintaxis abstracta de cada módulo y al índex global.

Utilizando estos componentes el CDT permite:

- Buscar.
- Navegar.
- Proveer asistencia de código.
- Conocer quién llama a una función (“calling tree”) y qué funciones son llamadas dentro de una función (“call tree”).
- Conocer tipos básicos y tipos derivados, además de jerarquías de tipos.
- Obtener jerarquías y dependencias de encabezados.
- Resaltar sintaxis.
- Modificar código, etc.

Dentro del CDT, se está trabajando de forma incipiente, en la elaboración de un marco de trabajo para llevar a cabo análisis estático de C y C++, el cual es llamado CODAN (“Code Analysis”) [Laskavaia, 2010]. El objetivo de CODAN es crear componentes comunes y APIs compartidas como:

- Interfaz de usuario para controlar la habilitación y la especificación de parámetros para los problemas reportados.
- Diferentes modos de ejecución (en tiempo de codificación, por demanda y en tiempo de compilación)
- Información adicional de los problemas reportados.
- API para reportar problemas.
- Clases abstractas para los verificadores.
- Verificadores de muestra.
- Ambiente de pruebas basado en “JUnit”.

La arquitectura CODAN se muestra a continuación [Laskavaia, 2011]:

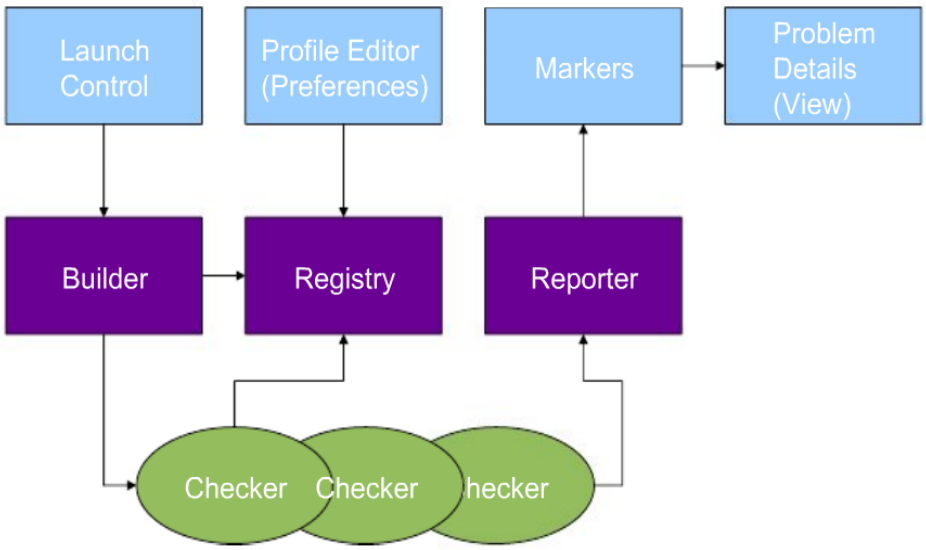

Figura 2. Arquitectura CODAN.

Se planea proveer otros modelos para la introspección de código además de los ya provistos por CDT (Índex y AST), como el modelo de control de flujo "Control Flow Graph", el modelo de control de datos "Data Flow Graph", el modelo de llamadas "Call Graph", entre otros.

Los modelos que ya se encuentran en uso actualmente en algunos verificadores de muestra son los ya maduros AST e Índex, y un incipiente modelo de control de flujo "Control Flow Graph".

1.2.6. El lenguaje C

El lenguaje C fue gestado entre 1969 y 1973, en paralelo con el desarrollo incipiente del sistema operativo Unix. Dennis Ritchie convirtió B en C entre 1971 y 1973, manteniendo la mayor parte de la sintaxis del lenguaje B, pero añadiendo tipos y muchos otros cambios más [Ritchie, 1993].

C es un lenguaje de propósito general con expresiones compactas, control de flujo, estructuras de datos y un conjunto basto de operadores. C no es un lenguaje de muy alto

nivel, ni muy extenso, y no está enfocado a un área específica de aplicación [Kernighan et. al., 1998].

Sin duda el éxito de Unix en sí mismo fue el factor más importante para que C fuera exitoso también [Ritchie, 1993]. En cambio, el uso de C en Unix y su portabilidad a una amplia variedad de máquinas, fue importante para el éxito del sistema. A pesar de algunos aspectos misteriosos para los novatos y ocasionalmente incluso para los expertos, C sigue siendo un lenguaje simple y pequeño, traducible mediante compiladores simples y pequeños también. Aprender a generar programas eficientes en tiempo de ejecución y espacio no es difícil al mismo tiempo que el lenguaje tiene el nivel suficiente de abstracción para que los programas sean portables.

A pesar de que C es un lenguaje pequeño y sencillo, es muy similar a una daga extremadamente cortante. Si se sabe cortar con ella se pueden hacer cortes finos y precisos, pero si por el contrario no se tiene cuidado o no se sabe cortar con ella, uno mismo puede herirse gravemente. Los cortes no deseados son los múltiples modos de falla nada fáciles de dominar que el lenguaje de programación C tiene implícitamente.

Algunos de los modos de falla de C son:

- Una función que se supone que regresa algo, pero no provee nada en la expresión de retorno.
- Operaciones combinando variables enteras con signo, y sin signo.
- Las conversiones implícitas.
- Campos de bits sin especificación de signo.
- Cuando en una unión se lee un miembro diferente al que se escribió.
- La precedencia de los operadores de bits respecto a los operadores de igualdad.
- El uso de apuntadores y variables sin inicializar, etc.
- Las conversiones explícitas de apuntadores de un tipo, a apuntadores de otro tipo.

En las manos de un programador experto, la mayoría de las limitaciones de C pueden ser evitadas [Jones, 2006], esto, por supuesto, es un problema para los programadores novatos. El problema es similar al dilema que enfrentan los padres cuando sus hijos aprenden a

manejar. La solución que la mayoría de los padres adoptan es darles a sus hijos vehículos grandes y lentos con muy buenos frenos. Unos pocos utilizan vehículos deportivos.

La generalidad del lenguaje C y su enfoque, permiten un amplio uso del mismo desde utilerías, librerías y programas completos para sistemas Windows, Unix, Linux, etc., hasta aplicaciones en sistemas embebidos de muy distintas índoles, en las cuales el software es ejecutado en un micro-controlador o en algún otro dispositivo similar programable.

Para los estudiantes de computación de hoy, aprender C es como tomar una clase de latín [Barr, 2009]. Pero C es cualquier cosa menos historia, y no es para nada un lenguaje muerto. C continúa siendo el lenguaje dominante en el campo rápidamente creciente del desarrollo de software embebido. C fue el lenguaje más usado para el desarrollo de software embebido entre 1997 y 2009. C ha dominado cuando múltiples lenguajes podrían ser escogidos (81% en promedio). Lo más importante es que los desarrolladores de sistemas embebidos no van a dejar de usar C en un futuro cercano, primero porque hay compiladores de C disponibles para la vasta mayoría de los CPU's, y segundo porque C ofrece la combinación justa de bajo nivel y alto nivel para programar a nivel procesador y a nivel manejador. Hasta que el uso de C comience a disminuir, las habilidades de programación en este lenguaje, seguirán siendo importantes.

C es adecuado para la programación de sistemas de bajo nivel. Gracias a su estandarización, el lenguaje está disponible en una enorme variedad de plataformas. Linux, el sistema operativo libre líder, está escrito en C. En 2005, C y sus descendientes rindieron cuenta del 60% de los proyectos de sourceforge.net [Scott, 2006].

SEGUNDO CAPÍTULO
ESTADO DEL ARTE

2. ESTADO DEL ARTE

En esta sección revisaremos el estado actual del análisis estático de código C, en lo que se refiere a los usos del análisis estático, reglas de codificación y herramientas de análisis estático automático.

2.1. Análisis estático

Debido a los límites tanto teóricos como prácticos del análisis estático, los diseñadores enfrentan un reto entre precisión y escalabilidad [Evans y Larochelle, 2002]. El análisis preciso de un programa en C, depende de varios problemas indecidibles, por lo que los diseñadores tienen que escoger entre limitar el tipo de chequeos de análisis estático a campos donde no se dependa de este tipo de problemas, o admitir cierto grado de imprecisión (falsos positivos y falsos negativos) en los resultados.

De cualquier forma, analizar código sin ejecutarlo ya sea mediante herramientas de análisis estático automático, o mediante inspecciones manuales del mismo es considerada una buena práctica de desarrollo de software. Aunque su objetivo primordial es detectar defectos de software antes de que el análisis dinámico ocurra y corregirlos, se ha encontrado empíricamente [Nagappan y Ball, 2005] que la densidad de defectos de análisis estático se correlaciona con la densidad de defectos previa a la liberación, por lo que la densidad de defectos de análisis estático puede utilizarse para predecir la densidad de defectos previa a la liberación, con un alto grado de sensibilidad.

El grado de predicción mejora cuando se emplea una combinación de análisis estático ligero (“lightweight static analysis”), el cual debería ser parte del proceso de desarrollo de aplicaciones sensibles a la seguridad [Evans y Larochelle, 2002], con análisis estático sobre código ya integrado.

Por otro lado, mediante el estudio de fallas de análisis estático, reportes de prueba y fallas reportadas por clientes, se ha encontrado [Zheng et. al., 2006] que el número de fallas de análisis estático puede ser efectivo para identificar módulos problemáticos de software, y que el análisis estático es complementario a otras técnicas de detección de fallas como inspecciones y pruebas para la producción de software de alta calidad.

En lo que concierne a la detección de vulnerabilidades de seguridad de software, conocer la herramienta específica de análisis estático parece ser más importante que tener experiencia en sí en seguridad [Baca et. al., 2009], mientras que los mejores resultados se obtienen cuando se cuenta con una combinación de experiencia en seguridad y cierta familiaridad con la herramienta específica. Finalmente, para que el análisis estático pueda ser un detector de fallas efectivo, la tasa de falsos positivos tiene que ser muy baja (20% o menor recomendado).

2.1.1. Reglas de codificación

Aunque MISRA-C es de los estándares de codificación más utilizados como base para la creación de estándares de codificación industriales y para el desarrollo de código C, los estudios disponibles sobre MISRA-C 1998 y 2004 no parecen soportar la utilización tan extendida y difundida de este conjunto de reglas de codificación.

En una comparación entre MISRA-C 1998 y MISRA-C 2004, aplicada a siete paquetes de software comerciales [Hatton, 2007], se concluye que al igual que MISRA-C 1998, la versión 2004 del estándar se queda también en la zona donde corregir las transgresiones puede incrementar el número total de fallas en vez de disminuirlas, debido al ruido de las reglas y al fenómeno de re-inyección de fallas [Adams, 1984]; en otras palabras el espíritu original de reducir el número total de fallas de software mediante el uso de MISRA-C en forma inalterada y sin una buena política de desviación, puede no sólo incumplirse sino de forma contraproducente empeorarse. Esto es consistente con estudios más recientes como [Boogerd y Moonen, 2009b] y [Boogerd y Moonen, 2009a], que han buscado la relación entre las transgresiones de reglas MISRA-C 2004 y la densidad de fallas (entre versiones de software, módulos, líneas de código específicas y fallas reales) y han encontrado sólo unas pocas reglas MISRA (10 reglas en el primer estudio, 9 reglas en el segundo) como predictoras de fallas significativas.

Debido a las deficiencias de MISRA-C y a los diferentes ámbitos y necesidades de estandarización, existen otras opciones de uso y aceptación industrial como:

- CERT C
- HIS
- JPL

- IPA/SEC C
- Netrino C

2.1.1.1. CERT C

Como parte del “Software Engineering Institute” existe una organización llamada CERT, cuya misión es estudiar las vulnerabilidades de seguridad de internet, investigar los cambios a largo plazo en los sistemas interconectados y desarrollar información y entrenamiento que ayude a mejorar la seguridad [Carnegie Mellon, 2011].

Parte de los esfuerzos de CERT en el área de aseguramiento de software (“Software Assurance”), específicamente en lo que se refiere a codificación segura (“Secure Coding”) han producido CERT C.

Mediante un proceso basado en una comunidad wiki [Carnegie Mellon, 2010], CERT ha coordinado el desarrollo de estándares de codificación segura, en el cual más de 500 contribuyentes y revisores han participado. La versión 1.0 del estándar de codificación seguro está disponible como un libro de Addison-Wesley y la versión 2.0 está todavía en desarrollo.

2.1.1.2. HIS

HIS (“Herstellerinitiative Software”) es un comité compuesto por Audi, BMW, Daimler, Porsche y Volkswagen [HIS, 2009] que tiene el objetivo de mejorar la competencia en los métodos de diseño de software y aseguramiento de calidad para las unidades de control que corren sobre microprocesadores. El objetivo principal es desarrollar y utilizar estándares conjuntos.

El grupo de trabajo HIS de pruebas de software liberó un subconjunto de las reglas de MISRA-C 1998 aplicables a este grupo [LDRA, 2011a]. Respecto a MISRA-C 2004 también incluyó reglas adicionales. En total el estándar HIS comprende 115 reglas de codificación, de las cuales 3 no se consideran analizables estáticamente.

2.1.1.3. JPL

Al igual que otras organizaciones que desarrollan código para aplicaciones de seguridad crítica, JPL (“Jet Propulsion Laboratory”) ha escogido a C como lenguaje de programación.

Para regular y estandarizar el uso del lenguaje C, JPL desarrollo un conjunto de 10 reglas que pueden hacer más confiable el análisis de componentes críticos de software [Holzmann, 2006]. Las reglas son:

Regla	***Contenido***	***Justificación***
Regla 1	*Restringe todo el código a estructuras de control simples. No utilices goto, setjmp o longjmp, o recursión directa o indirecta.*	*Las estructuras de control simples se traducen en mayores capacidades de análisis y frecuentemente en mayor claridad de código.*
Regla 2	*Todos los ciclos deben tener un límite superior fijo, de tal forma que estáticamente pueda determinarse que el ciclo no excede el límite máximo de iteraciones. Los calendarizadores de procesos están exentos.*	*La ausencia de recursión y los límites pre-establecidos en los ciclos evitan código que nunca termina.*
Regla 3	*No utilices memoria dinámica posterior a la inicialización.*	*Las funciones para reservar memoria como malloc y los recolectores de basura ("garbage collectors") frecuentemente tienen comportamientos impredecibles que pueden impactar significativamente el desempeño.*
Regla 4	*Ninguna función debe ser mayor a 60 líneas de código (lo que puede imprimirse en una hoja en formato estándar de una línea por enunciado "statement" y una línea por declaración).*	*Cada función debería ser la unidad lógica en el código, entendible y verificable como unidad. Las funciones extensas son frecuentemente un signo de código mal*

		estructurado.
Regla 5	*La densidad de aserciones del código debería promediar dos aserciones por función.*	*Las estadísticas indican que las pruebas unitarias encuentran frecuentemente al menos un defecto por cada 10 ó 100 líneas de código. La tasa de intercepción de defectos incrementa significativamente con una densidad de aserciones mayor.*
Regla 6	*Declara todos los objetos con el menor alcance posible.*	*Esta regla se basa en el principio de encapsulamiento de datos. Un menor número de asignaciones facilita el diagnóstico de problemas.*
Regla 7	*Una función que llama a otra función debe checar el valor de retorno cuando es distinto a void, y cada función llamada debe checar la validez de los parámetros provistos.*	*El valor de retorno de una función no debe ser ignorado, especialmente si la función debe propagar el valor de error en la jerarquía de llamadas.*
Regla 8	*El uso del pre-procesador debe limitarse a la inclusión de encabezados y a la definición de macros simples, además de que la compilación condicional deberá mantenerse al mínimo.*	*El pre-procesador de C es una herramienta poderosa de ofuscación que puede destruir la claridad del código.*
Regla 9	*El uso de apuntadores debe restringirse a:*	*Los apuntadores pueden*

	-apuntadores simples, no dobles ni más allá. *-operaciones de apuntadores escondidas en definiciones de macros o sinónimos en declaraciones de tipos ("typedefs").* *-los apuntadores a función no son permitidos.*	*hacer que el código sea difícil de seguir y de analizar. Por otro lado, los apuntadores a función deben utilizarse sólo si existen razones poderosas respaldándolos.*
Regla 10	*Todo el código debe compilarse desde el primer día con todos los warnings habilitados en el nivel de chequeo más estricto disponible. Todo el código debe compilar sin warnings y todo el código debe ser checado diariamente con al menos una herramienta de análisis estático poderosa y deberá pasar con cero warnings.*	*Actualmente existen varios analizadores estáticos extremadamente efectivos en el mercado, incluso algunos cuántos gratuitos, por lo que no hay excusa para no utilizarlos.*

Tabla 6. Reglas JPL

A diferencia de otras reglas de codificación con típicamente cientos de reglas, JPL provee un conjunto pequeño de reglas con la intención de que todas ellas sean lo suficientemente claras y específicas para implantarse y verificarse de forma manual o automática, al mismo tiempo que puedan lograrse efectos medibles en la confiabilidad y verificabilidad del software.

Estas reglas están siendo utilizadas de forma experimental en JPL para escribir software de operación crítica, encontrando beneficios en la seguridad.

2.1.1.4. IPA/SEC C

La guía de prácticas de codificación IPA/SEC C fue creada por el Centro de Ingeniería de Software de la Agencia Japonesa de Promoción de Tecnologías de Información [LDRA, 2011b]. Las prácticas fueron diseñadas para ayudar en la producción de código de alta calidad, sin importar las habilidades de los programadores.

La guía se basa en experiencia japonesa en el desarrollo de sistemas embebidos, combinada con MISRA-C 2004, los estándares de codificación y de estilo “Indian Hill” y los estándares de codificación de GNU.

Debido a que la guía se enfoca en cubrir atributos de calidad como confiabilidad, mantenibilidad, portabilidad, convenciones de nombres, estilo y eficiencia, en contraste con MISRA-C 2004 (el cual se enfoca principalmente en la confiabilidad), cubre un espectro más amplio. También es un complemento a la seguridad, y por lo tanto, a CERT C.

2.1.1.5. Netrino C

La compañía Netrino fue fundada en 1999 en Elkridge Maryland. Netrino produce una gran variedad de dispositivos embebidos, desde dispositivos médicos hasta cepillos de dientes electrónicos.

Netrino C es un estándar de codificación para C embebido que fue desarrollado por Netrino [Cheung, 2010] para minimizar los errores en el firmware mediante la creación de reglas que no sólo evitan los errores, sino que mejoran la mantenibilidad y la portabilidad del software embebido.

El estándar de codificación detalla principios de programación de software, especifica convenciones de nombres y describe reglas para los tipos, funciones, macros, variables, etc. Se resaltan las reglas que han demostrado reducir o eliminar cierto tipo de errores.

2.1.2. Análisis estático automático

Los verificadores formales de software son efectivos, pero requieren de una especificación formal completa del programa objetivo y casi siempre son demasiado caros y difíciles de utilizar, incluso en los programas donde la seguridad es crítica [Evans y Larochelle, 2002]. En cambio, el análisis estático ligero (“lightweight static analysis”) requiere incrementalmente más esfuerzo que utilizar un compilador, pero sólo una fracción del esfuerzo que se emplearía para llevar a cabo una verificación formal completa.

Por otro lado, buscando el valor del análisis estático automático mediante el estudio de fallas de inspección automática, fallas de inspección manual y peticiones de cambio para sistemas industriales de software de gran escala [Zheng et. al., 2006], se ha encontrado que:

- El costo del análisis estático automático es de la misma magnitud que el costo de las revisiones manuales por falla detectada, pero su efectividad es menos dependiente de factores humanos.
- El rendimiento en la remoción de defectos del análisis estático automático no es significativamente diferente al de las revisiones manuales, mientras el rendimiento en la remoción de defectos de las pruebas basadas en ejecución es dos o tres veces más alto y por lo tanto más efectivo.
- El número de fallas de análisis estático automático en un módulo puede ser una buena medida para la detección de módulos con propensión a fallas.
- Las herramientas de análisis estático automático identifican predominantemente los defectos de asignación (errores con la inicialización de bloques de control o estructuras de datos) y de chequeo (errores en la lógica del programa para validar datos y valores antes de que sean utilizados, condiciones de ciclos, etc).
- La gran mayoría de fallas encontradas por las revisiones manuales son fallas de algoritmo (errores de eficiencia o de exactitud), de documentación (errores en la documentación y notas de mantenimiento) y de chequeo.
- Un gran porcentaje de los errores de programación detectados por las herramientas de análisis estático automático tienen el potencial de causar vulnerabilidades de seguridad.
- El análisis estático automático es un complemento económico y costeable comparado con otras formas de verificación y validación.

2.1.2.1. Desarrollos y soluciones existentes

A continuación se presentan algunas de las herramientas COTS existentes más importantes de análisis estático para C, acompañadas de sus características principales, sus ventajas y sus desventajas (ver Anexo B).

Analizando la tabla del Anexo B, se observa que la mayor parte de las herramientas de análisis estático referidas no se encuentran bien integradas a un ambiente de desarrollo (IDE). No checan por algún estándar de codificación en específico, requieren de trabajo de

configuración adicional y corren separadamente, tomando (en muchos de los casos) más tiempo que el necesario para la compilación.

Respecto a la integración y al tiempo necesario para correr, esto es consistente con lo planteado por Kleidermacher [Kleidermacher, 2008].

BLAST (licencia Apache), Frama-C (código abierto) y LDRA Testbed (comercial) son los más completos. BLAST requiere de entrada otros programas en C, lo cual lo hace un tanto difícil de configurar. Frama-C recibe una especificación en un lenguaje llamado ACSL que hay que aprender y la herramienta está todavía en desarrollo. LDRA parece ser el estado del arte tanto para las pruebas unitarias, como de integración y para el análisis estático pero necesita reformatear el código, es lento correrlo y su costo es elevado.

Respecto a la integración con Eclipse, existe una extensión para integrar BLAST y Eclipse. LDRA Testbed puede ser invocado desde Eclipse mediante una extensión, y Goanna es lo más cercano a la integración perfecta con Eclipse CDT, ya que corre combinando Eclipse CDT y el compilador gcc. Aunque es capaz de detectar desviaciones en tiempo de codificación, es una herramienta comercial de código cerrado que no se basa en un estándar de codificación específico y solamente puede ser integrada a Eclipse en los sistemas Linux.

TERCER CAPÍTULO
PLANTEAMIENTO DE LA SOLUCIÓN

3. PLANTEAMIENTO DE LA SOLUCIÓN

La solución al problema presentado en la sección 1.1 Planteamiento del problema consistirá de las siguientes partes:

1. Definición de la investigación.
    a. Justificación e importancia del problema.
    b. Preguntas de investigación.
    c. Propuesta de solución
    d. Variables de investigación.
    e. Objetivos de investigación.
    f. Alcance de la investigación.
2. Método de desarrollo.
3. Desarrollo
    a. Definición de reglas de análisis estático.
    b. Especificación e implementación de la solución.
    c. Evaluación.
    d. Resultados.

En el primer punto se definirán los detalles de la investigación como el alcance, los objetivos, las variables de investigación, entre otros aspectos; en el segundo punto se definirá y se expondrá el método de desarrollo de software que será utilizado para implantar la solución, y en el último punto se definirán las reglas de análisis estático objetivo y se implantará y evaluará la solución.

3.1. Definición de la investigación

En esta sección se definirán y se detallarán los principales componentes de la presente investigación con el propósito de establecer claramente la importancia, el alcance y los objetivos.

3.1.1. Justificación e importancia del tema

C es el lenguaje de programación dominante para el desarrollo de software embebido [Barr, 2009]. Linux (en cualquiera de sus múltiples variantes), el sistema operativo de código abierto líder, está escrito en C [Scott, 2006].

Considerando la relevancia del lenguaje C (ver sección 1.2.6) para el desarrollo de software embebido, para el desarrollo Linux, en la utilización de Linux como sistema operativo embebido, el crecimiento que han tenido y tendrán los sistemas embebidos [Global Information, 2005] y los múltiples modos de falla del lenguaje (lo dependiente de la implementación, lo no especificado, lo implícito, etc.), resulta de vital importancia estudiar y analizar el lenguaje en sí mismo y las técnicas de verificación que pudieran coadyuvar en la detección eficaz y oportuna de defectos.

Para lenguajes como C, el análisis estático es una técnica efectiva para descubrir errores de software [Sommerville, 2007]. El costo del análisis estático automático es de la misma magnitud que el costo de las revisiones manuales por falla detectada, pero su efectividad es menos dependiente de factores humanos [Zheng et. al., 2006].

Mientras la gran mayoría de fallas encontradas por las revisiones manuales son fallas de algoritmo, de documentación y de chequeo, un gran porcentaje de los errores de programación detectados por las herramientas de análisis estático automático, tienen el potencial de causar vulnerabilidades de seguridad, por lo que el análisis estático automático es un complemento económico y costeable comparado con otras formas de verificación y validación.

Aunque el análisis estático automático es considerado efectivo en la detección de defectos de software, el tiempo requerido para ejecutarlo, la falta de integración con los ambientes de desarrollo y el alto porcentaje de falsos positivos (30% al 100%) son obstáculos que siguen evitando su adopción generalizada (ver secciones 1.1 y 1.3.3.1).

Finalmente, si tomamos en cuenta que para muchas de las reglas de los estándares de codificación no hay evidencia sustancial que las ligue a fallas reales [Boogerd y Moonen, 2009a], no siendo la excepción el caso de MISRA-C (ver sección 1.3.2), se enfrenta un panorama en el cual la técnica de verificación es costeable y efectiva, pero las

implantaciones ruidosas y complicadas. Es ahí donde la presente investigación toma rumbo y sentido como se describe a continuación.

3.1.2. Preguntas de investigación

¿Existirá un conjunto de reglas de análisis estático que sea considerablemente menos ruidoso que MISRA-C?

¿Será posible llegar a una solución de análisis estático que se encuentre bien integrada al ambiente de desarrollo, cuya ejecución sea expedita, que se base en reglas de codificación ligadas a evidencia de falla objetiva y cuyo uso en campo arroje una relación de falsos positivos considerablemente menor al 30%?

3.1.3. Propuesta de solución

Dado que el ruido de análisis estático automático depende en primer lugar del ruido del conjunto de reglas, la propuesta de solución consiste en definir un conjunto de reglas de análisis estático que consista exclusivamente de reglas B.2, a fin de implantarlo extendiendo Eclipse CDT, para así integrarlo directamente al editor de código.

3.1.3.1. Hipótesis

Mediante la extensión a Eclipse CDT, puede implantarse un robusto y ágil analizador estático de código C que detecte violaciones en tiempo de codificación.

3.1.3.2. Estándar de codificación base

El estándar de codificación C base elegido para la presente investigación es EC-A (ver sección 1.2.4.4 para más detalles), ya que a diferencia de MISRA-C, EC-A no requiere de licencia o autorización alguna para ser utilizado, y lo más importante e interesante, provee un conjunto de reglas que se encuentran respaldadas por mediciones de falla.

3.1.3.3. Plataforma de desarrollo base

La plataforma de desarrollo elegida es Eclipse CDT debido a que:

- Es por naturaleza extensible
- Representa tecnología probada, confiable y escalable

- Es estable para la innovación, la libertad y la experimentación
- No tiene vendedor
- Es soportada como ambiente de desarrollo de sistemas embebidos y Linux
- Permite realizar introspección de código C y C++

3.1.3.4. Ventajas y desventajas potenciales

Las ventajas potenciales son:

- Debido a que la solución se basará exclusivamente en reglas B.2, el ruido de las reglas de codificación tenderá a cero; por lo tanto, el ruido de análisis estático dependerá en su totalidad del ruido de la solución de análisis estático automático.
- Debido a que la solución se integrará directamente al editor de código, la ejecución será simple y expedita, y las transgresiones a las reglas de codificación podrán ser detectadas en tiempo de codificación.

Las desventajas potenciales son:

- Debido a que el soporte a la introspección de código C y C++ en Eclipse CDT es incipiente las APIs existentes podrían no ser lo suficientemente maduras y estables para el uso planteado.
- Debido a que la solución utilizará Java, esta podría padecer ciertos problemas de desempeño típicamente relacionados con las soluciones Java.

3.1.4. Variables de investigación

Para la presente investigación se identifican las siguientes variables dependientes:

i. La proporción de falsos positivos.
ii. La proporción de falsos negativos.

3.1.5. Objetivos específicos

Los objetivos específicos de la presente investigación son:

1. Un conjunto de reglas de análisis estático basadas en EC-A.
2. Una extensión (“plug-in”) configurable para Eclipse CDT que ejecute análisis estático de código C en tiempo de codificación.
3. La especificación de la extensión para el análisis estático de código C.

4. Porcentaje de falsos positivos y de falsos negativos de la extensión sobre casos de prueba.
5. Porcentaje de falsos positivos y de falsos negativos de la extensión ya en uso en campo sobre código de producción.

3.1.6. Alcance

Aunque mucho de EC-A es igualmente relevante para C++ el proyecto de investigación se encontrará limitado al lenguaje C. No se estudiará ni se evaluará el lenguaje C++.

La extensión o “plug-in” configurable para Eclipse CDT se basará únicamente en el conjunto de criterios definidos en la investigación por lo que no será posible añadir criterios nuevos a la extensión sin necesidad de modificar el código fuente.

La extensión de análisis estático podría no estar totalmente completada al término del plazo propuesto para esta investigación ya que el tiempo total de desarrollo podría sobrepasar el tiempo y los recursos disponibles.

Finalmente las mediciones en campo se tomarán en sólo cinco proyectos reales debido a la disponibilidad de recursos.

3.2.Método de desarrollo

El desarrollo se centrará en la validación sobre casos de prueba de falsos positivos y de falsos negativos.

Las mediciones de falsos positivos y de falsos negativos serán tomadas sobre casos de prueba y en un ambiente real de utilización de la herramienta.

El método de desarrollo se describe gráficamente a continuación:

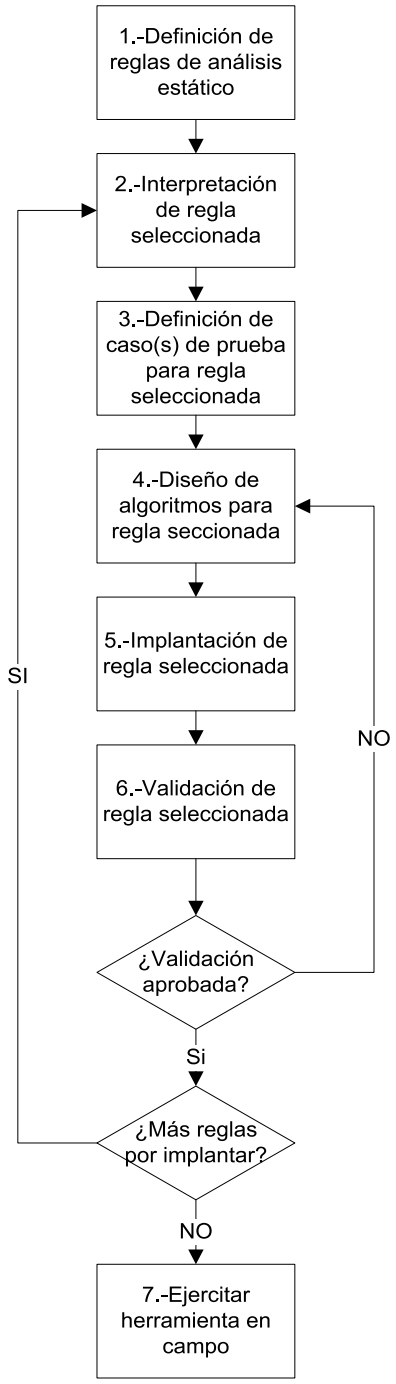

Figura 3. Método de desarrollo de software.

En el paso 1 se definirá un conjunto de reglas de análisis estático basado en EC-A. En el paso 2 se analizará e interpretará la regla en cuestión a manera de análisis de requisitos. En el paso 3 se llevará a cabo el diseño y la definición de los casos de prueba para cada regla.

En el paso 4 se diseñará el algoritmo para la regla en cuestión. En el paso 5 se codificará la regla de análisis estático en cuestión. En el paso 6 se validarán las reglas de análisis estático con la ayuda de los casos de prueba desarrollados en el paso 3 y finalmente en el paso 7 se ejercitará la herramienta en un ambiente real y se obtendrán mediciones y retroalimentación al respecto.

CUARTO CAPÍTULO
DESARROLLO

4. DESARROLLO

4.1. Reglas de verificación de análisis estático

Tomando como base el subconjunto de reglas de ISO C soportado por mediciones, EC-A, se definió un subconjunto de reglas EC-A.

Para la definición del subconjunto de reglas, se tomó en cuenta no sólo el alcance de la investigación y los recursos disponibles, sino las siguientes características típicamente utilizadas para clasificar un buen conjunto de requisitos de software.

a) Claridad: el objetivo de la regla es entendible mediante la lectura y comprensión de su descripción.
b) Factibilidad: la regla puede implantarse utilizando las capacidades actuales de Eclipse CDT.
c) Completitud: la regla provee todos los detalles requeridos para entender su alcance.
d) Implantabilidad: la regla es lo suficientemente atómica y realizable de tal forma que pueden diseñarse algoritmos para cubrirla e implantarse.
e) Consistencia: la regla permite las aplicaciones típicas y válidas del lenguaje.
f) Verificabilidad: pueden diseñarse y aplicarse chequeos y pruebas para la regla en cuestión.

Regla	***Alcance e interpretación***
S_GLOB	*Acotada a los siguientes casos debido a que es una regla muy abierta y extensa.* *a. "Return" requerido para funciones que regresan algo.* *b. "Return" con expresión requerida para funciones que regresan algo.* *c. Pérdida de información en expresiones de retorno para funciones que regresan algo.* *d. Pérdida de signo en expresiones de retorno para funciones que regresan algo.* *e. Expresiones de retorno no requeridas para funciones que regresan nada.*
F_PROT	*Tanto definiciones como llamadas de funciones deben ser precedidas por sus respectivos prototipos de tal forma que el compilador pueda llevar a cabo un chequeo explícito de tipos.*

F_COMP	*Los argumentos provistos en las llamadas a funciones y los parámetros de las definiciones de funciones deben ser compatibles tanto en número como en tipo con sus respectivos prototipos.*
E_PREC	*Debido a que la precedencia de los operadores de igualdad es mayor que la de los operadores de bits, que la de los operadores de asignación y que la de los operadores relacionales su combinación sin mecanismos explícitos de evaluación puede producir resultados no deseados.* *Acotada al chequeo entre operadores de bits y de igualdad y operadores de asignación y de igualdad.* *a. Expr1 OprBit Valor1 OprIgualdad Valor2* *b. Expr1 OprAsignación Expr2 OprIgualdad Valor* *Ejemplos:* *i.* `if (flags & FLAG != 0)` *ii.* `while (c=getc(in) != EOF)`
E_SIDE	*Todo el código debe tener efectos secundarios asociados.*
E_COMM	*La expresión a la izquierda del operador coma debe tener efectos secundarios.*
E_UNEG	*No tiene sentido comparar una expresión sin signo con una expresión negativa.*
E_CSIGN	*No debe permitirse que conversiones implícitas provoquen que un objeto pierda información o su naturaleza signada.* *Acotado a los casos de asignación y para las expresiones de retorno cuando:* *a. Un "long double" se asigna o se convierte a "float" o a algo más pequeño.* *b. Un "double" o un "float" se asigna o se convierte a "int" o a algo más pequeño.* *c. Un "long int" se asigna o se convierte a "short" o "char".* *d. Un "int" o un "short" se asigna o se convierte a "char".* *e. Un "signed long", un "signed int", un "signed short" o un "signed char" se asigna o se convierte a una variable del mismo tipo pero sin signo explícito ("unsigned long", "unsigned int", etc.).*
E_REACH	*Todo el código deberá de ser alcanzable.*
E_HIDE	*Los nombres de variables de módulo o de variables globales no debe ser reusados por variables locales.*
E_EXTR	*No deben declararse variables externas dentro de funciones o de bloques.*
E_FEQL	*Los objetos de punto flotante no deben ser comparados.*
E_FLOOP	*Los objetos de punto flotante no deben ser utilizados como variables de control de ciclos.* *Acotado para las expresiones de control de ciclos.*
E_BITF1	*Los bit-fields de tamaño 1 deben ser declarados con signo o sin signo explícitamente.*
E_NARRW	*La conversión explícita de un apuntador a entero no deberá truncar la dirección referida por el apuntador.*

	Debido a que los apuntadores son típicamente de 32 bits, los casos a checar son: *a. La conversión de un apuntador a "char".* *b. La conversión de un apuntador a "short".*
E_PCAST	*Los apuntadores no deben ser convertidos explícitamente a otros tipos de apuntadores. Esta regla fue dejada de lado ya que al menos para software embebido existen muchos casos donde su uso es inevitable. Por lo tanto su chequeo sería visto como ruido por los desarrolladores.*
E_CASE1	*Todas las estructuras multi-decisión ("switch") deberán tener como mínimo un caso y deberán tener un caso por defecto ("default").*
E_CFALL	*En las estructuras multi-decisión ningún caso o caso por defecto deberá ser alcanzable si hay código entre el caso previo y el caso en cuestión.*
P_ARGS2	*Los argumentos deben utilizarse sólo una vez en los macros con argumentos.* *Esta regla fue dejada de lado ya que los APIs no permiten acceder a las funciones del preprocesador.*
P_PAREN	*Todos los argumentos de los macros deben ser puestos entre paréntesis a menos que esto implique un error de sintaxis.* *Esta regla fue dejada de lado ya que puede no ser realizable.*

Tabla 7. Subconjunto de reglas EC-A

4.2. Extensión de la plataforma Eclipse CDT

La extensión se basa en las APIs y el ambiente de pruebas utilizado por CODAN. Los paquetes desarrollados son "org.eclipse.cdt.codan.internal.checkers" y "org.eclipse.cdt.codan.core.internal.checkers". El primero contiene las clases que implementan los chequeos de análisis estático y el segundo contiene las clases que son utilizadas tanto como pruebas automáticas como pruebas de regresión.

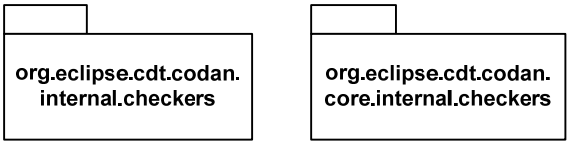

Figura 4. Paquetes de la extensión a Eclipse CDT.

El siguiente diagrama presenta las clases de análisis estático desarrolladas.

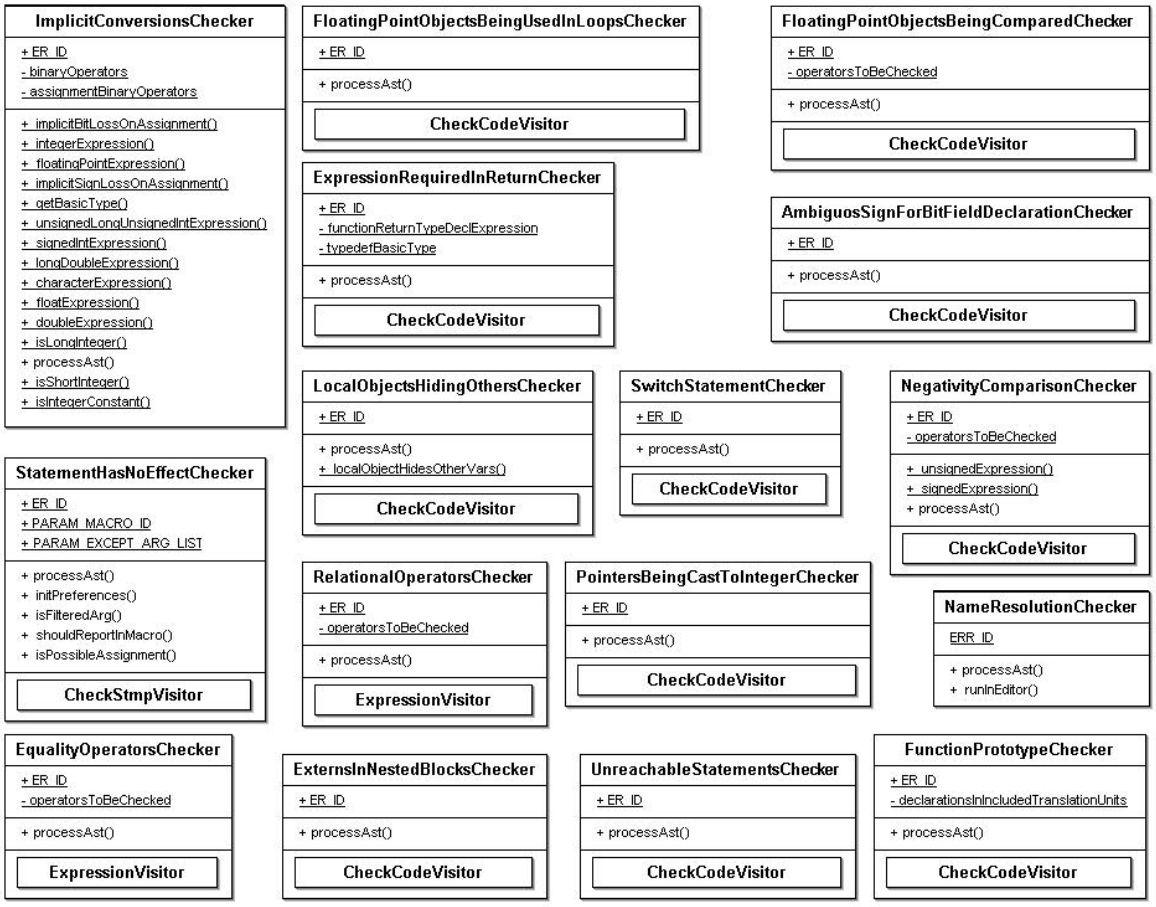

Figura 5. Diagrama de clases.

Cada clase encapsula un tipo de verificación de análisis estático y cada clase cubre una o más reglas EC-A como se explica en la siguiente sección.

La clase “ImplicitConversionsChecker” detecta los casos en los cuales puede ocurrir pérdida de información o pérdida de signo no deseada, la clase “FloatingPointObjectsBeingUsedInLoopsChecker” detecta cuando se utilizan variables de tipo flotante para el control de ciclos, la clase “FloatingPointObjectsBeingComparedChecker” reporta los casos en los cuales se comparan entidades de punto flotante ya que errores de precisión podrían ocurrir, la clase “ExpressionRequiredInReturnChecker” detecta errores comunes en las expresiones de retorno, la clase “AmbiguosSignForBitFieldDeclarationChecker” reporta cuando se omite el signo para los campos de bits de longitud 1, la clase “StatementHasNoEffectChecker”

detecta una gran variedad de instrucciones sin efecto, la clase "LocalObjectsHidingOthersChecker" reporta cuando variables locales esconden variables globales, la clase "SwitchStatementChecker" verifica que las construcciones "switch" no se encuentren vacías y que tengan un caso por defecto, la clase "NegativityComparisonChecker" reporta cuando una variable no signada es comparada con cantidades negativas, la clase "RelationalOperatorsChecker" detecta posibles problemas de precedencia cuando los operadores de igualdad son combinados con operadores relacionales, la clase "PointersBeingCastToIntegerChecker" verifica que las conversiones de tipo de apuntadores a enteros no pierdan información, la clase "NameResolutionChecker" reporta los nombres que no son reconocidos como variables o entidades válidas, la clase "EqualityOperatorsChecker" reporta posibles errores de precedencia cuando se combinan operadores de igualdad con operadores de asignación y de bits, la clase "ExternsInNestedBlocksChecker" reporta las declaraciones externas dentro de las funciones o en bloques más internos, la clase "UnreachableStatementsChecker" reporta las instrucciones inalcanzables y finalmente la clase "FunctionPrototypeChecker" verifica que para cada llamada a función que se realice y para cada función exista un prototipo compatible en número y tipo de argumentos. En la sección 4.3.7 se presentan mayores detalles de las reglas cubiertas y ejemplos relacionados.

4.2.1. Matriz de rastreabilidad

A continuación se presenta el subconjunto de reglas EC-A, los detalles de su implementación, su localización en el código fuente y el modelo CODAN utilizado.

Regla	***Estado***	***Clase***	***Modelo***	***Comentarios***
S_GLOB	*Implantada*	*ExpressionRequiredInReturnChecker*	*AST ControlFlowGraph*	
F_PROT	*Implantada*	*FunctionPrototypeChecker*	*AST Índex*	*Necesita modificarse el Parser para que este tipo de chequeos no tomen*

				tantos recursos ó ejecutar el chequeo sólo cuando se solicite.
F_COMP	*Implantada*	*FunctionPrototypeChecker*	*AST* *Índex*	*Necesita modificarse el Parser para que este tipo de chequeos no tomen tantos recursos ó ejecutar el chequeo sólo cuando se solicite.*
E_PREC	*Implantada*	*EqualityOperatorsChecker*	*AST*	
E_SIDE	*Implantada*	*StatementHasNoEffectChecker*	*AST*	
E_COMM	*Implantada*	*StatementHasNoEffectChecker*	*AST*	
E_UNEG	*Implantada*	*NegativityComparisonChecker*	*AST*	
E_CSIGN	*Implantada*	*ImplicitConversionsChecker* *ExpressionRequiredInReturnChecker*	*AST* *ControlFlowGraph*	
E_REACH	*Implantada*	*UnreachableStatementsChecker*	*AST* *ControlFlowGraph*	*Se encontró que algunos nodos no alcanzados no son incluidos en el modelo.*
E_HIDE	*Implantada*	*LocalObjectsHidingOthersChecker*	*AST* *Índex*	*Debido a que se necesitan bloqueos de lectura del índex el chequeo afecta el editor por lo que debería ejecutarse sólo cuando se solicite.*
E_EXTR	*Implantada*	*ExternsInNestedBlocksChecker*	*AST*	
E_FEQL	*Implantada*	*FloatingPointObjectsBeingCompared Checker*	*AST*	

E_FLOOP	*Implantada*	*FloatingPointObjectsBeingUsedInLoops Checker*	*AST*	
E_BITF1	*Implantada*	*AmbiguosSignForBitFieldDeclaration Checker*	*AST*	
E_NARRW	*Implantada*	*PointersBeingCastToIntegerChecker*	*AST*	
E_PCAST	*No implantada*			
E_CASE1	*Implantada*	*SwitchStatementChecker*	*AST*	
E_CFALL	*Implantada*	*SwitchStatementChecker*	*ControlFlowGraph*	*Similar al caso de* **E_REACH** *se encontraron errores que no permiten el chequeo confiable de esta regla.*
P_ARGS2	*No implantada*			
P_PAREN	*No implantada*			

4.3. Especificación de la extensión

A continuación se describen los detalles y los requisitos de la extensión desarrollada así como las reglas que cubre, ejemplos, detalles de instalación, etc.

4.3.1. Requisitos generales

Los requisitos generales de la extensión desarrollada son:

Rubro	***Requisito***
Sistema operativo	Cualquiera con Java 1.6 o posterior
Plataforma	Eclipse CDT 7.0
Lenguaje objetivo	C
Tipo de proyecto Eclipse	Cualquier proyecto de Eclipse CDT
Estándar de codificación base	EC-A
Configuración por defecto	Todas las reglas activadas para todo el proyecto
Configuración por regla	Activación, desactivación más patrones de inclusión y exclusión para archivos y

	directorios
Invocación	Automática con el editor de código o por demanda sobre todo el proyecto

Tabla 8. Requisitos de la extensión a Eclipse CDT

4.3.2. Instalación

La instalación de la extensión de análisis estático en tiempo de codificación es sencilla. Una vez que se tiene Eclipse CDT 7.0 instalado sólo hay que copiar los archivos jar de la extensión al directorio Eclipse\plugins y reiniciar Eclipse CDT.

Cuando la extensión ha sido instalada exitosamente el menú contextual del proyecto incluye la opción “Run C/C++ Code Analysis” el cual corre el análisis estático sobre todo el proyecto.

4.3.3. Configuración por defecto

La configuración por defecto de la herramienta consiste en tener todas las reglas activas para el proyecto en cuestión sin patrones de exclusión o inclusión. La activación, desactivación de reglas y los patrones de exclusión e inclusión están accesibles dentro del menú contextual “Properties”, en la categoría “C/C++ General”, “Code Analysis” como se muestra a continuación.

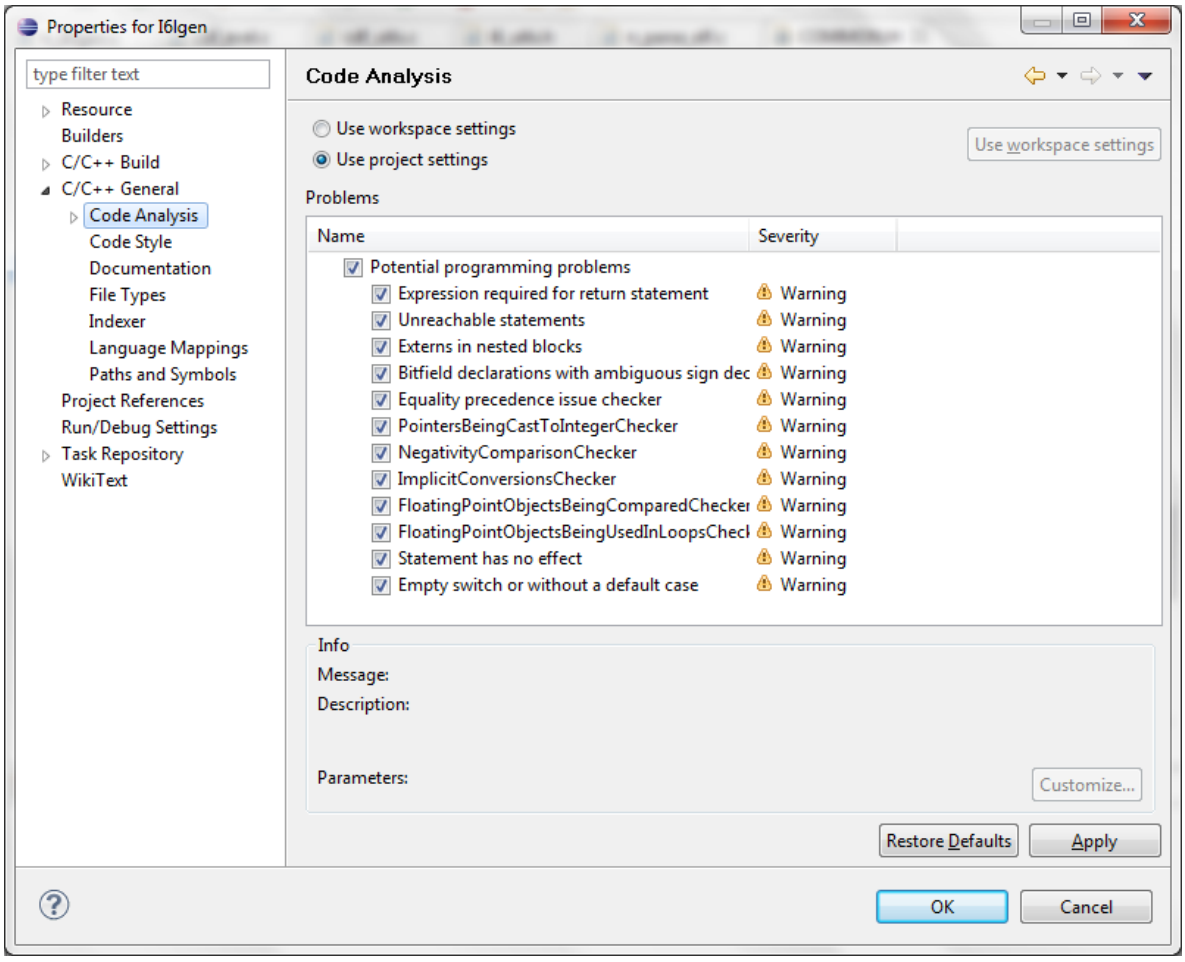

Figura 6. Ventana de configuración de la extensión.

4.3.4. Activación y desactivación de reglas

La activación y desactivación de reglas se controla mediante el estado del “checkbox” relacionado. Por ejemplo si quiere desactivarse la regla “Empty switch or without a default case” basta con de-seleccionar dicha regla y aceptar el cambio de configuración. Las reglas seleccionadas son las reglas activas.

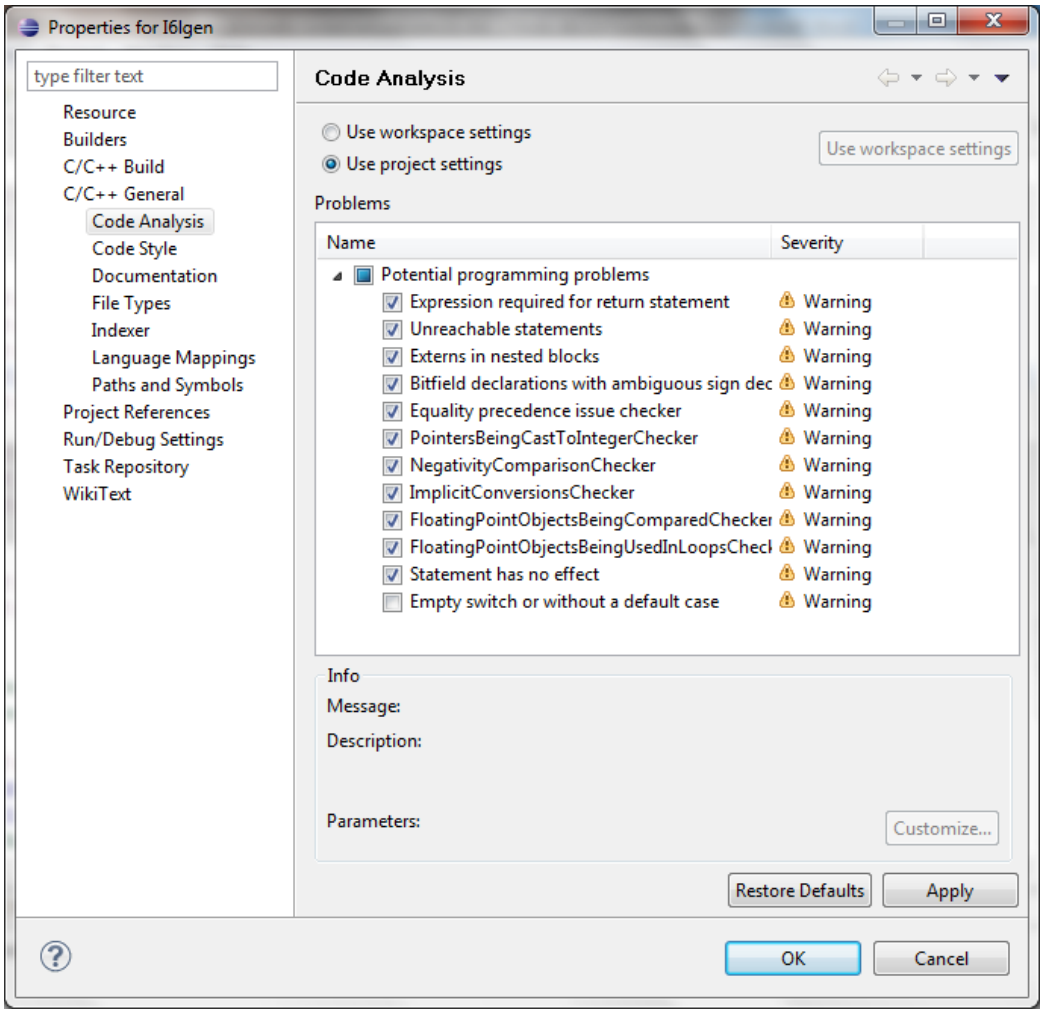

Figura 7. Desactivación de reglas.

4.3.5. Patrones de inclusión y exclusión

Además de tener la posibilidad de activar o desactivar selectivamente ciertas reglas de análisis estático es posible especificar patrones de inclusión y exclusión para archivos y directorios por regla.

Los patrones de exclusión omiten el chequeo de cierta regla para el patrón o los patrones especificados. Los patrones de inclusión especifican el alcance de aplicación de la regla.

Si no existen patrones ni de exclusión ni de inclusión para cierta regla activa el alcance es todo el código fuente del proyecto.

Los patrones de inclusión y de exclusión son agregados de la siguiente forma:

a. Estando en la ventana de configuración escogiendo una regla específica.
b. Haciendo click en “Customize”.
c. Dando click en la pestaña “Scope”.
d. Dando click en “Add” o “Add Multiple”.

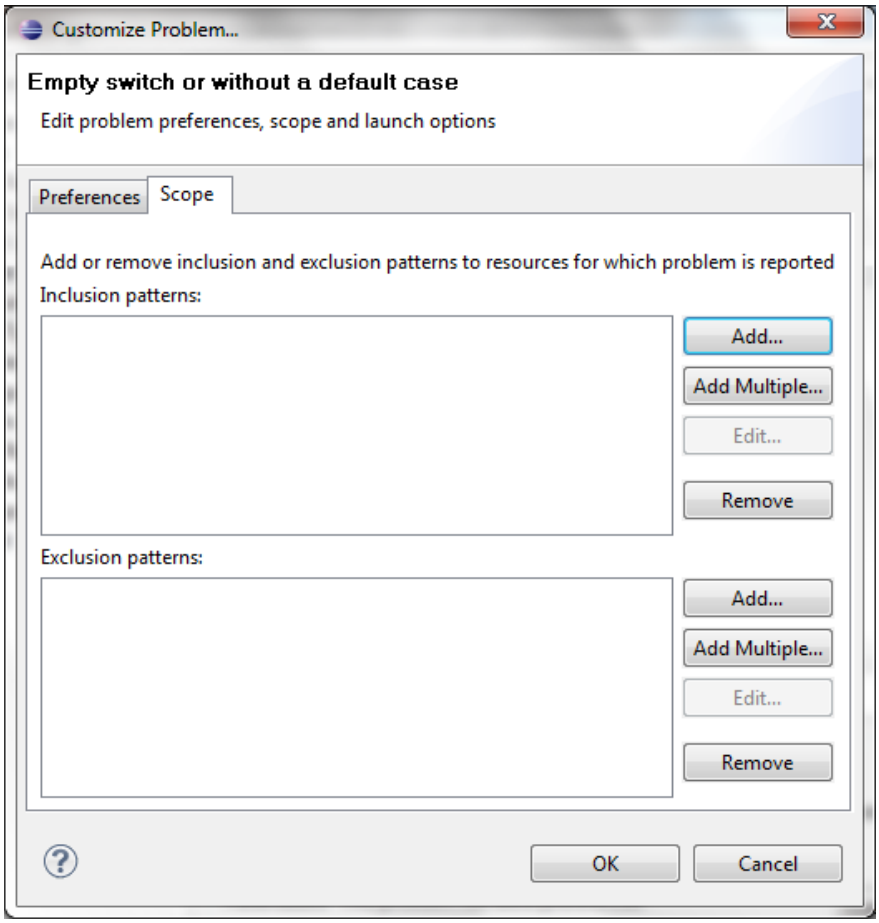

Figura 8. Ventana de especificación de patrones de inclusión o exclusión.

La remoción de patrones de inclusión y exclusión se da mediante la selección de ciertos patrones y seleccionando la acción “Remove”.

4.3.6. Invocación

Una de las principales características de la extensión de análisis estático desarrollada es que la invocación se encuentra totalmente integrada al editor por lo que típicamente sólo hay que escribir o editar código para que el analizador estático sea invocado.

No obstante si se desea correr el análisis estático sobre todo el proyecto como en el caso de código heredado es necesario sobre el nodo principal del proyecto ejecutar la acción secundaria “Run C/C++ Code Analysis”.

4.3.7. Reglas incluidas

A continuación se presentan las reglas de análisis estático que son cubiertas y ejemplos típicos relacionados.

4.3.7.1. S_GLOB – “Expression required for return statement”

Este chequeo verifica que se tengan instrucciones “return” cuando se necesitan con sus respectivas expresiones de retorno. Los casos reportados son:

a. Funciones que deben regresar algo sin instrucciones “return”
b. Expresiones ausentes en funciones que regresan algo.
c. Pérdida de información o de signo en las expresiones de retorno.
d. Presencia de expresiones en instrucciones “return” para funciones void.

Un ejemplo del caso d se muestra a continuación.

```
void void_function(void) {
return 10; //Mensaje reportado aquí
}
```

4.3.7.2. E_PREC – “Equality precedence issue checker”

Este chequeo reporta aquellas situaciones en las cuales la combinación de los operadores de igualdad con operadores de bits u operadores de asignación puede no producir los resultados deseados. Un ejemplo de este tipo de casos se muestra a continuación.

```
if (var & 0x01 != 0) { //Mensaje reportado aquí
}
```

4.3.7.3. E_SIDE, E_COMM – "Statement has no effect"

Este chequeo reporta aquellas instrucciones que no tienen efecto incluyendo el caso donde la expresión a la izquierda del operador coma no tiene efectos. Un ejemplo del tipo de casos reportados se muestra a continuación.

```
main() {
int a,b;

b+a; //Mensaje reportado aquí
}
```

4.3.7.4. E_UNEG – "Negativity comparison checker"

Este chequeo reporta aquellas expresiones sin signo comparadas por negatividad. Un ejemplo se muestra a continuación.

```
unsigned long int ulong_variable=50;


unsigned short function(void) {
if (ulong_variable < -1) //Mensaje reportado aquí
return 50;
else {
return 100;
}
}
```

4.3.7.5. E_CSIGN – "ImplicitConversionsChecker"

Este chequeo reporta para las expresiones de retorno y para las asignaciones aquellas situaciones donde se trunca el tamaño de una variable o se pierde el signo de esta debido a conversiones implícitas. El caso de las expresiones de retorno es cubierto por el chequeo "Expression required for return statement". El caso de las asignaciones es cubierto por el presente chequeo. Un ejemplo reportado de pérdida de información se muestra a continuación.

```
int main(void) {
long double ld = 1.45544445577888;
float f = 1.2;
```

```
f = ld; //Mensaje reportado aquí
return 0;
}
```

4.3.7.6. E_REACH – "Unreachable statements"

Este chequeo reporta las expresiones inalcanzables. Un ejemplo se muestra a continuación.

```
switch (variable) {
case LAB1:
return;
variable--; //Mensaje reportado aquí
}
```

4.3.7.7. E_EXTR – "Externs in nested blocks"

Este chequeo reporta las variables que son declaradas como externas dentro de funciones o dentro de otros bloques. Un ejemplo se muestra a continuación.

```
int main(void) {
extern double module_scope_var; //Mensaje reportado aquí

return (int) module_scope_var;
}
```

4.3.7.8. E_FEQL

Este chequeo, "FloatingPointObjectsBeingComparedChecker", reporta las expresiones flotantes que son comparadas por igualdad o desigualdad. Un ejemplo se muestra a continuación.

```
#define FLOAT_VALUE 4.99999998e+011

int main(void) {
float fvar = FLOAT_VALUE;

if (fvar == FLOAT_VALUE) { //Mensaje reportado aquí
return 1;
} else {
return 0;
}
}
```

4.3.7.9. E_FLOOP

Este chequeo, “FloatingPointObjectsBeingUsedInLoopsChecker”, reporta los ciclos que utilizan expresiones de control flotantes. Un ejemplo se muestra a continuación.

```
int main(void) {
float a = 10;

while (a) { //Mensaje reportado aquí
      printf("Secuencia: %i\n", a--);
}

return 0;
}
```

4.3.7.10. E_BITF1

Este chequeo, “Bitfield declarations with ambiguous sign”, reporta los campos de bits de longitud 1 para los cuales su signo es ambiguo. Un ejemplo se muestra a continuación.

```
typedef struct My_Struct_ {
int OneBitBitField :1; //Mensaje reportado aquí
} typedefed_structure;
```

4.3.7.11. E_NARRW – “PointersBeingCastToIntegerChecker”

Este chequeo reporta cuando la dirección de un apuntador es convertida a “char” o a “short”. Un ejemplo se muestra a continuación.

```
struct My_Struct *my_struct_pointer;

int main(void) {
short addr;

addr = (short) my_struct_pointer; //Mensaje reportado aquí

return addr;
}
```

4.3.7.12. E_CASE1 – "Empty switch or without a default case"

Este chequeo reporta cuando un "switch" está vacío o cuando le falta el caso por defecto. Un ejemplo se muestra a continuación.

```
switch (value) { //Mensaje reportado aquí
case CASE1:
break;
case CASE2:
break;
}
```

4.4. Evaluación

En esta sección se determinará la proporción de falsos positivos y de falsos negativos de la herramienta de análisis estático desarrollada. Dicha proporción será medida en primer lugar sobre los casos de prueba, y en segundo lugar sobre el uso en campo de la herramienta en 5 proyectos reales de software. La lista de proyectos de software seleccionados para la evaluación de la herramienta desarrollada y algunos detalles como tamaño, lenguaje de programación, etc., son presentados a continuación.

Proyecto	***Propósito***	***Lenguaje***	***Compilador***	***Tamaño***	***Plataforma***
I6lgen	Conversión de información de depuración en formato COFF, ELF-DWARF, ELF-STABS a formato IEEE para propósitos de calibración, pruebas e instrumentación.	ANSI C	Bcc32 y gcc	Mediano (71,562 líneas)	Windows, Linux, Unix HP, Unix Solaris
I695x	Conversión de archivos IEEE a formato legible y producción de listado	ANSI C y C++	Bcc32 y gcc	Grande (98,746 líneas)	Windows, Linux, Unix HP,

	de símbolos				Unix Solaris
Bootloader	Reprogramación integrada en el mismo micro-controlador vía CAN.	ANSI C con extensiones para software embebido	Ca850	Pequeño (25,748 líneas)	Micro-controlador V850 DX4
Halo	Aplicaciones de telemática utilizando GPS y red de telefonía móvil.	ANSI C y C++ con extensiones para software embebido	IAR	Grande (104,209 líneas)	Micro-controlador V850 DX4

Tabla 9. Proyectos de software seleccionados

4.4.1. Medición sobre casos de prueba

Esta sección presenta la proporción de falsos positivos y de falsos negativos medida sobre los casos de prueba desarrollados, que son utilizados tanto como pruebas automáticas como pruebas de regresión.

4.4.1.1. Proporción de falsos positivos

Respecto al número total de reglas cubiertas la proporción de falsos positivos sobre los casos de prueba es del 0%. La siguiente tabla describe la proporción por regla.

Regla	***Clase de pruebas***	***Modelo***	***Proporción de falsos positivos***
S_GLOB	*ReturnCheckerTest*	*AST ControlFlowGraph*	*0%*

E_PREC	*EqualityOperatorsCheckerTest*	*AST*	*0%*
E_SIDE	*StatementHasNoEffectCheckerTest*	*AST*	*0%*
E_COMM	*StatementHasNoEffectCheckerTest*	*AST*	*0%*
E_UNEG	*NegativityComparisonCheckerTest*	*AST*	*0%*
E_CSIGN	*ImplicitConversionsCheckerTest* *ReturnCheckerTest*	*AST* *ControlFlowGraph*	*0%* *0%*
E_REACH	*UnreachableStatementsCheckerTest*	*AST* *ControlFlowGraph*	*0%*
E_EXTR	*ExternsInNestedBlocksCheckerTest*	*AST*	*0%*
E_FEQL	*FloatingPointObjectsBeingCompared CheckerTest*	*AST*	*0%*
E_FLOOP	*FloatingPointObjectsBeingUsedInLoops CheckerTest*	*AST*	*0%*
E_BITF1	*AmbiguosSignForBitFieldDeclaration Checker*	*AST*	*0%*
E_NARRW	*PointersBeingCastToIntegerChecker Test*	*AST*	*0%*
E_CASE1	*SwitchStatementChecker*	*AST*	*0%*

Tabla 10. Proporción de falsos positivos sobre casos de prueba

4.4.1.2. Proporción de falsos negativos

Respecto al número total de reglas cubiertas la proporción de falsos negativos sobre los casos de prueba es del 15.38%. La siguiente tabla describe la proporción por regla.

Regla	***Clase de pruebas***	***Modelo***	***Proporción de falsos negativos***
S_GLOB	*ReturnCheckerTest*	*AST* *ControlFlowGraph*	*0%*
E_PREC	*EqualityOperatorsCheckerTest*	*AST*	*0%*
E_SIDE	*StatementHasNoEffectCheckerTest*	*AST*	*0%*
E_COMM	*StatementHasNoEffectCheckerTest*	*AST*	*12.5%*
E_UNEG	*NegativityComparisonCheckerTest*	*AST*	*0%*
E_CSIGN	*ImplicitConversionsCheckerTest* *ReturnCheckerTest*	*AST* *ControlFlowGraph*	*0%* *0%*
E_REACH	*UnreachableStatementsCheckerTest*	*AST* *ControlFlowGraph*	*20%*

E_EXTR	*ExternsInNestedBlocksCheckerTest*	*AST*	*0%*
E_FEQL	*FloatingPointObjectsBeingCompared CheckerTest*	*AST*	*0%*
E_FLOOP	*FloatingPointObjectsBeingUsedInLoops CheckerTest*	*AST*	*0%*
E_BITF1	*AmbiguosSignForBitFieldDeclaration Checker*	*AST*	*0%*
E_NARRW	*PointersBeingCastToIntegerChecker Test*	*AST*	*0%*
E_CASE1	*SwitchStatementChecker*	*AST*	*0%*

Tabla 11. Proporción de falsos negativos sobre casos de prueba

4.4.2. Medición en campo

Esta sección presenta la información recabada de falsos positivos y de falsos negativos sobre el uso en campo de la herramienta en 5 proyectos reales de software después de un periodo de uso de tres meses.

4.4.2.1. Proporción de falsos positivos

Respecto al número total de reglas cubiertas la proporción de falsos positivos del uso en campo de la herramienta es del 7.69%. La siguiente tabla describe la proporción por regla.

Regla	***Clase de pruebas***	***Modelo***	***Proporción de falsos positivos***
S_GLOB	*ReturnCheckerTest*	*AST ControlFlowGraph*	*0%*
E_PREC	*EqualityOperatorsCheckerTest*	*AST*	*0%*
E_SIDE	*StatementHasNoEffectCheckerTest*	*AST*	*0%*
E_COMM	*StatementHasNoEffectCheckerTest*	*AST*	*25%*
E_UNEG	*NegativityComparisonCheckerTest*	*AST*	*0%*
E_CSIGN	*ImplicitConversionsCheckerTest ReturnCheckerTest*	*AST ControlFlowGraph*	*0% 0%*
E_REACH	*UnreachableStatementsCheckerTest*	*AST ControlFlowGraph*	*0%*
E_EXTR	*ExternsInNestedBlocksCheckerTest*	*AST*	*0%*
E_FEQL	*FloatingPointObjectsBeingCompared*	*AST*	*0%*

	CheckerTest		
E_FLOOP	*FloatingPointObjectsBeingUsedInLoops CheckerTest*	*AST*	*0%*
E_BITF1	*AmbiguosSignForBitFieldDeclaration Checker*	*AST*	*0%*
E_NARRW	*PointersBeingCastToIntegerChecker Test*	*AST*	*0%*
E_CASE1	*SwitchStatementChecker*	*AST*	*0%*

Tabla 12. Proporción de falsos positivos del uso en campo de la herramienta

4.4.2.2. Proporción de falsos negativos

Respecto al número total de reglas cubiertas la proporción de falsos negativos del uso en campo de la herramienta es del 0%. La siguiente tabla describe la proporción por regla.

Regla	***Clase de pruebas***	***Modelo***	***Proporción de falsos negativos***
S_GLOB	*ReturnCheckerTest*	*AST ControlFlowGraph*	*0%*
E_PREC	*EqualityOperatorsCheckerTest*	*AST*	*0%*
E_SIDE	*StatementHasNoEffectCheckerTest*	*AST*	*0%*
E_COMM	*StatementHasNoEffectCheckerTest*	*AST*	*0%*
E_UNEG	*NegativityComparisonCheckerTest*	*AST*	*0%*
E_CSIGN	*ImplicitConversionsCheckerTest* *ReturnCheckerTest*	*AST* *ControlFlowGraph*	*0%* *0%*
E_REACH	*UnreachableStatementsCheckerTest*	*AST* *ControlFlowGraph*	*0%*
E_EXTR	*ExternsInNestedBlocksCheckerTest*	*AST*	*0%*
E_FEQL	*FloatingPointObjectsBeingCompared CheckerTest*	*AST*	*0%*
E_FLOOP	*FloatingPointObjectsBeingUsedInLoops CheckerTest*	*AST*	*0%*
E_BITF1	*AmbiguosSignForBitFieldDeclaration Checker*	*AST*	*0%*
E_NARRW	*PointersBeingCastToIntegerChecker Test*	*AST*	*0%*
E_CASE1	*SwitchStatementChecker*	*AST*	*0%*

Tabla 13. Proporción de falsos positivos del uso en campo de la herramienta

QUINTO CAPÍTULO

RESULTADOS

5. RESULTADOS

5.1. Resultados generales

Los resultados generales de falsos positivos y de falsos negativos se presentan a continuación.

Medición	***Proporción de falsos positivos***	***Proporción de falsos negativos***
Sobre casos de prueba	0%	15.38%
Sobre el uso en campo	7.69%	0%
Promedio	3.84%	7.69%

Tabla 14. Resultados generales de falsos positivos y de falsos negativos

Adicionalmente a las mediciones de falsos positivos y de falsos negativos se recabó información respecto al uso y la configuración de la herramienta mediante un formato de evaluación (ver Anexo A).

Respecto a la experiencia con otras herramientas de análisis estático la mayor parte de los evaluadores ya habían utilizado otras herramientas.

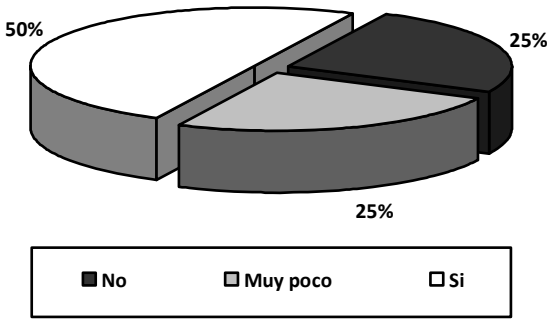

Figura 9. Experiencia previa de los evaluadores con herramientas de análisis estático.

Respecto a la configuración de la herramienta la gran mayoría opinó que es sencilla y rápida.

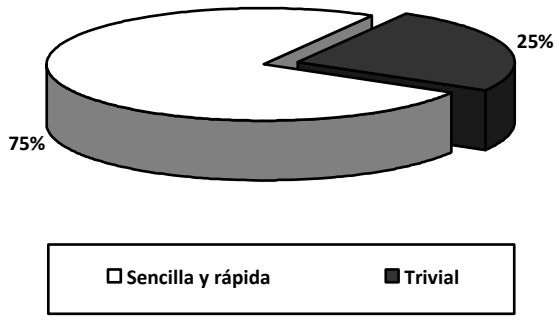

Figura 10. Opinión de los evaluadores sobre la configuración de la herramienta.

Respecto a la invocación de la herramienta dentro del editor de código la mayoría cree que es innovadora y sencilla.

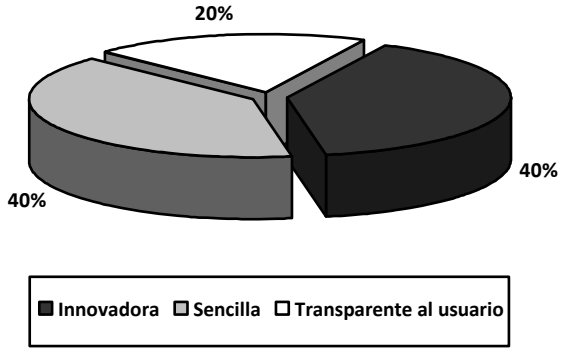

Figura 11. Opinión de los evaluadores respecto a la invocación de la herramienta.

Respecto a los errores potenciales reportados por la herramienta el 100% cree que son acertados.

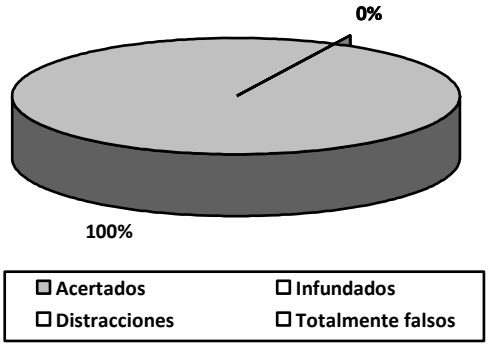

Figura 12. Opinión de los evaluadores respecto los errores reportados por la herramienta.

Respecto a problemas enfrentados durante la evaluación de la herramienta la mayoría reportó que el desempeño del editor se veía afectado.

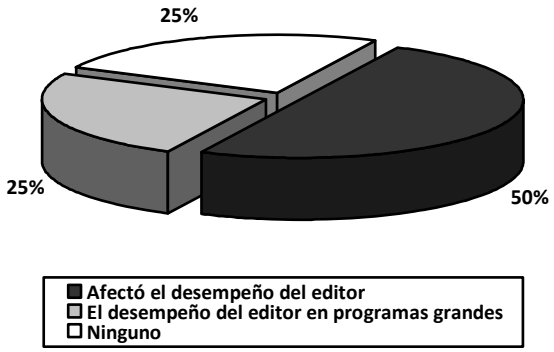

Figura 13. Problemas enfrentados durante la evaluación de la herramienta.

Respecto a sugerencias de mejora a la herramienta el rubro más frecuente fue el de incluir más chequeos.

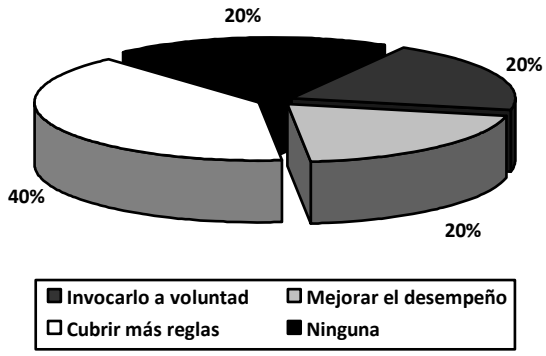

Figura 14. Sugerencias de mejora a la herramienta.

Respecto a la evaluación general de la herramienta el 100% respondió que le pareció una experiencia interesante.

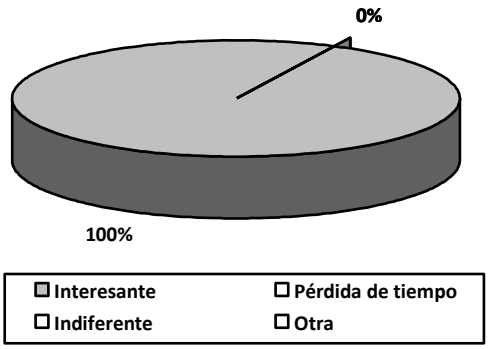

Figura 15. Experiencia en general de la evaluación de la herramienta.

5.2. Ventajas y desventajas de la solución

De las mediciones realizadas, de la aplicación del formato de evaluación y de la opinión proporcionada por los evaluadores encontramos que las ventajas de la solución desarrollada son:

a. Poco ruido de análisis estático debido al bajo porcentaje de bajos positivos y de falsos negativos.
b. Que la solución es independiente de la plataforma de desarrollo y de la plataforma objetivo.
c. Configuración sencilla y rápida.
d. Invocación sencilla y transparente al usuario.

La desventaja de la solución desarrollada es que el desempeño del editor de código se ve afectado, particularmente cuando se trata de proyectos grandes.

5.3. Conclusiones generales

Mediante la extensión a Eclipse CDT y el chequeo de reglas ligadas a modos de falla es posible implantar analizadores estáticos ad hoc de C que detecten en tiempo de codificación violaciones de análisis estático con relaciones de falsos positivos menores al 10%.

La ventaja más importante de este tipo de soluciones es la baja proporción de falsos positivos, en segundo lugar la independencia de la plataforma de desarrollo y de la plataforma objetivo, y finalmente la facilidad de uso.

De los modelos de introspección los más maduros y acertados son el Índex y el AST, mientras el modelo de control de flujo es aún muy impreciso para ser de uso real.

Pensando en chequeos en tiempo de codificación el AST y el modelo de control de flujo pueden ser utilizados sin afectar el desempeño del editor siempre y cuando tales chequeos sean pequeños en magnitud y complejidad. En contraste debido a que el Índex requiere de bloqueos de lectura los cuales hacen conflicto con el editor, los chequeos que requieran de este modelo deben ejecutarse únicamente mediante la solicitud explícita del usuario. El mismo criterio aplica para los chequeos en tiempo de codificación que tiendan a ser extensos y complejos.

Finalmente debido a que el "parser" de Eclipse CDT fue concebido para soportar la edición de código C y C++, hace falta mayor soporte para la introspección de código (más y mejores modelos) y para el análisis estático del mismo.

5.4. Trabajo futuro

Debido a que los problemas de desempeño observados son inconsistentes y difíciles de reproducir sería conveniente como trabajo futuro, determinar si tales problemas son herencia del editor de Eclipse CDT ó si son debidos a algún error de implementación en los algoritmos de análisis estático actuales. El líder del proyecto Eclipse CDT y la comunidad CDT tienen presente [Schaefer, 2010] que hay reportes relacionados al desempeño del editor cuando este es utilizado en proyectos grandes.

Otro aspecto que podría considerarse sería la inclusión de más chequeos de análisis estático basados también en EC-A, teniendo en mente que chequeos pequeños pueden realizarse en tiempo de codificación mientras chequeos más complejos y extensos deben hacerse ya sea extendiendo el "parser" ó ejecutando tales chequeos sólo bajo petición explícita del usuario.

Finalmente podría extenderse aún más el soporte del "parser" de Eclipse CDT para el análisis estático y/o trabajar en modelos adicionales para el soporte de otro tipo de chequeos de análisis estático y/o mejorar los modelos actuales.

BIBLIOGRAFÍA

[Adams, 1984] Adams, Edward N. 1984. *"Optimizing preventive service of software products"*. IBMJournal of Research and Development.

[Anderson, 2009] Anderson, Paul. 2009. *"Finding defects using Holzmann's Power of 10 rules for writing safety critical code"*. Disponible en: http://www.embedded.com/216500700. Consultado el 13 de febrero de 2011.

[Automotive SIG, 2010] Automotive SIG. 2010. Páginas 15 y 58 en *"Automotive SPICE Process Assesment Model"*. Spice User Group.

[Baca et. al., 2009] Baca, Dejan., Petersen, Kai., Carlsson, Bengt., y Lundberg, Lars. 2009. *"Static Code Analysis to Detect Software Security Vulnerabilities – Does Experience Matter?"*. IEEE.

[Barr, 2009] Barr, Michael. 2009. *"Real men program in C"*. Disponible en: http://www.embedded.com/218600142. Consultado el 7 de febrero de 2010.

[Boogerd y Moonen, 2009a] Boogerd, Cathal., y Moonen, Leon. Página 2 en *"Assessing the Value of Coding Standards: An Empirical Study"*. *Technical Report* TUDSERG-2008-017. Delft University of Technology.

[Boogerd y Moonen, 2009b] Boogerd, Cathal., y Moonen, Leon. 2009. *"Evaluating the Relation Between Coding Standard Violations and Faults Within and Across Software Versions"*. IEEE.

[Carnegie Mellon, 2010] Carnegie Mellon University. 2010. *"Secure Coding Standards"*. Disponible en: https://www.cert.org/secure-coding/scstandards.html. Consultado el 21 de marzo de 2011.

[Carnegie Mellon, 2011] Carnegie Mellon University. 2011. *"Welcome to CERT"*. Disponible en: https://www.cert.org/. Consultado el 21 de marzo de 2011.

[Cheung, 2010] Cheung, Ken. 2010. *"LDRA, Netrino Team on Embedded C Coding Standard for Static Analysis"*. Disponible en: http://edageek.com/2010/04/15/misra-model/. Consultado el 27 de marzo de 2011.

[Clayber et. al., 2006] Clayber, Eric., y Rubel, Dan. 2006. Foreword en *"Eclipse: Building Commercial-Quality Plug-ins"*. Addison Wesley Professional.

[CMMI Product Team, 2006] CMMI Product Team. 2006. Página 496 en *"CMMI for Development Version 1.2"*. Software Engineering Institute.

[Evans y Larochelle, 2002] Evans, David., y Larochelle, David. 2002. Páginas 43 y 44 en "*Improving Security Using Extensible Lightweight Static Analysis"*. IEEE Software.

[Global Information, 2005] Global Information Inc. 2005. *"Future of Embedded Systems Technology"*. Disponible en: http://www.the-infoshop.com/report/bc31319-embedded-systems.html. Consultado el 10 de abril de 2001.

[Hatton, 2003a] Hatton L. 2003. Páginas 4-13 en "*Safer Language Subsets: an overview and a case history, MISRA C"*. Information and Software Technology.

[Hatton, 2003b] Hatton L. 2003. *"EC-A measurement based safer subset of ISO C suitable for embedded system development"*. Computing Laboratory. University of Kent.

[Hatton, 2007] Hatton L. 2007. Página 10 en *"Language subsetting in an industrial context: A comparison of MISRA C 1998 and MISRA C 2004"*. Information and Software Technology.

[HIS, 2009] Hersteller Initiative Software. 2009. *"HIS"*. Disponible en: http://portal.automotive-his.de/index.php?lang=english. Consultado el 27 de marzo de 2011.

[Holzmann, 2006] Holzmann, Gerard J. 2006. *"The Power of 10: Rules for Developing Safety-Critical Code"*. NASA/JPL Laboratory for Reliable Software.

[Humphreys, 2009] Humphreys, Paul. 2009. "*PRODUCT HOW-TO: Automating Compliance to MISRA C/C++ Standards"*. Disponible en: http://www.embedded.com/221600117. Consultado el 9 de febrero de 2010.

[IEEE, 2002] IEEE Standards Board. 2002. Páginas 70 y 73 en *"IEEE Std 610.12-1990(R2002)"*. IEEE.

[Jones, 2006] Jones, Nigel. 2006. *"Introduction to MISRA C*". Disponible en: http://www.eetimes.com/discussion/other/4023981/Introduction-to-MISRA-C. Consultado el 2 de marzo de 2011.

[Kernighan et. al., 1998] Kernighan, Brian W., y Ritchie, Dennis M. 1998. Prefacio en "*The C Programming Language"*. Prentice-Hall.

[Kleidermacher, 2008] Kleidermacher, David N. 2008. "*Using static analysis to diagnose & prevent failures in safety-critical device designs"*. Disponible en: http://www.embedded.com/210601784. Consultado el 15 de febrero de 2010.

[Kremenek et. al., 2004] Kremenek, Ted., Ashcraft, Ken., Yang, Junfeng., y Engler, Dawson. 2004. In Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). Páginas 83-93 en "*Correlation exploitation in error ranking"*. ACM Press.

[Laplante, 2007] Laplante, Philip A. 2007. Páginas 516 y 530 en *"What every engineer should know about software engineering"*. CRC Press.

[Laskavaia, 2010] Laskavaia, Alena. 2010. *"Codan: a C/C++ Static Analysis Framework for CDT"*. Disponible en: http://www.slideshare.net/laskava/cc-static-analysis-framework-for-cdt-codan-3551890. Consultado el 28 de febrero de 2011.

[Laskavaia, 2011] Laskavaia, Alena. 2011. *"Codan: a C/C++ Static Analysis Framework for CDT"*. Disponible en: http://www.slideshare.net/laskava/eclipse-con2011-v11. Consultado el 29 de junio de 2011.

[LDRA, 2011a] LDRA Ltd. 2011. *"HIS Certification with the LDRA tool suite"*. Disponible en: http://www.ldra.com/his.asp. Consultado el 27 de marzo de 2011.

[LDRA, 2011b] LDRA Ltd. 2011. *"IPA/SEC C Coding Practice Checking with TBvision"*. Disponible en: http://www.ldra.com/sec-c.asp. Consultado el 27 de marzo de 2011.

[Marciniak, 1994] Marciniak, John J. 1994. Páginas 1341 y 1349 en *"Encyclopedia of software engineering"*. Vol II. John Wiley & Sons.

[Marineau-Mes, 2005] Marineau-Mes, Sebastian. 2005. *"A bit of History"*. Disponible en: http://www.eclipsecon.org/2005/presentations/EclipseCon05_8.2.pdf. Consultado el 28 de febrero de 2011.

[MIRA, 2009a] MIRA Ltd. 2009. *"A Brief history of MISRA"*. Disponible en: http://www.misra.org.uk/MISRAHome/AbriefhistoryofMISRA/tabid/69/Default.aspx. Consultado el 17 de febrero de 2011.

[MIRA, 2009b] MIRA Ltd. 2009. *"MISRA C FAQ"*. Disponible en: http://www.misra-c.com/MISRACFAQ/tabid/168/Default.aspx. Consultado el 20 de febrero de 2011.

[Nagappan y Ball, 2005] Nagappan, Nachiappan., y Ball, Thomas. 2005. *"Static Analysis Tools as Early Indicators of Pre-Release Defect Density"*. ACM.

[Perry, 1999] Perry, William. 1999. Página 258 en *"Year 2000 Software Testing"*. Wiley Computer Publishing.

[RAE, 2010] Real Academia Española. 2010. *Diccionario de la Lengua Española*. Disponible en: http://www.rae.es/rae.html. Consultado el 22 de junio de 2011.

[Recoskie et. al., 2007] Recoskie, Chris y Tibbitts, Beth. 2007. *"C/C++ Source Code Introspection Using the CDT"*. Disponible en: http://dev.eclipse.org/viewcvs/viewvc.cgi/org.eclipse.ptp/doc/presentations/EclipseCon_2007_C_and_C%2B%2B_Code_Introspection_Using_The_CDT.pdf?root=Tools_Project&view=co. Consultado el 28 de febrero de 2011.

[Ritchie, 1993] Ritchie, Dennis M. 1993. "*The Development of the C Language*". Disponible en: http://cm.bell-labs.com/cm/cs/who/dmr/chist.html. Consultado el 15 de febrero de 2011.

[Schach, 2007] Schach, Stephen R. 2007. Páginas 5 y 96 de"*Object Oriented & Classical Software Engineering*". Séptima Edición. McGraw-Hill.

[Schaefer, 2010] Schaefer, Doug. 2010. "*Doug on the Eclipse CDT*". Disponible en: http://cdtdoug.blogspot.com/2010/02/ok-eclipse-you-have-3-seconds.html. Consultado el 2 de junio de 2011.

[Schorn, 2006] Schorn, Markus. 2006. "*Using CDT APIs to programmatically introspect C/C++ code*". Disponible en: http://wiki.eclipse.org/images/c/c7/CDT_APIs_for_code_introspection.pdf. Consultado el 28 de febrero de 2011.

[Scott, 2006] Scott, Michael L. 2006. Páginas 5-6 en "*Programming Language pragmatics*". Segunda edición. Morgan Kaufmann Publishers.

[Sommerville, 2007] Sommerville, Ian. 2007. Páginas 7, 516, 518, 520, 527, 528-530 en "*Software Engineering*". Octava Edición. Addison Wesley.

[The Eclipse foundation, 2011a] The Eclipse foundation. 2011. "*About the Eclipse Foundation*". Disponible en: http://www.eclipse.org/org/. Consultado el 28 de febrero de 2011.

[The Eclipse foundation, 2011b] The Eclipse foundation. 2011. "*What is Eclipse?*". Disponible en: http://www.eclipse.org/home/newcomers.php. Consultado el 28 de febrero de 2011.
[The Eclipse foundation, 2011c] The Eclipse foundation. 2011. "*Eclipse Projects*". Disponible en: http://www.eclipse.org/projects/. Consultado el 28 de febrero de 2011.

[The Eclipse foundation, 2011d] The Eclipse foundation. 2011. "*Awards and opinions*". Disponible en: http://www.eclipse.org/community/awards.php. Consultado el 28 de febrero de 2011.

[The Eclipse foundation, 2011e] The Eclipse foundation. 2011. "*Eclipse C/C++ Development Tooling*". Disponible en: http://www.eclipse.org/cdt/. Consultado el 28 de febrero de 2011.

[Walters, 2005] Walters, John K. 2005. "*Eclipse CDT: Reshaping the C/C++ Tools Market*". Disponible en: http://adtmag.com/articles/2005/09/14/eclipse-cdt-reshaping-the-cc-tools-market.aspx. Consultado el 28 de febrero de 2011.

[Whitbread, 2009] Whitbread, Martin. 2009. *"Standardizing on a common embedded Eclipse IDE"*. Disponible en: http://www.embedded.com/219501492. Consultado el 28 de febrero de 2011.

[Zheng et. al., 2006] Zheng, Jiang., Williams, Laurie., Nagappan, Nachiappan., Snipes, Will., Hudelpohl, John P., y Vouk, Mladen A. 2006. *"On the Value of Static Analysis for Fault Detection in Software"*. IEEE.

GLOSARIO DE TERMINOS

API: interfaz de programación de aplicaciones. Se le llama así al grupo de métodos o rutinas que un módulo tiene disponible para la comunicación e interacción con otros módulos.

CAN: "Controller Area Network". Protocolo de comunicación serial típicamente empleado en el sector automotriz para la comunicación en red de múltiples dispositivos.

IDE: Ambiente integral de desarrollo. Es un entorno para la creación de aplicaciones dentro del cual pueden realizarse la mayor parte de las actividades necesarias para el desarrollo de software. Desde acceder a los requerimientos, diseñar, codificar, hasta depurar y liberar software.

Parser: modulo encargado del reconocimiento léxico y del análisis sintáctico y semántico de una entrada determinada en base a una gramática específica.

ANEXOS

ANEXO A

FORMATO DE EVALUACIÓN DE ANESINE

1. ¿Ha empleado previamente otras herramientas de análisis estático?
    a. Si:_____
    b. No
    c. Muy poco
    d. Esta es mi primer experiencia
2. ¿Cómo considera la configuración requerida por la herramienta?
    a. Trivial
    b. Sencilla y rápida
    c. Similar a la de otras herramientas
    d. Complicada y tardada
    e. Realmente tediosa y compleja
    f. Otra_____
3. ¿Cómo considera la forma en que se invoca ANESINE dentro del editor de código?
    a. Transparente al usuario
    b. Innovadora
    c. Sencilla
    d. Similar a la de otras herramientas
    e. Un dolor de cabeza
    f. Otra_____
4. Si reportó falsos positivos ¿En qué regla(s) ocurrieron y cuántos?
    a. Item1:_____
    b. Item2:_____
    c. Item3:_____
5. Si reportó falsos negativos ¿En qué regla(s) ocurrieron y cuántos?
    a. Item1:_____
    b. Item2:_____
    c. Item3:_____
6. En general ¿Cómo considera los errores potenciales reportados por ANESINE?
    a. Acertados
    b. Infundados
    c. Distracciones
    d. Totalmente falsos
    e. Otra_____
7. ¿Enfrentó algún problema con ANESINE? ¿Cuál(es)?
    a. Si:_____ _____
    b. No. Ninguno
8. ¿Qué sugeriría cambiar de ANESINE?
    a. _____ _____
9. En general ¿Qué le pareció la experiencia evaluando la herramienta?
    a. Total pérdida de tiempo
    b. Indiferente
    c. Interesante y prometedora
    d. Otra_____

ANEXO B

Herramienta	***Tipo de análisis estático***	***Ventajas***	***Desventajas***
BLAST (Licencia Apache 2.0) *http://mtc.epfl.ch/software-tools/blast/index-epfl.php*	Checa que la características de comportamiento se cumplan	Puede ser utilizado para verificar que se hace un uso de la memoria seguro combinándolo con otra utilerías Puede utilizarse para generar casos de prueba Plug-in para Eclipse disponible que requiere Cygwin	Modela los enteros como enteros matemáticos Asume que toda la aritmética de apuntadores es segura Checa únicamente contra especificación Las entradas de BLAST son otros programas en C
Clang (Open Source) *http://clang-analyzer.llvm.org/available_checks.html*	Fugas de memoria Liberaciones dobles de memoria Malas referencias a apuntadores	Se pueden emitir mejoras o contribuir Bien integrado dentro de MAC OS X Podría ser compilado para otros sistemas ya que el código está disponible	Es más lento incluso que la compilación Desarrollo todavía en progreso No checa contra un estándar de codificación específico
Frama-C (Open Source) *http://frama-c.cea.fr/index.html*	Checa contra especificación funcional	Arquitectura extensible Lenguaje para especificación funcional Interfaz gráfica y de línea de comandos	Se requiere escribir la especificación funcional contra la cual verificar en lenguaje ACSL lo cual puede llegar a ser complicado y tedioso Depende de extensiones

			disponibles lo que se puede verificar No está integrada a una IDE tiene su propia interfaz gráfica Está en desarrollo
Splint (Open Source) *http://www.splint.org/*	Utilización de apuntadores nulos Almacenaje no propiamente definido Inconsistencia de tipos Violaciones al encapsulamiento de información Problemas de administración de memoria Alias peligrosos Modificaciones y usos de variables globales inconsistentes con las interfaces Ciclos posiblemente infinitos Desbordamiento de búferes Implementaciones e invocaciones de macros peligrosas Violaciones de	Numerosas reglas de chequeo Código abierto Configuración adicional mediante anotaciones en comentarios en el código	Utilería de sistemas Unix Interfaz exclusiva de línea de comandos No integrada a una IDE No checa contra un estándar de codificación específico

	convenciones de nombres configurables		
LDRA Testbed (comercial) *http://www.ldra.com/staticanalysis.asp*	Verificación contra varios estándares de codificación (MISRA C/MISRA-C:2004 y DERA C) Métricas Halstead Estructuración del código Complejidad ciclomática, Knots Complejidad ciclomática esencial Esencial Knots Código inalcanzable Análisis de flujo de datos estático Análisis del flujo de información Análisis de ciclos Análisis de funciones recursivas Análisis de interfases entre funciones	Aceptado para el análisis de aplicaciones críticas Estándares de codificación a verificar específicos y seleccionables También puede hacer análisis dinámico, pruebas unitarias, pruebas de integración y pruebas de sistema Posible invocación desde Eclipse Reportes HTML	No integrado a una IDE tiene su propia interfaz Correrlo puede tomar más que compilar el código Necesita reformatear una copia del código fuente ya que no trabaja sobre el código original Costo elevado
Goanna (comercial) *http://redlizards.com/produc*	Comparación negativa de un valor sin signo Posible división por cero Límites de arreglos	Numerosos chequeos Puede correr individualmente durante el desarrollo o en todo el proyecto	En el ambiente Windows está disponible sólo para Visual Studio Integración a Eclipse sólo disponible para Linux

ts.html	Aritmética de apuntadores con malloc Liberación de memoria inconsistente Referencia a apuntadores posiblemente nulos Uso de memoria ya liberada Mal uso de apuntadores Problemas en ciclos Código inútil y código muerto Comportamiento no especificado	cuando se compila Chequeos activos configurables Integración a Visual Studio y a Eclipse	No checa contra un estándar de codificación específico La licencia expira anualmente (costo anual individual $749)

Tabla 15. Herramientas existentes de análisis estático

HOJA DE DATOS DEL MICRO-CONTROLADOR V850 DX4

NEC

Draft User Manual

V850E2/Dx4(-H)

32-bit Single-Chip Microcontroller

Important User Information

This is a draft version of the V850E2/Dx4(-H) User Manual. The user/reader needs to be aware about the following facts:

- **During draft status of the User Manual the content might change without further notice or indication. Thus this draft document may not contain a maintained revision history**
- **Some product features may not be described in this draft document.**
- **Some descriptions of the product features in this draft document may need to be corrected.**

Document No. U20018EE0V3UM00

Date Published March 31, 2010

Printed in Germany

NEC

Draft User Manual

V850E2/Dx4(-H)

32-bit Single-Chip Microcontroller

µPD70F3522
µPD70F3523
µPD70F3524
µPD70F3525
µPD70F3526
µPD70F3532
µPD70F3535
µPD70F3536
µPD70F3537

Document No. U20018EE0V3UM00
Date Published March 31, 2010

Printed in Germany

Notes for CMOS Devices

(1) Precaution against ESD for semiconductors

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All text and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

(2) Handling of unused input pins for CMOS

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

(3) Status before initialization of MOS devices

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

Legal Notes

- The information in this document is current as of <publishing month year>. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such NEC Electronics products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavours to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: “Standard”, “Special” and “Specific”.

  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated “quality assurance program” for a specific application. The recommended applications of NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

  "Standard": Computers, office equipment, communications equipment, text and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.

  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).

  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

  The quality grade of NEC Electronics products is “Standard” unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact NEC Electronics sales

representative in advance to determine NEC Electronics 's willingness to support a given application.

Note 1. "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

2. "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668, Japan
Tel: 044 4355111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554,l
U.S.A.
Tel: 408 5886000
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211 65030
http://www.eu.necel.com/

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908 691133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01 30675800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091 5042787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 6387200

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02 667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven,
The Netherlands
Tel: 040 2654010

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27
ZhiChunLu Haidian District,
Beijing 100083, P.R.China
Tel: 010 82351155
http://www.cn.necel.com/

NEC Electronics Shanghai Ltd.
Room 2511-2512, Bank of China
Tower,
200 Yincheng Road Central,
Pudong New Area,
Shanghai 200120, P.R. China
Tel: 021 58885400
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
12/F., Cityplaza 4,
12 Taikoo Wan Road, Hong Kong
Tel: 2886 9318
http://www.hk.necel.com/

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R.O.C.
Tel: 02 27192377

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253 8311
http://www.sg.necel.com/

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku, Seoul,
135-080, Korea Tel: 02-558-3737
http://www.kr.necel.com/

Preface

Readers This manual is intended for users who want to understand the functions of the concerned microcontrollers.

Purpose This manual presents the hardware manual for the concerned microcontrollers.

Organization This system specification describes the following sections:

- Pin function
- CPU function
- Internal peripheral function

Module instances These microcontrollers may contain several instances of a dedicated module. In general the different instances of such modules are identified by the index “n”, where “n” counts from 0 to the number of instances minus one.

Legend Symbols and notation are used as follows:

- Weight in data notation: Left is high order column, right is low order column
- Active low notation: $\overline{xxx}$ (pin or signal name is over-scored) or /xxx (slash before signal name)
- Memory map address: High order at high stage and low order at low stage

Note Additional remark or tip

Caution Item deserving extra attention

Numeric notation

- Binary: xxxx or xxx_B
- Decimal: xxxx
- Hexadecimal: $xxxx_H$ or 0x xxxx

Numeric prefixes representing powers of 2 (address space, memory capacity):

- K (kilo): $2^{10} = 1024$
- M (mega): $2^{20} = 1024^2 = 1,048,576$
- G (giga): $2^{30} = 1024^3 = 1,073,741,824$

Register contents X, x = don’t care

Diagrams Block diagrams do not necessarily show the exact wiring in hardware but the functional structure.

Timing diagrams are for functional explanation purposes only, without any relevance to the real hardware implementation.

Further Information For further information see http://www.eu.necel.com.

Table of Contents

Notes for CMOS Devices 5

Legal Notes 6

Regional Information 8

Preface 9

Table of Contents 11

Chapter 1 Introduction 35

1.1 V850E2/Dx4(-H) products features 35

1.2 Related Documents 45

1.3 Ordering Information 45

Chapter 2 Port Functions 47

2.1 V850E2/Dx4(-H) Port Features 47

2.2 Overview 48

2.2.1 Terms 49

2.2.2 Pin function configuration 50

2.2.3 Pin data input/output 51

2.2.4 Port control logic diagram 54

2.2.5 Writing to protected registers 55

2.3 Port Group Configuration Registers 56

2.3.1 Port control registers overview 56

2.3.2 Pin function configuration registers 58

2.3.3 Pin data input/output registers 66

2.3.4 Configuration of electrical characteristics registers 71

2.4 V850E2/Dx4(-H) Port Groups Configuration 80

2.4.1 Port registers protection clusters 80

2.4.2 Common port functions 81

2.4.3 V850E2/Dx4(-H) μPD70F3522, μPD70F3523 port functions 87

2.4.4 V850E2/Dx4(-H) μPD70F3524, μPD70F3525, μPD70F3526 port functions 101

2.4.5 V850E2/Dx4(-H) μPD70F3532 port functions 114

2.4.6 V850E2/Dx4(-H) μPD70F3535, μPD70F3536 port functions 133

2.4.7 V850E2/Dx4(-H) μPD70F3537 port functions 153

2.4.8 Non-port input/output signals 172

2.4.9 Alphabetic pin function list 173

2.4.10 Port functions during/after reset and in stand-by modes 189

2.4.11 Recommended connection of unused pins 189

2.5 Port Filters 190

2.5.1 Port filters assignment 190

2.5.2 Port filters clock supply 195

2.6 Port Filters Functional Description 197

2.6.1 Analog filters 197

2.6.2 Digital filters 200

2.6.3 Filter control registers . . . 203

Chapter 3 CPU System Function . . . 207

3.1 Overview . . . 207
3.1.1 Peripheral Protection Unit . . . 209
3.1.2 Timing Supervision Unit . . . 212
3.2 Operation Modes . . . 213
3.2.1 Normal operation mode . . . 213
3.2.2 Flash programming mode . . . 213
3.2.3 Boundary Scan mode . . . 213
3.3 Address Space . . . 214
3.3.1 CPU data address and physical program address space . . . 214
3.3.2 Program and data space . . . 214
3.4 V850E2/Dx4(-H) Address Map . . . 216
3.4.1 DMA address map . . . 216
3.4.2 V850E2/Dx4(-H) CPU address maps . . . 217
3.4.3 Memory areas . . . 221
3.5 Back-up RAM (BURAM) . . . 224
3.5.1 Back-up RAM protection . . . 225
3.6 CPU Subsystem - HBUS Bridge (μPD70F353x) . . . 227
3.6.1 Functional description . . . 227
3.6.2 Register overview . . . 229
3.6.3 Register details . . . 230
3.7 Write protected Registers . . . 239
3.7.1 Register protection clusters . . . 239
3.7.2 Register protection unlock sequence . . . 240
3.7.3 Register protection and interrupt/emulation break . . . 241
3.7.4 V850E2/Dx4(-H) write protected registers . . . 242
3.7.5 V850E2/Dx4(-H) Protection registers overview . . . 244
3.7.6 Control protection clusters registers details . . . 246
3.7.7 Clock monitors protection cluster registers details . . . 247
3.7.8 Port protection clusters registers details . . . 248
3.7.9 Self-programming protection cluster registers details . . . 249

Chapter 4 μPD70F353x Bus Architecture . . . 251

4.1 Overview . . . 251
4.2 Bus Systems . . . 252
4.2.1 XBUS - Cross Connection Bus system . . . 252
4.3 Memory Map . . . 257
4.3.1 CPU memory map . . . 257
4.3.2 XBUS memory map . . . 258
4.4 Memory and Access Protection . . . 260
4.4.1 CPU memory and access protection . . . 260
4.4.2 GMPU protected data areas . . . 260
4.4.3 XBUS Cross Connection Bus data access protection (GMPU) . . . 261
4.5 Bus access exceptions controller . . . 276

Chapter 5 Interrupt Functions . . . 287

5.1 Features . . . 287

5.2 V850E2/Dx4(-H) Interrupt Requests . . . 290
5.2.1 V850E2/Dx4(-H) interrupt sources . . . 290
5.2.2 V850E2/Dx4(-H) FE level non-maskable interrupt sharing . . . 314
5.2.3 V850E2/Dx4(-H) EI level maskable interrupt sharing . . . 316
5.2.4 V850E2/Dx4(-H) DMA interrupt selection . . . 319
5.3 Edge Detection Configuration . . . 320
5.4 Interrupt Controller Control Registers . . . 321
5.5 Interrupt Acknowledgment and Restoring . . . 330
5. 5. 1 FE level non-maskable interrupt caused by FENMI interrupt request . . . 330
5. 5. 2 Restore from FE level non-maskable interrupt (FENMI) . . . 332
5. 5. 3 FE level maskable interrupt caused by FEINT interrupt request . . . 332
5. 5. 4 Restore from FE level maskable interrupt (FEINT) servicing . . . 334
5. 5. 5 EI level maskable interrupt caused by EIINT interrupt request . . . 335
5. 5. 6 Restore from EI level maskable interrupt (EIINT) . . . 337
5.6 Interrupt Operation . . . 338
5. 6. 1 Mask function of EI level maskable interrupt (EIINT) . . . 338
5. 6. 2 Interrupt priority level judgment . . . 338
5. 6. 3 Priority mask function . . . 344
5. 6. 4 Pending interrupt report function . . . 344
5. 6. 5 In-service priority clear function . . . 345
5.7 Exception Handler Address Switching Function . . . 345

Chapter 6 DMA/DTS Controller (DMAC) . . . 347
6.1 V850E2/Dx4(-H) DMA/DTS Features . . . 347
6.2 Definition of Terms . . . 352
6.3 General . . . 353
6.3.1 DMA controller (DMAC) function . . . 353
6.3.2 DMA trigger factor register (DTFR) function . . . 353
6.3.3 Data transfer service (DTS) function . . . 353
6.3.4 DTS factor selector (DTSFSL) function . . . 353
6.3.5 DMA access memory map . . . 355
6.3.6 Prioritization of channels . . . 355
6.3.7 Arbitration of transfer requests . . . 356
6.3.8 Stand-by function . . . 357
6.4 DMAC Function . . . 358
6.4.1 Features . . . 358
6.4.2 DMAC setting registers . . . 360
6.4.3 Enabling or disabling writing control registers . . . 370
6.5 DMA Control Registers . . . 371
6.5.1 DTRCx – DMA transfer request control register (x = 0, 1) . . . 371
6.5.2 DTRSn – DMA transfer request select register . . . 372
6.5.3 DSAnL – DMA source address register L . . . 373
6.5.4 DSAnH – DMA source address register H . . . 374
6.5.5 DSCn – DMA source chip select register . . . 375
6.5.6 DNSAnL – DMA next source address register L . . . 376
6.5.7 DNSAnH – DMA next source address register H . . . 377
6.5.8 DNSCn – DMA next source chip select register . . . 378
6.5.9 DDAnL – DMA destination address register L . . . 379
6.5.10 DDAnH – DMA destination address register H . . . 381
6.5.11 DDCn – DMA destination chip select register . . . 382

6.5.12 DNDAnL – DMA next destination address register L 383
6.5.13 DNDAnH – DMA next destination address register H 384
6.5.14 DNDCn – DMA next destination chip select register 385
6.5.15 DTCn – DMA transfer count register 386
6.5.16 DNTCn – DMA next transfer count register 387
6.5.17 DTCCn – DMA transfer count compare register 388
6.5.18 DTCTn – DMA transfer control register 389
6.5.19 DTSn – DMA transfer status register 391
6.6 DMAC Function Details 393
6.6.1 DMAC transfer setting flow 393
6.6.2 DMAC transfer modes 395
6.6.3 DMAC channel priority control 398
6.6.4 Valid DMA transfer request conditions 399
6.6.5 Next address function 400
6.6.6 Aborting/resuming DMA transfer 401
6.6.7 Error response support 402
6.6.8 Stand-by support 402
6.7 DTFR Function 403
6.7.1 Features 403
6.8 DTFR Control Registers 404
6.8.1 DTFRn – DTFRn register 404
6.8.2 DRQCLR – DMA request clear register 405
6.8.3 DRQSTR – DMA request check register 406
6.9 DTS Function 407
6.9.1 Features 407
6.10 DTS Control Registers 410
6.10.1 DTS0TSR – DTS transfer status register [register group A] 411
6.10.2 DTS0TRC – DTS transfer request control register [register group A] 412
6.10.3 DTS0ICR – DTS initialization control register [register group A] 413
6.10.4 DTS0BTR – DTS base table register [register group A] 415
6.10.5 DTS0BVR – DTS base vector register [register group A] 416
6.10.6 DTS0ACR – DTS active channel register [register group A] 417
6.10.7 DTS0TST – DTS TI hold status register [register group A] 418
6.10.8 DTS0HC – DTS TI hold channel number register [register group A] 419
6.10.9 DTS0SAR – DTS source address register [register group B] 421
6.10.10 DTS0DAR – DTS destination address register [register group B] 422
6.10.11 DTS0TCEA – DTS transfer counter or else address register [register group B] 423
6.10.12 DTS0SCS – DTS source address count size register [register group B] . . 425
6.10.13 DTS0DCS – DTS destination address count size register [register group B] . . 426
6.10.14 DTS0CIR – DTS control information register [register group B] 427
6.10.15 DTS0ECSRA – DTS extension address count size/repeat address registers [register group B] 429
6.11 DTS Function Details 431
6.11.1 Transfer information (TI) 431
6.11.2 DTS transfer setting flow 434
6.11.3 Basic operations of DTS 435
6.11.4 Transfer modes 436
6.11.5 Transfer types 437

6.11.6 Special functions . . . 453
6.11.7 Transfer count . . . 455
6.11.8 Chain function . . . 455
6.11.9 TI hold function . . . 457
6.11.10 Interrupt output function . . . 458
6.11.11 Interrupt and chain of transfers during block transfer . . . 458
6.11.12 TI write back skip function . . . 459
6.11.13 DTS channel priority control . . . 460
6.11.14 Valid DTS transfer request conditions . . . 460
6.11.15 Aborting/resuming DTS transfer . . . 460
6.11.16 Error response support . . . 461
6.11.17 Stand-by support . . . 462
6.12 DTSFSL Function . . . 463
6.12.1 Features . . . 463
6.13 DTSFSL Control Registers . . . 464
6.13.1 DTSENn – DTS transfer enable register . . . 464
6.13.2 DTSHENm – DTS hold enable register (m = 0 to 3) . . . 465
6.14 DTSFSL Function Details . . . 467
6.14.1 Interrupt output function . . . 467

Chapter 7 Flash Memory . . . 469

7.1 Code Flash Memory Overview . . . 470
7.1.1 Code flash memory features . . . 470
7.1.2 Code flash memory mapping . . . 471
7.1.3 Data flash memory map . . . 472
7.2 Code Flash Memory functional Outline . . . 473
7.2.1 Code flash memory erasure and rewrite . . . 476
7.3 Data Flash Memory . . . 477
7.3.1 Data flash memory features . . . 477
7.3.2 Data flash writing . . . 477
7.4 Flash Programming with Flash Programmer . . . 478
7.4.1 Programming environment . . . 478
7.4.2 Communication modes . . . 479
7.4.3 Pin connection with flash programmer PG-FP5 . . . 481
7.4.4 Flash memory programming control . . . 482
7.5 Code Flash Self-Programming . . . 489
7.5.1 Self-programming enable . . . 490
7.5.2 Self-programming library functions . . . 491
7.5.3 Self-programming internal RAM occupancy . . . 491
7.5.4 Secure self-programming (boot cluster swapping) . . . 492
7.5.5 Interrupt handling during flash self-programming . . . 496
7.6 Flash Mask Options . . . 497
7.6.1 OPBT0 - Flash mask option register 0 . . . 499

Chapter 8 Clock Controller . . . 501

8.1 Clock Controller Overview . . . 501
8.2 General Description of Clock Generation and Control . . . 504
8.2.1 Clock generators . . . 506
8.2.2 Clock selectors . . . 507

8.3 Clock Generators . . . 509
8.3.1 Main Oscillator (MainOsc) clock generator . . . 509
8.3.2 Sub Oscillator (SubOsc) clock generator . . . 512
8.3.3 Low Speed Internal Oscillator (Low Speed IntOsc) clock generator . . . 514
8.3.4 High Speed Internal Oscillator (High Speed IntOsc) clock generator . . . 515
8.3.5 Phase-Locked Loop (PLL) clock generators . . . 517
8.3.6 Writing to protected registers . . . 520
8.4 Clock Selection . . . 521
8.4.1 Clock domains of Always–On–Area . . . 524
8.4.2 Clock domains of Isolated–Area-0 . . . 530
8.4.3 Clock domains of Isolated–Area-1 . . . 535
8.5 Frequency Output Function (FOUT) . . . 550
8.5.1 FOUT Clock Divider (FOUTDIV) . . . 551
8.5.2 FOUT Clock Divider registers overview . . . 551
8.5.3 FOUT Clock Divider control registers details . . . 552
8.6 Clock Monitor A (CLMA) . . . 554
8.6.1 V850E2/Dx4(-H) CLMA features . . . 554
8.6.2 CLMA enable and start-up options . . . 557
8.6.3 Functional Overview . . . 558
8.6.4 Functional Description . . . 559
8.6.5 Writing to protected registers . . . 562
8.6.6 Clock Monitor Registers . . . 563
8.7 Clock Controller Registers . . . 568
8.7.1 Clock Controller registers overview . . . 568
8.7.2 Clock generators registers . . . 569
8.7.3 Clock selector control register . . . 584

Chapter 9 Stand-by Controller (STBC) . . . 587
9.1 V850E2/Dx4(-H) Stand-by Controller Features . . . 587
9.2 Stand-by Controller functions . . . 590
9.2.1 Wake-up . . . 593
9.2.2 I/O buffer control . . . 596
9.2.3 Power save mode transitions . . . 598
9.2.4 Power save mode entry and exit example flows . . . 599
9.2.5 Writing to protected registers . . . 610
9.2.6 Stand-by Controller registers overview . . . 611
9.2.7 Stand-by Controller control registers details . . . 612
9.2.8 Wake-up event controller registers details . . . 620
9.2.9 Oscillator wake-up mask registers details . . . 623

Chapter 10 Code Protection and Security . . . 625
10.1 Overview . . . 625
10.2 Flash Programmer and Self-Programming Protection . . . 626
10.3 On-Chip Debug Interface Protection . . . 627
10.3.1 On-Chip Debug enable flag . . . 627
10.3.2 On-Chip Debug ID code . . . 628
10.3.3 On-Chip Debug protection levels summary . . . 628
10.3.4 On-Chip Debug control registers . . . 629
10.4 Flash Writer and Self-Programming Protection . . . 632

Chapter 11 Reset Controller 633
11.1 Functional Overview 633
11.2 Functional Description 637
11.2.1 Reset flags 637
11.2.2 Power-On Clear (POC) 638
11.2.3 Low-Voltage Indicator (LVI) 639
11.2.4 Very-Low-Voltage Indicator (VLVI) 640
11.2.5 External RESET 641
11.2.6 Watchdog Timers reset 642
11.2.7 Software reset 642
11.2.8 Clock Monitors reset 642
11.2.9 Debugger reset 643
11.2.10 I/O buffer reset 643
11.2.11 PowerGood indicator 643
11.2.12 Writing to protected registers 644
11.3 Registers 645
11.3.1 Reset Controller registers overview 645
11.3.2 Reset Controller general control registers details 646
11.3.3 Software reset control registers details 649
11.3.4 Low-Voltage Indicator reset control registers 651
11.3.5 PowerGood reset flag registers 652
11.3.6 Very Low-Voltage flag control registers 653

Chapter 12 OS Timer (OSTM) 655
12.1 V850E2/Dx4(-H) OSTM Features 655
12.2 Functional Overview 657
12.3 Functional Description 658
12.3.1 Count clock 658
12.3.2 Output modes 659
12.3.3 Interrupt request generation 660
12.3.4 Starting and stopping the timer 661
12.3.5 Interval timer mode 661
12.3.6 Free-run compare mode 665
12.4 OS Timer Registers 668
12.4.1 OS Timer registers overview 668
12.4.2 OS Timer registers details 668

Chapter 13 Window Watchdog Timer A (WDTA) 675
13.1 V850E2/Dx4(-H) WDTA Features 675
13.2 WDTA Start-up Options 678
13.2.1 V850E2/Dx4(-H) WDTAn start modes 679
13.3 Functional Overview 680
13.4 Functional Description 681
13.4.1 WDTA after reset release 682
13.4.2 WDTA trigger 685
13.4.3 Error detection 686
13.4.4 75% interrupt output 688
13.4.5 Window function 689
13.5 Evaluation of the Watchdog status 690

13.6 Registers . . . 691
13.6.1 WDTA registers overview . . . 691
13.6.2 WDTA registers details . . . 692

Chapter 14 Timer Array Unit A (TAUA) . . . 699
14.1 V850E2/Dx4(-H) TAUA Features . . . 699
14.2 TAUA Input Selection . . . 708
14.2.1 TAUA0 input selection . . . 708
14.2.2 TAUA4 input selection . . . 712
14.3 Functional Overview . . . 714
14.3.1 Terms . . . 716
14.4 Functional Description . . . 717
14.5 General Operating Procedure . . . 719
14.6 Operation Modes . . . 720
14.7 Concepts of Synchronous Channel Operation . . . 721
14.7.1 Rules . . . 721
14.7.2 Simultaneous start and stop of synchronous channel counters . . . 723
14.8 Simultaneous Rewrite . . . 724
14.8.1 Introduction . . . 724
14.8.2 How to control simultaneous rewrite . . . 726
14.8.3 Other general rules of simultaneous rewrite . . . 727
14.8.4 Types of simultaneous rewrite . . . 728
14.9 Channel Output Modes . . . 736
14.9.1 General procedure for specifying a channel output mode . . . 738
14.9.2 Channel output modes controlled independently by TAUAn signals . . . 739
14.9.3 Channel output modes controlled synchronously by TAUAn signals . . . 741
14.10 Start Timing of Operating Modes . . . 746
14.10.1 Interval Timer Mode, Judge Mode, Capture Mode, Up Down Count Mode . . . 746
14.10.2 Event Mode . . . 747
14.10.3 All other operating modes . . . 747
14.11 TAUAnTTOUTm Output and INTTAUAnIm Generation when Counter Starts or Restarts . . . 748
14.12 Interrupt Generation upon Overflow . . . 750
14.12.1 Capture Mode . . . 751
14.12.2 Capture and One Count Mode . . . 752
14.12.3 Count Capture Mode . . . 753
14.12.4 Capture and Gate Count Mode . . . 754
14.13 TAUAnTTINm Edge Detection . . . 755
14.14 Assigning DMA Window Addresses . . . 756
14.15 Independent Channel Operation Functions . . . 757
14.16 Independent Channel Interrupt Functions . . . 757
14.16.1 Interval Timer Function . . . 758
14.16.2 TAUAnTTINm Input Interval Timer Function . . . 765
14.16.3 Delay Count Function . . . 771
14.16.4 One-Pulse Output Function . . . 776
14.17 Independent Channel Signal Measurement Functions . . . 781
14.17.1 TAUAnTTINm Input Pulse Interval Measurement Function . . . 782
14.17.2 TAUAnTTINm Input Signal Width Measurement Function . . . 789
14.17.3 Overflow Interrupt Output Function (During TAUAnTTINm Width Measurement) . . . 796

14.17.4 TAUAnTTINm Input Period Count Detection Function. 801
14.17.5 Overflow Interrupt Output Function (During TAUAnTTINm Input Period Count Detection) 807
14.17.6 TAUAnTTINm Input Pulse Interval Judgment Function 812
14.17.7 TAUAnTTINm Input Signal Width Judgment Function. 817
14.18 Independent Channel Real-Time Functions 822
14.18.1 Real-Time Output Function Type 1 823
14.18.2 Real-Time Output Function Type 2 830
14.19 Independent Channel Simultaneous Rewrite Functions 837
14.19.1 Simultaneous Rewrite Trigger Generation Function Type 1 838
14.19.2 Simultaneous Rewrite Trigger Generation Function Type 2 844
14.20 Independent Channel One-Phase PWM Function 850
14.20.1 One-Phase PWM Output Function 851
14.21 Other Independent Channel Functions 858
14.21.1 External Event Count Function 859
14.21.2 Clock Divide Function 866
14.21.3 TAUAnTTINm Input Position Detection Function. 873
14.22 Synchronous Channel Operation Functions 879
14.23 Synchronous PWM Signal Functions Triggered at Regular Intervals 879
14.23.1 PWM Output Function. 880
14.23.2 Trigger Start PWM Output Function 891
14.23.3 Delay Pulse Output Function. 902
14.23.4 AD Conversion Trigger Output Function Type 1 918
14.24 Synchronous PWM Signal Functions Triggered by an External Signal 920
14.24.1 One-Shot Pulse Output Function 921
14.24.2 Offset Trigger Output Function 933
14.25 Synchronous Triangle PWM Functions 943
14.25.1 Triangle PWM Output Function 944
14.25.2 Triangle PWM Output Function with Dead Time 955
14.25.3 AD Conversion Trigger Output Function Type 2 978
14.26 Synchronous Real-Time Output Functions 980
14.26.1 Synchronous Real-Time Output Function Type 1 981
14.26.2 Synchronous Real-Time Output Function Type 2 992
14.26.3 Synchronous Real-Time Output Function Type 3 1003
14.27 Synchronous Non-Complementary and Complementary Functions 1014
14.27.1 Non-Complementary Modulation Output Function Type 1 1015
14.27.2 Non-Complementary Modulation Output Function Type 2 1028
14.27.3 Complementary Modulation Output Function 1042
14.28 Other Synchronous Channel Functions 1060
14.28.1 Interrupt Culling Function 1060
14.29 Registers 1069
14.29.1 TAUAn registers overview 1069
14.29.2 TAUAn prescaler registers details 1071
14.29.3 TAUAn control registers details 1074
14.29.4 TAUAn output registers details 1084
14.29.5 TAUAn channel output level registers details 1090
14.29.6 TAUAn simultaneous rewrite register details 1091
14.29.7 TAUAn DMA window registers. 1094
14.29.8 TAUAn emulation register 1096

Chapter 15 Timer Array Unit J (TAUJ) 1097
15.1 V850E2/Dx4(-H) TAUJ Features 1097
15.2 TAUJ Input Selection 1101
15.2.1 TAUJ0/TAUJ1 input selection 1101
15.2.2 TAUJ2 input selection 1103
15.3 Functional Overview 1104
15.3.1 Terms 1106
15.4 Functional Description 1107
15.5 General Operating Procedure 1109
15.6 Operation Modes 1110
15.7 Concepts of Synchronous Channel Operation 1111
15.7.1 Rules 1111
15.7.2 Simultaneous start and stop of synchronous channel counters 1113
15.8 Simultaneous Rewrite 1114
15.8.1 Introduction 1114
15.8.2 How to control simultaneous rewrite 1115
15.8.3 Other general rules of simultaneous rewrite 1116
15.8.4 Simultaneous rewrite procedure 1117
15.9 Channel Output Modes 1119
15.9.1 General procedure for specifying a channel output mode 1121
15.9.2 Channel output modes controlled independently by TAUJn signals 1122
15.9.3 Channel output modes controlled synchronously by TAUJn signals 1123
15.10 Start Timing of Operating Modes 1124
15.10.1 Interval Timer Mode, Capture Mode 1124
15.10.2 Other operating modes 1125
15.11 TAUJnTTOUTm Output and INTTAUJnIm Generation when Counter Starts or Restarts 1126
15.12 Interrupt Generation upon Overflow 1127
15.12.1 Capture Mode 1128
15.12.2 Capture and One Count Mode 1129
15.12.3 Count Capture Mode 1130
15.12.4 Capture and Gate Count Mode 1131
15.13 TAUJnTTINm Edge Detection 1132
15.14 Independent Channel Operation Functions 1133
15.15 Independent Channel Interrupt Functions 1133
15.15.1 Interval Timer Function 1134
15.15.2 TAUJnTTINm Input Interval Timer Function 1141
15.16 Independent Channel Signal Measurement Fuctions 1147
15.16.1 TAUJnTTINm Input Pulse Interval Measurement Function 1148
15.16.2 TAUJnTTINm Input Signal Width Measurement Function 1155
15.16.3 Overflow Interrupt Output Function (During TAUJnTTINm Width Measurement) 1162
15.16.4 TAUJnTTINm Input Period Count Detection Function 1166
15.16.5 Overflow Interrupt Output Function (During TAUJnTTINm Input Period Count Detection) 1172
15.17 Other Independent Channel Functions 1177
15.17.1 TAUJnTTINm Input Position Detection Function 1178
15.18 Synchronous PWM Signal Functions Triggered at Regular Intervals 1184
15.18.1 PWM Output Function 1185
15.19 Registers 1196

15.19.1 TAUJn registers overview 1196
15.19.2 TAUJn prescaler registers details 1197
15.19.3 TAUJn control registers details 1200
15.19.4 TAUJn output registers details. 1211
15.19.5 TAUJn simultaneous rewrite register details 1214
15.19.6 TAUJn emulation register 1216

Chapter 16 Real-Time Clock (RTCA) 1217
16.1 V850E2/Dx4(-H) RTCA Features 1217
16.2 Functional Overview 1220
16.3 Functional Description 1221
16.3.1 Operation modes. 1222
16.3.2 Clock counter format 1222
16.3.3 Fixed interval interrupt function 1223
16.3.4 Alarm interrupt function 1224
16.3.5 Clock error correction 1225
16.4 Registers 1229
16.4.1 RTCA registers overview. 1229
16.4.2 RTCA control registers details. 1231
16.4.3 RTCA sub-counter registers details 1235
16.4.4 RTCA clock counter and buffer registers details 1239
16.4.5 RTCA special counter and buffer registers details 1254
16.4.6 RTCA alarm setting registers details. 1258
16.4.7 RTCA emulation register details 1261
16.5 Procedures for Setup, Writing and Reading 1262
16.5.1 Initial setting of the RTCA 1262
16.5.2 Updating clock counters 1264
16.5.3 Reading clock counters 1265
16.5.4 Reading RTCAnSRBU 1268
16.5.5 Writing to RTCAnSUBU 1269
16.5.6 Writing to RTCAnSCMP 1270
16.6 Timing diagrams 1271
16.6.1 Timing of RTCA counter start 1271
16.6.2 Timing of RTCA while counter is enabled 1272
16.6.3 Timing of sub-counter buffer read while counter is enabled 1273

Chapter 17 CAN Controller (FCN) 1275
17.1 V850E2/Dx4(-H) FCN Features 1275
17.2 FCN0 and FCN1 connection 1279
17.3 Features 1281
17.3.1 Overview of functions 1282
17.3.2 Configuration. 1283
17.4 Internal Registers of FCN 1284
17.4.1 CAN Controller configuration 1284
17.4.2 CAN Controller Registers Overview 1286
17.4.3 Register bit configuration 1288
17.5 Bit Set/Clear Function 1293
17.6 Control Registers 1295
17.6.1 FCN global registers 1295

17.6.2 FCN module registers 1304
17.6.3 Message buffer registers 1325
17.7 CAN Controller Initialization 1334
17.7.1 Initialization of FCN module 1334
17.7.2 Initialization of message buffer 1334
17.7.3 Redefinition of message buffer 1334
17.7.4 Transition from initialization mode to operation mode 1336
17.7.5 Resetting error counter FCNnCMERCNT of FCN module 1337
17.8 Message Reception 1338
17.8.1 Message reception 1338
17.8.2 Receive data read 1339
17.8.3 Receive history list function 1340
17.8.4 Mask function 1342
17.8.5 Multi buffer receive block function 1344
17.8.6 Remote frame reception 1345
17.9 Message Transmission 1347
17.9.1 Message transmission 1347
17.9.2 Transmit history list function 1349
17.9.3 Automatic block transmission (ABT) 1351
17.9.4 Transmission abort process 1353
17.9.5 Remote frame transmission 1354
17.10 Power Saving Modes 1355
17.10.1 FCN sleep mode 1355
17.10.2 FCN stop mode 1358
17.10.3 Example of using power saving modes 1359
17.11 Interrupt Function 1360
17.12 Diagnosis Functions and Special Operational Modes 1361
17.12.1 Receive-only mode 1361
17.12.2 Single-shot mode 1362
17.12.3 Self-test mode 1363
17.12.4 Receive/transmit operation in each operation mode 1364
17.13 Time Stamp Function 1365
17.13.1 Time stamp function 1365
17.14 Baud Rate Settings 1366
17.14.1 Baud rate setting conditions 1366
17.14.2 Representative examples of baud rate settings 1370
17.15 Operation of the CAN Controller 1374
17.15.1 Initialization 1374
17.15.2 Message transmission 1380
17.15.3 Message reception 1395
17.15.4 Power save modes 1400

Chapter 18 Clocked Serial Interface G (CSIG) 1407
18.1 V850E2/Dx4(-H) CSIG Features 1407
18.2 Functional Overview 1412
18.3 Functional Description 1414
18.3.1 Master/slave mode 1414
18.3.2 Master/slave connections 1415
18.3.3 Transmission clock selection 1417
18.3.4 Data transfer modes 1418

18.3.5 Data length selection. 1419
18.3.6 Serial data direction select function 1421
18.3.7 Communication in slave mode 1422
18.3.8 CSIG interrupts 1423
18.3.9 Handshake function. 1425
18.3.10 Loop-back mode 1428
18.3.11 Error detection. 1429
18.4 CSIG Control Registers. 1432
18.5 Operating Procedure Example 1444

Chapter 19 High-Speed Serial Flash Interface (HSFI) 1447
19.1 V850E2/Dx4(-H) HSFI Features. 1447
19.2 High-Speed Serial Flash Interface Clock Dividers 1453
19.3 Functional Overview 1456
19.4 Functional Description 1458
19.4.1 CSI mode 1458
19.4.2 Flash mode 1464
19.4.3 Soft reset. 1487
19.4.4 Standby 1487
19.5 Registers 1488
19.5.1 HSFI registers overview 1488
19.5.2 HSFI general control registers details 1490
19.5.3 HSFI CSI control registers details 1495
19.5.4 HSFI CS Generator control registers details 1502
19.5.5 HSFI Protocol Unit control registers details 1504
19.5.6 HSFI Access Manager control registers details 1509
19.6 Operating procedures 1518
19.6.1 Single transmission mode (transmission/reception of data) 1518
19.6.2 Reception of data 1520
19.6.3 Continuous transmission mode (transmission/reception of data) 1522
19.6.4 Reception of data with error 1524

Chapter 20 Asynchronous Serial Interface E (URTE) 1527
20.1 V850E2/Dx4(-H) URTEn Features. 1527
20.2 Features 1531
20.3 Configuration. 1532
20.4 URTEn Registers 1533
20.5 Interrupt Request Signals. 1549
20.5.1 Transmission interrupt request INTUAEnTIT. 1549
20.5.2 Reception interrupt request INTUAEnTIR 1550
20.5.3 Status interrupt request INTUAEnTIS 1551
20.5.4 Receive/status interrupt request INTUAEnTRA 1551
20.6 Operation 1551
20.6.1 Data formats 1551
20.6.2 BF transmission/reception format 1553
20.6.3 BF transmission 1555
20.6.4 BF reception 1556
20.6.5 Transmission data consistency check 1558
20.6.6 URTEn transmission 1559

20.6.7 Continuous transmission procedure 1560
20.6.8 URTEn reception 1562
20.6.9 Reception errors 1564
20.6.10 Parity types and operations 1565
20.6.11 Digital receive data noise filter 1566
20.7 Baud Rate Generator 1567

Chapter 21 I2C Interface (IICB) 1569
21.1 V850E2/Dx4(-H) IICB Features 1569
21.2 Functional Overview 1572
21.3 I^2C Bus Mode Functions 1574
21.3.1 Pin configuration 1574
21.4 I^2C Bus Definition 1575
21.4.1 Start Condition 1576
21.4.2 Addresses 1576
21.4.3 Extension code 1577
21.4.4 Transfer direction specification 1577
21.4.5 Acknowledge (ACK) 1578
21.4.6 Data 1579
21.4.7 Stop condition 1579
21.4.8 Wait state 1580
21.4.9 Arbitration 1582
21.5 Registers 1583
21.6 Operation 1606
21.6.1 Single transfer mode 1606
21.6.2 Continuous transfer mode 1611
21.6.3 Arbitration 1616
21.6.4 Entering and exiting wait state 1617
21.6.5 Extension code 1622
21.7 Interrupt Request Signals 1623
21.7.1 Single transfer mode 1624
21.7.2 Continuous transfer mode 1627
21.8 Interrupt Outputs and Statuses 1633
21.8.1 Single transfer mode (master device operation) 1634
21.8.2 Single transfer mode (slave device operation: during slave address reception (IICBnSTR0.IICBnSSC0 bit = 1)) 1637
21.8.3 Single transfer mode (slave device operation: during extension code reception (IICBnSTR0.IICBnSSEX bit = 1)) 1641
21.8.4 Single transfer mode (non-participation in communications) 1645
21.8.5 Single transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1): operation as slave after arbitration loss) 1646
21.8.6 Single transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1): non-participation in communications after arbitration loss) 1648
21.8.7 Single transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1): non-participation in communications after arbitration loss (during extension code transfer)) 1654
21.8.8 Continuous transfer mode (master device operation (reception)) 1655
21.8.9 Continuous transfer mode (master device operation (transmission)) 1658
21.8.10 Continuous transfer mode (slave device operation (reception): during slave address reception (IICBnSTR0.IICBnSSC0 bit = 1)) 1661

21.8.11 Continuous transfer mode (slave device operation (reception): during extension code reception (IICBnSTR0.IICBnSSEX bit = 1)) 1665
21.8.12 Continuous transfer mode (slave device operation (transmission): during slave address reception (IICBnSTR0.IICBnSSC0 bit = 1)). 1669
21.8.13 Continuous transfer mode (slave device operation (transmission): during extension code reception (IICBnSTR0.IICBnSSEX bit = 1)) 1673
21.8.14 Continuous transfer mode (non-participation in communications) 1677
21.8.15 Continuous transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1) (when address was transferred during reception): operation as slave after arbitration loss) 1678
21.8.16 Continuous transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1) (when address was transferred during reception): non-participation in communications after arbitration loss). 1680
21.8.17 Continuous transfer mode (arbitration loss operation (IICBnSTR0.IICBnALDF bit = 1) (when address was transferred during reception): non-participation in communications after arbitration loss (during extension code transfer)) . . 1685
21.9 Setting Sequence 1687
21.9.1 Single master environment 1687
21.9.2 Multi-master environment 1691

Chapter 22 FlexRay™ (FLX) 1699
22.1 V850E2/Dx4(-H) FLXn Features 1699
22.2 E-Ray Overview 1702
22.2.1 Conventions 1702
22.2.2 Definition 1702
22.2.3 References 1702
22.2.4 Terms and abbreviations 1702
22.2.5 Functional overview. 1703
22.2.6 Block diagram 1705
22.2.7 Host CPU interface timing 1707
22.2.8 Reset timing 1707
22.3 Programmer's Model 1708
22.3.1 Register map 1708
22.3.2 E-Ray registers 1714
22.3.3 Special registers 1719
22.3.4 Interrupt registers 1726
22.3.5 CC control registers 1747
22.3.6 CC status registers 1771
22.3.7 Message buffer control registers 1788
22.3.8 Message buffer status registers 1794
22.3.9 Identification Registers 1807
22.3.10 Input buffer 1809
22.3.11 Output buffer 1818
22.4 Functional Description 1829
22.4.1 Communication cycle 1829
22.4.2 Communication modes 1831
22.4.3 Clock synchronization 1832
22.4.4 Error handling 1834
22.4.5 Communication controller states 1836
22.4.6 Network management 1851
22.4.7 Filtering and masking 1851

22.4.8 Transmit process 1854
22.4.9 Receive process 1857
22.4.10 FIFO function 1859
22.4.11 Message handling 1861
22.4.12 Message RAM 1871
22.4.13 Module interrupt 1880
22.5 Appendix 1882
22.5.1 Assignment of FlexRay Configuration Parameters 1882
22.6 Cautions 1884
22.6.1 Loop back mode operates only at 10 MBit/s. 1884
22.6.2 Noise following a dynamic frame that delays idle detection may fail to stop slot counting for the remainder of the dynamic segment 1884
22.6.3 Register FLXnRCV displays wrong value. 1884
22.6.4 After reception of a valid sync frame followed by a valid non-sync frame in the same static slot the received sync frame may be ignored. 1885
22.6.5 Sync frame overflow flag FLXnEIR.FLXnSFO may be set if slot counter is greater than 1024. 1885
22.6.6 Acceptance of startup frames received after reception of more than gSyncNodeMax sync frames. 1887
22.6.7 Initial rate correction value of an integrating node is zero if pMicroInitialOffsetA,B = 00H. 1887
22.6.8 Incorrect rate and/or offset correction value if second Secondary Time Reference Point (STRP) coincides with the action point after detection of a valid frame. 1888
22.6.9 Flag SFS.MRCS is set erroneously although at least one valid sync frame pair is received. 1888
22.6.10 Rate correction set to zero in case of SyncCalcResult=MISSING_TERM. 1889
22.6.11 A sequence of received WUS may generate redundant FLXnSIR.FLXnWUPA/B events. 1889

Chapter 23 I2S Interface (IISA) 1891
23.1 V850E2/Dx4(-H) IISA Features 1891
23.2 I2S Interface clock selector 1894
23.2.1 I2S Clock Dividers 1897
23.2.2 I2S clock inverter 1899
23.3 Functional Overview 1900
23.4 Functional Description 1901
23.4.1 Operation modes 1902
23.4.2 Transmission and reception of data. 1902
23.4.3 Word select signal (IISATWS) 1903
23.4.4 Data length and data format 1903
23.4.5 Transfer format 1905
23.4.6 Generation of interrupt requests and status flags 1906
23.4.7 Basic procedures 1909
23.5 Registers 1911
23.5.1 I^2S Bus Interface registers overview 1911
23.5.2 Control registers details 1912
23.5.3 Status registers details 1916
23.5.4 Data registers details 1920
23.5.5 Emulation register 1921

Chapter 24 PCM-PWM (PCMP) 1923

24.1 V850E2/Dx4(-H) PCMP Features 1923

24.2 Functional Overview 1926

24.3 Functional Description 1927

24.3.1 PCM-coded data input 1928

24.3.2 Mono mode / stereo mode 1929

24.3.3 Output sampling frequency 1930

24.3.4 Output modes 1931

24.3.5 PCM-PWM output generators 1933

24.3.6 Output control 1937

24.3.7 Starting and stopping the PCM-PWM conversion 1938

24.3.8 Example setup 1938

24.4 Registers 1939

24.4.1 PCM-PWM registers overview 1939

24.4.2 PCM-PWM registers details 1940

Chapter 25 LCD Controller/Driver (LCCT) 1951

25.1 V850E2/Dx4(-H) LCD Controller/Driver Features 1951

25.2 Functional Overview 1954

25.3 Functional Description 1955

25.3.1 LCD panel addressing 1955

25.3.2 Common signals and segment signals 1957

25.3.3 Activation of LCD segments 1960

25.3.4 Setting frame frequency and display mode 1961

25.3.5 Cascaded operation 1963

25.3.6 Operation in standby mode 1965

25.4 Display Example 1966

25.5 Timing Diagrams 1969

25.6 LCD Controller/Driver Registers 1975

Chapter 26 LCD Bus Interface (LCBI) 1981

26.1 V850E2/Dx4(-H) LCD Bus Interface Features 1981

26.2 Functional Overview 1984

26.3 Functional Description 1986

26.3.1 Non-TFT operation mode 1987

26.3.2 TFT operation mode 1994

26.3.3 Color usage 1997

26.3.4 Write buffer details 1999

26.3.5 Data transfer types 2001

26.3.6 Soft reset 2005

26.3.7 Change control registers settings 2005

26.4 Registers 2006

26.4.1 LCD Bus Interface registers overview 2006

26.4.2 LCD Bus Interface general control registers details 2008

26.4.3 LCD Bus Interface non-TFT mode control registers details 2019

26.4.4 LCD Bus Interface TFT mode control registers details 2022

26.4.5 LCD Bus Interface data registers details 2026

26.4.6 CLUT Data RAM registers 2030

Chapter 27 A/D Converter A (ADCA) 2031

27.1 V850E2/Dx4(-H) ADCA Features 2031

27.2 H/W trigger expansion 2034

27.2.1 ADCAn H/W trigger selection 2034

27.2.2 ADCAn H/W trigger edge selection 2034

27.2.3 ADCAn H/W trigger tables 2035

27.3 Functional Overview 2038

27.4 Cautions 2040

27.5 Functional Description 2041

27.5.1 Basic Operation 2042

27.5.2 Clock usage 2043

27.5.3 Channels and channel groups 2043

27.5.4 A/D conversion modes 2045

27.5.5 Starting A/D conversion (start trigger modes) 2047

27.5.6 Stopping A/D conversion 2049

27.5.7 Standby mode 2050

27.5.8 Pausing and resuming A/D conversion (ADC-HALT mode) 2051

27.5.9 Resolution, sampling and conversion times 2051

27.5.10 Interrupt generation 2052

27.5.11 Storage of A/D conversion result 2053

27.5.12 Result check functions 2054

27.5.13 Self-diagnosis functions 2056

27.5.14 Discharge function 2060

27.5.15 Buffer amplifier function 2060

27.5.16 Stabilization control 2061

27.6 Registers 2062

27.6.1 ADCA registers overview 2062

27.6.2 Control registers details 2064

27.6.3 Conversion status registers 2073

27.6.4 S/W trigger registers details 2077

27.6.5 ADCA conversion result registers details 2079

27.6.6 ADCA conversion result check registers details 2084

27.6.7 Diagnose functions control registers 2088

27.6.8 Emulation register 2090

Chapter 28 Intelligent Stepper Motor Driver (ISM) 2091

28.1 V850E2/Dx4(-H) ISM Features 2091

28.2 Functional Overview 2094

28.3 Functional Description 2096

28.3.1 Functional description stepper motor driving 2096

28.3.2 Functional description zero point detection 2098

28.3.3 Clocking 2101

28.3.4 Channel management 2101

28.3.5 Generation and processing of PWM signals to drive a motor 2104

28.3.6 Stepper motor driving 2111

28.3.7 Zero Point Detection (ZPD) 2116

28.3.8 PWM value look-up tables and ZPD tables in RAM 2129

28.3.9 Timing 2137

28.3.10 Interrupt requests and status flags 2142

28.3.11 Soft reset 2144
28.3.12 Procedures 2145
28.4 Intelligent Stepper Motor Driver Registers 2150
28.4.1 Intelligent Stepper Motor Driver registers overview 2150
28.4.2 Common control registers 2152
28.4.3 ISMn movement calculator parameters 2165
28.4.4 ISMn movement calculator variables 2167
28.4.5 ZPD registers 2169
28.4.6 ISMn emulation mode register 2180

Chapter 29 Sound Generator (SG) 2181
29.1 V850E2/Dx4(-H) SG Features 2181
29.2 Functional Overview 2183
29.2.1 Description 2184
29.2.2 Principle of operation 2185
29.3 Functional Description 2188
29.3.1 Generating the tone 2188
29.3.2 Generating the volume information 2189
29.3.3 Updating the register buffer values 2191
29.3.4 Automatic logarithmic fading (ALD) 2193
29.3.5 Automatic decrement/increment (ADI) of volume 2195
29.3.6 Automatic duration control (ADC) 2196
29.3.7 INTSGnTI interrupt 2197
29.4 Sound Generator Application Hints 2198
29.4.1 Initialization 2198
29.4.2 Start sound 2199
29.4.3 Stop sound 2199
29.4.4 Pause Sound 2199
29.4.5 Change sound volume 2200
29.4.6 Register buffering at interrupt generation 2200
29.4.7 Constant sound volume 2200
29.4.8 Generate special sounds 2200
29.5 Registers 2201
29.5.1 SG registers overview 2201
29.5.2 SG registers details 2202

Chapter 30 Voltage Comparator (VCPC) 2213
30.1 V850E2/Dx4(-H) VCPC Features 2213
30.2 Overview 2216
30.2.1 Description 2217
30.2.2 Comparison results 2218
30.2.3 Stand-by mode 2218
30.2.4 Timing 2219
30.3 Voltage Comparator Registers 2220

Chapter 31 2D Drawing Engine (DRW) 2223
31.1 V850E2/Dx4(-H) DRW Features 2223
31.2 Functional Overview 2225
31.3 Introduction 2226

31.4 Drawing Features . . . 2228
31.4.1 Drawing features summary . . . 2228
31.4.2 Vector drawing . . . 2229
31.4.3 BitBLT . . . 2230
31.5 Input and Output Data Formats . . . 2231
31.5.1 Source and destination data . . . 2231
31.5.2 Texture colour formats . . . 2231
31.5.3 Framebuffer colour formats . . . 2232
31.6 Texture data processing . . . 2233
31.6.1 Texture colour format . . . 2233
31.6.2 RLE unit . . . 2234
31.6.3 Colour Lookup Table (CLUT) . . . 2237
31.6.4 Colour keying . . . 2237
31.7 Rendering Pipeline . . . 2238
31.7.1 Coordinate transformation . . . 2238
31.7.2 Rasterization . . . 2238
31.7.3 Edge setup linear case . . . 2240
31.7.4 Edge setup quadratic case . . . 2244
31.7.5 Band filter . . . 2247
31.7.6 Clamping unit . . . 2248
31.7.7 Combiner unit . . . 2249
31.7.8 Rasterization optimization . . . 2250
31.7.9 Colourization . . . 2253
31.7.10 Texturing . . . 2254
31.7.11 Blending . . . 2258
31.8 Rendering Modes . . . 2260
31.8.1 Register based mode . . . 2260
31.8.2 Display list based mode . . . 2260
31.9 Interrupts . . . 2263
31.9.1 Interrupt sources . . . 2263
31.9.2 Interrupt control . . . 2263
31.10 Performance Counters . . . 2264
31.11 2D Drawing Engine Registers . . . 2265
31.11.1 2D Drawing Engine registers overview . . . 2265
31.11.2 Control registers details . . . 2267
31.11.3 Colour registers details . . . 2278
31.11.4 Limiter registers details . . . 2281
31.11.5 Texture registers details . . . 2285
31.11.6 Miscellaneous registers details . . . 2300

Chapter 32 Video Input Interface (VI) . . . 2307
32.1 V850E2/Dx4(-H) VI Features . . . 2307
32.1.1 Video Input Interface control . . . 2310
32.2 Functional Overview . . . 2311
32.3 Data Formats and Synchronization . . . 2313
32.3.1 Video input data formats . . . 2316
32.3.2 Synchronization signals . . . 2317
32.4 Video Capturing . . . 2320
32.4.1 Video capturing modes . . . 2320
32.4.2 Start and stop of video capturing . . . 2321

32.4.3 Register update synchronization . . . 2322
32.5 Scaling, Cropping and Storing . . . 2323
32.5.1 Cropping . . . 2324
32.5.2 Scaling . . . 2325
32.5.3 Storing . . . 2330
32.6 Colour Format Conversions . . . 2331
32.6.1 RGB(666)/(565) to RGB(888) conversion . . . 2331
32.6.2 YUV(4:2:2) to YUV(4:4:4) conversion . . . 2332
32.6.3 YUV(4:4:4) to RGB(888) conversion . . . 2332
32.6.4 RGB(888) to Y(8) conversion . . . 2333
32.7 YUV Adjustment . . . 2333
32.8 Dithering . . . 2334
32.9 Video Data FIFO and Framebuffer addressing . . . 2335
32.9.1 Framebuffer data formats . . . 2335
32.9.2 Video data FIFO . . . 2335
32.9.3 Framebuffer addressing . . . 2337
32.10 Interrupts . . . 2338
32.10.1 Interrupt sources . . . 2338
32.11 Video Input Registers . . . 2340
32.11.1 Video Input registers overview . . . 2340
32.11.2 Video Input registers details . . . 2341

Chapter 33 Multi-Layer Video Output (MVO) . . . 2363
33.1 V850E2/Dx4(-H) MVO Features . . . 2363
33.2 Multi-Layer Video Output Modes . . . 2366
33.2.1 Separate mode . . . 2366
33.2.2 Video Output - Video Input synchronized mode . . . 2368
33.2.3 GSVCTRL - Video interface control register . . . 2370
33.3 Multi-Layer Video Output Pixel Clock Generation . . . 2371
33.3.1 Multi-Layer Video Output Clock Divider . . . 2372
33.3.2 MVOn RSDS interface control register details . . . 2374
33.4 Functional Overview . . . 2378
33.4.1 Features summary . . . 2378
33.4.2 Functions . . . 2379
33.4.3 Configuration . . . 2380
33.4.4 Overlay . . . 2381
33.4.5 Memory frame/display frame . . . 2382
33.5 Display control . . . 2383
33.5.1 Display control . . . 2383
33.5.2 Scaling . . . 2385
33.5.3 Outside area processing of display frame . . . 2386
33.6 Pixel format . . . 2389
33.6.1 Pixel format list . . . 2389
33.6.2 Transmittance format selection function . . . 2393
33.7 Transparency function . . . 2394
33.7.1 Layer transparency . . . 2395
33.7.2 Colour key function . . . 2399
33.7.3 Colour key permissible range setting function . . . 2400
33.8 Other display functions . . . 2401
33.8.1 Constant colour . . . 2401

33.8.2 Background colour 2403
33.8.3 Internal bit expansion 2404
33.8.4 Endian conversion 2409
33.9 Display output 2410
33.9.1 Signal functions 2410
33.9.2 Display dot clock 2410
33.9.3 Sync signal 2411
33.9.4 External synchronization 2412
33.9.5 Display on/off function 2412
33.9.6 Dithering function 2412
33.9.7 Brightness function 2414
33.9.8 Gamma correction function 2414
33.9.9 Colour detection signal (CDE) 2414
33.9.10 Colour detection signal permissible range setting function 2415
33.9.11 Output timing adjustment function 2415
33.10 Register update timing 2418
33.11 Interrupt control 2418
33.12 Registers 2419
33.12.1 Interface control registers 2421
33.12.2 Display control registers 2430
33.12.3 Memory frame control registers 2437
33.12.4 Display frame control registers 2438
33.12.5 Layer unit registers 2442
33.12.6 Data of tables 2460

Chapter 34 Single-Layer Video Output (SVO) 2463
34.1 V850E2/Dx4(-H) SVO Features 2463
34.2 Single-Layer Video Output Configuration 2466
34.3 Single-Layer Video Output Pixel Clock Generation 2468
34.3.1 Single-Layer Video Output Clock Divider 2469
34.4 Functional Overview 2471
34.5 Colour Formats 2472
34.5.1 CLUT palette 2473
34.5.2 Display data output formats 2474
34.6 Timing Signals 2475
34.6.1 Display data clock signal 2475
34.6.2 Synchronization signals 2475
34.7 DMA FIFO and Framebuffer Addressing 2477
34.7.1 FIFO SVOn watermark 2478
34.7.2 Framebuffer addressing 2478
34.8 Interrupts 2479
34.8.1 Interrupt sources 2479
34.8.2 Interrupt controller 2480
34.9 Start and Stop of Video Output 2481
34.10 Single-Layer Video Output I/F Registers 2482
34.10.1 SVOn registers overview 2482
34.10.2 Video Output registers details 2483

Chapter 35 Video Output Timing Controller (TCON) 2495

35.1 V850E2/Dx4(-H) TCON Features 2495
35.2 Functional Overview 2499
35.3 Functional Description 2500
35.3.1 Generation of synchronization pulse signals 2501
35.3.2 Combination of synchronization pulse signals 2506
35.3.3 Inversion of pixel clock input 2508
35.3.4 Inversion of Video Output Interface timing signals 2508
35.3.5 Configuration of TCONn 2508
35.3.6 Bypassing TCONn functions 2508
35.4 Video Output Timing Controller Registers 2509
35.4.1 Video Output Timing Controller registers overview 2509
35.4.2 Control registers details 2510
35.4.3 Sync-pulse generator registers details 2511
35.4.4 Sync-Pulse Combining Unit registers details 2514

Chapter 36 Video Output Monitor (VOMN) 2515
36.1 V850E2/Dx4(-H) VOMN Features 2515
36.2 Functional Overview 2518
36.3 Functional Description 2518
36.3.1 Definiton of the monitored area 2519
36.3.2 Generation of checksums 2520
36.3.3 Indication of transmission error 2521
36.3.4 Status indicators 2522
36.3.5 Active level of synchronization signals 2522
36.3.6 Using the Video Output Monitor (basic procedure) 2523
36.4 Video Output Monitor Registers 2524
36.4.1 Video Output Monitor registers overview 2524
36.4.2 Control registers details 2525
36.4.3 Status registers details 2533

Chapter 37 SDRAM Memory Controller (EMC) 2535
37.1 V850E2/Dx4(-H) EMC Features 2535
37.2 µPD70F3532 SDRAM Controller I/O Signals 2537
37.3 µPD70F3535 to µPD70F3537 SDRAM Controller and embedded SDRAM connection 2538
37.3.1 SDRAM Memory Controller initialization 2539
37.3.2 Port control settings 2540
37.4 Functional Overview 2542
37.5 Memory Map 2543
37.6 Memory connected Area / non-connected Area 2544
37.7 ECC Protection 2545
37.7.1 ECC mode enable 2545
37.7.2 Memory map with ECC 2546
37.7.3 Memory size and ECC 2547
37.7.4 ECC data storage 2548
37.7.5 ECC status information 2548
37.8 Write Access protected Area 2549
37.9 Write Access monitored Area 2550
37.10 SDRAM Clock 2551
37.11 Interrupts 2551

37.12 SDRAM Interface Commands 2552
37.13 SDRAM Memory Controller Registers 2561
37.13.1 SDRAM Memory Controller registers overview 2561
37.13.2 SDRAM Memory Controller registers details 2562

Chapter 38 On-chip Debug Unit (OCD) 2575
38.1 V850E2/Dx4(-H)On-chip Debug Features 2575
38.1.1 Modules behaviour during emulation break 2575
38.1.2 Signal masking 2577
38.2 Functional Overview 2578
38.3 Emulation Break Control 2580
38.4 Connection with On-Chip Debug Emulator 2581
38.5 Cautions on using On-Chip Debugging 2582

Chapter 39 Power Supply 2583
39.1 Power supply scheme 2583
39.1.1 µPD70F3522 to µPD70F3526 power supply scheme 2584
39.1.2 µPD70F3532 power supply scheme 2585
39.1.3 µPD70F3535 to µPD70F3537 power supply scheme 2587

Chapter 40 Boundary Scan 2589
40.1 Outline 2589
40.2 JTAG interface 2589
40.3 Entering Boundary Scan mode 2589
40.4 Boundary scan feature 2590
40.5 Boundary Scan applicable pins 2590
40.6 Device ID register (DID) 2592

Index 2593

Chapter 1 Introduction

1.1 V850E2/Dx4(-H) products features

The following tables show the common and different features of various devices.

(1) Product line overview V850E2/Dx4(-H) µPD70F352x

Table 1-1 V850E2/Dx4(-H) Product Series Overview (µPD70F352x) (1/3)

Part number		µPD70F3522	µPD70F3523	µPD70F3524	µPD70F3525	µPD70F3526
Nickname		DJ4-256K	DJ4-512K	DJ4-1M	DJ4-2M	DJ4-3M
Internal memory	Instruction flash	256 KB	512 KB	1 MB	2 MB	3 MB
	Data flash	32 KB				
	RAM	24 KB	48 KB	96 KB	192 KB	256 KB
	Back-up RAM (BURAM)	16 KB				
External memory interface	High Speed Flash I/F (CSI mode only) (HSFI)	–		1 channel		
CPU	CPU System	V850E2M-M				
	CPU frequency	40 MHz max. (+ 5% with SSCG)		80 MHz max. (+ 5% with SSCG)		
	System Protection Functions (SPF)	Memory Protection Unit (MPU) System Register Protection (SRP) Peripheral Protection Unit (PPU) Timing Supervision Unit (TPU)				
	Instruction cache	8 KB/ 2 way associative (4 KB/ way)				
DMA/DTS	DMA	8 channels				
	DTS	provided				
Operating clock	Main Oscillator (MainOsc)	4 MHz to 20 MHz				
	Low Speed Internal Oscillator (LS IntOsc)	240 KHz typ.				
	High Speed Internal Oscillator (HS IntOsc)	8 MHz typ.				
	Sub oscillator (SubOsc)	32768 Hz typ.				
	PLL0 (SSCG0)	40 MHz max. (+ 5% with SSCG)		80 MHz max. (+ 5% with SSCG)		
	PLL1	240 MHz max.				
	PLL2 (SSCG2)	240 MHz max. (+ 5% with SSCG)				
I/O ports	Input/Output	105				
A/D converter (ADCA)		16 channels, 12 bit				

Table 1-1 V850E2/Dx4(-H) Product Series Overview (µPD70F352x) (2/3)

Part number			µPD70F3522	µPD70F3523	µPD70F3524	µPD70F3525	µPD70F3526
Nickname			DJ4-256K	DJ4-512K	DJ4-1M	DJ4-2M	DJ4-3M
Timers	Timer Array Unit A (TAUA), 16 bit		5 units x 16 channels				
	Timer Array Unit J (TAUJ), 32 bit		3 units x 4 channels				
	Realtime Clock (RTCA) and calibration		1 unit				
	Window Watchdog (WDTA)		2 channels				
	Operating System Timer (OSTM)		1 channel				
	PCM-PWM Timer (PCMP)		provided				
Serial interfaces	CAN I/F (FCN)		3 channels (64 messages buffer)				
	UART I/F (URTE) with LIN support		2 channels				
	Synchronous I/F (CSIG)		3 channels				
	I^2C I/F (IICB)		2 channels				
	I^2S I/F (IISA)		–		1 channel		
Interrupts	Maskable	External	11				
		Internal	161		171		
	Non-maskable (NMI)	External	1				
		Internal	2				
LCD I/F	LCD Controller/ Driver (LCCT)		69 segments x 6 backplanes				
	LCD Bus I/F (LCBI)		8/18 bit				
Intelligent Stepper Motor Controller/ Driver (ISM)			6 channels incl. Zero-Point-Detection for each channel				
Other functions	Power-On-Clear (POC)		provided				
	Voltage Comparators (VCPC)		2 channels				
	Clock Monitors (CLMA)		provided for MainOsc, SubOsc, HS IntOsc, PLL0 supervision				
	Sound Generator (SG)		1 channel				
	Wake-up signal output		provided				
	Auxiliary frequency output (FOUT)		provided				
	On-Chip debug (OCD)		provided				

Table 1-1 V850E2/Dx4(-H) Product Series Overview (µPD70F352x) (3/3)

Part number		µPD70F3522	µPD70F3523	µPD70F3524	µPD70F3525	µPD70F3526
Nickname		DJ4-256K	DJ4-512K	DJ4-1M	DJ4-2M	DJ4-3M
Voltage supply	Internal supply	3.0 V to 5.5 V[a]				
	I/O supply	3.0 V to 5.5 V[a]				
	A/D Converter supply	3.0 V to 5.5 V[a]				
Operating Temperature		-40° C … +85° C (-40° C … +105° C with limited specification)[a]				
Package		144-pin LQFP/HLQFP				

a) Refer to the Data Sheet.

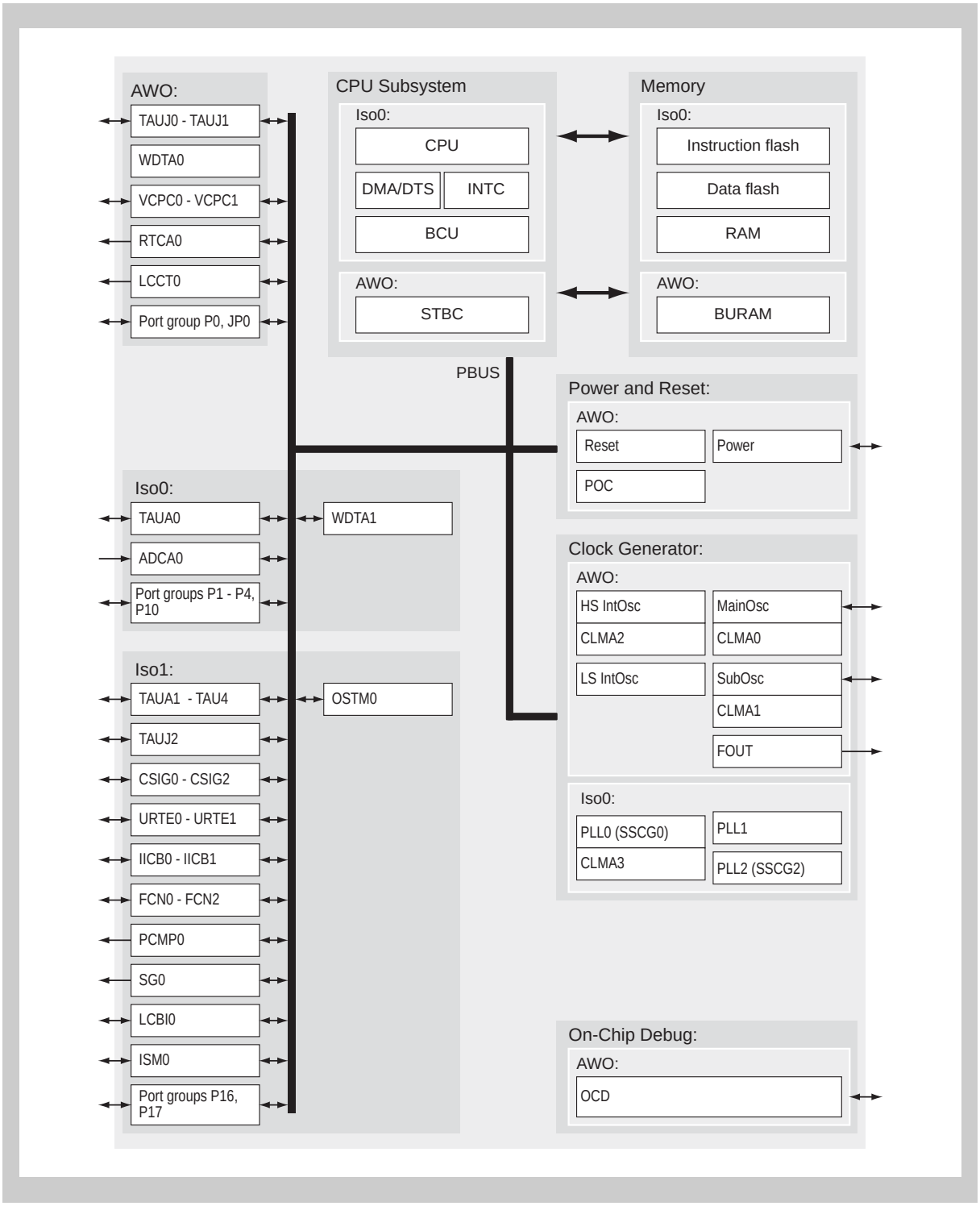

Figure 1-1 V850E2/Dx4(-H) µPD70F3522, µPD70F3523 block diagram

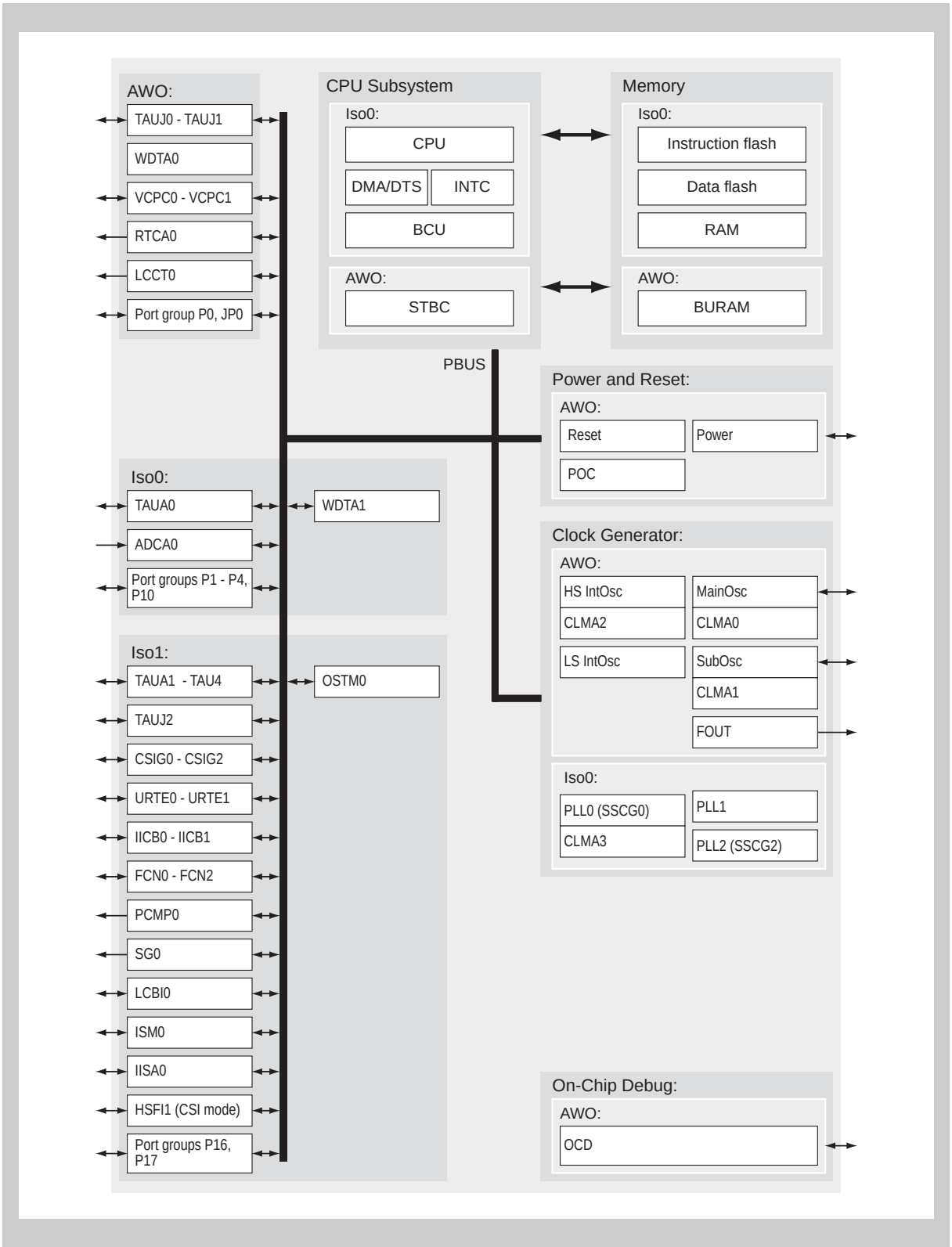

Figure 1-2 V850E2/Dx4(-H) µPD70F3524 to µPD70F3526 block diagram

(2) Product line overview V850E2/Dx4(-H) µPD70F353x

Table 1-2 V850E2/Dx4(-H) Product Series Overview (µPD70F353x) (1/2)

Part number		µPD70F3532	µPD70F3535	µPD70F3536	µPD70F3537
Nickname		DN4-H-3M	DP4-H3	DP4-H5	DP4-H8
Internal memory	Instruction flash	3 MB			
	Data flash	32 KB			
	RAM	256 KB			
	Back-up RAM (BURAM)	16 KB			
	Video RAM (eDRAM)	–	3 MB	5 MB	8 MB
CPU	CPU System	V850E2M-H			
	Floating Point Unit (FPU)	provided			
	CPU frequency	120 MHz max. (+ 5% with SSCG)			
	System Protection Functions (SPF)	Memory Protection Unit (MPU) System Register Protection (SRP) Peripheral Protection Unit (PPU) Timing Supervision Unit (TPU)			
	Instruction cache	16 KB / 4/2 way associative (4 KB/way)			
	AHB cache	16 KB/ 4 way associative (4 KB/ way)			
External SDRAM Memory I/F (EMC)		32 bit/128 MB	–		
DMA/DTS	DMA	8 channels			
	DTS	provided			
Operating clock	Main Oscillator (MainOsc)	4 MHz to 20 MHz			
	Low Speed Internal Oscillator (LS IntOsc)	240 KHz typ.			
	High Speed Internal Oscillator (HS IntOsc)	8 MHz typ.			
	Sub oscillator (SubOsc)	32768 Hz typ.			
	PLL0 (SSCG0)	120 MHz max. (+ 5% with SSCG)			
	PLL1	240 MHz max.			
	PLL2 (SSCG2)	240 MHz max. (+ 5% with SSCG)			
I/O ports	Input/Output	139	120		139
	Output	26	26		
A/D converter (ADCA)		16 channels, 12 bit			
Timers	Timer Array Unit A (TAUA), 16 bit	5 units x 16 channels			
	Timer Array Unit J (TAUJ), 32 bit	3 units x 4 channels			
	Realtime Clock (RTCA) and calibration	1 unit			
	Window Watchdog (WDTA)	2 channels			
	Operating System Timer (OSTM)	1 channel			
	PCM-PWM Timer (PCMP)	provided			

Table 1-2 V850E2/Dx4(-H) Product Series Overview (µPD70F353x) (2/2)

Part number			µPD70F3532	µPD70F3535	µPD70F3536	µPD70F3537
Nickname			DN4-H-3M	DP4-H3	DP4-H5	DP4-H8
Serial interfaces	CAN I/F (FCN)		3 channels (64 messages buffer)			
	UART I/F (URTE) with LIN support		4 channels	2 channels		
	Synchronous I/F (CSIG)		3 channels			
	I^2C I/F (IICB)		2 channels			
	I^2S I/F (IISA)		1 channel			
	FlexRay Controller (FLX)		1 channel (8 KB message RAM)			
Interrupts	Maskable	External	11			
		Internal	195			
	Non-maskable (NMI)	External	1			
		Internal	3			
LCD Bus I/F (LCBI)			8/18 bit			
Intelligent Stepper Motor Controller/Driver (ISM)			6 channels incl. Zero-Point-Detection for each channel			
Graphics functions	High Speed Flash I/F (HSFI)		2 channels			
	2D Drawing Engine (DRW)		provided			
	Single-layer Video Output (SVO)		1 channel (1024 x 1024, 1 layer)			
	Multi-layer Video Output (MVO)		1 channel (1280 x 1024, 4 layers)			
	Video Output Monitor (VOM)		2 channels			
	Video Input I/F (VI)		RGB(666), ITU-656	–		RGB(666), ITU-65
	Video Output Timing Controller (TCON)		2 channels			
	RSDS Encoder Output		provided			
Other functions	Power-On-Clear (POC)		provided			
	Voltage Comparators (VCPC)		2 channels			
	Clock Monitors (CLMA)		provided for MainOsc, SubOsc, HS IntOsc, PLL0 supervision			
	Sound Generator (SG)		1 channel			
	Wake-up signal output		provided			
	Auxiliary frequency output (FOUT)		provided			
	PowerGood (PWGD) signal input		provided			
	On-Chip debug (OCD)		provided			
Operating voltage	Internal supply		1.2 V ± 0.1 V[a]			
	I/O supply		3.0 V to 5.5 V[a]			
	A/D Converter supply		3.0 V to 5.5 V[a]			
Operating Temperature			-40° C … +85° C (-40° C … +105° C with limited specification)[a]			
Package			352-pin BGA	408-pin BGA		

a) Refer to the Data Sheet.

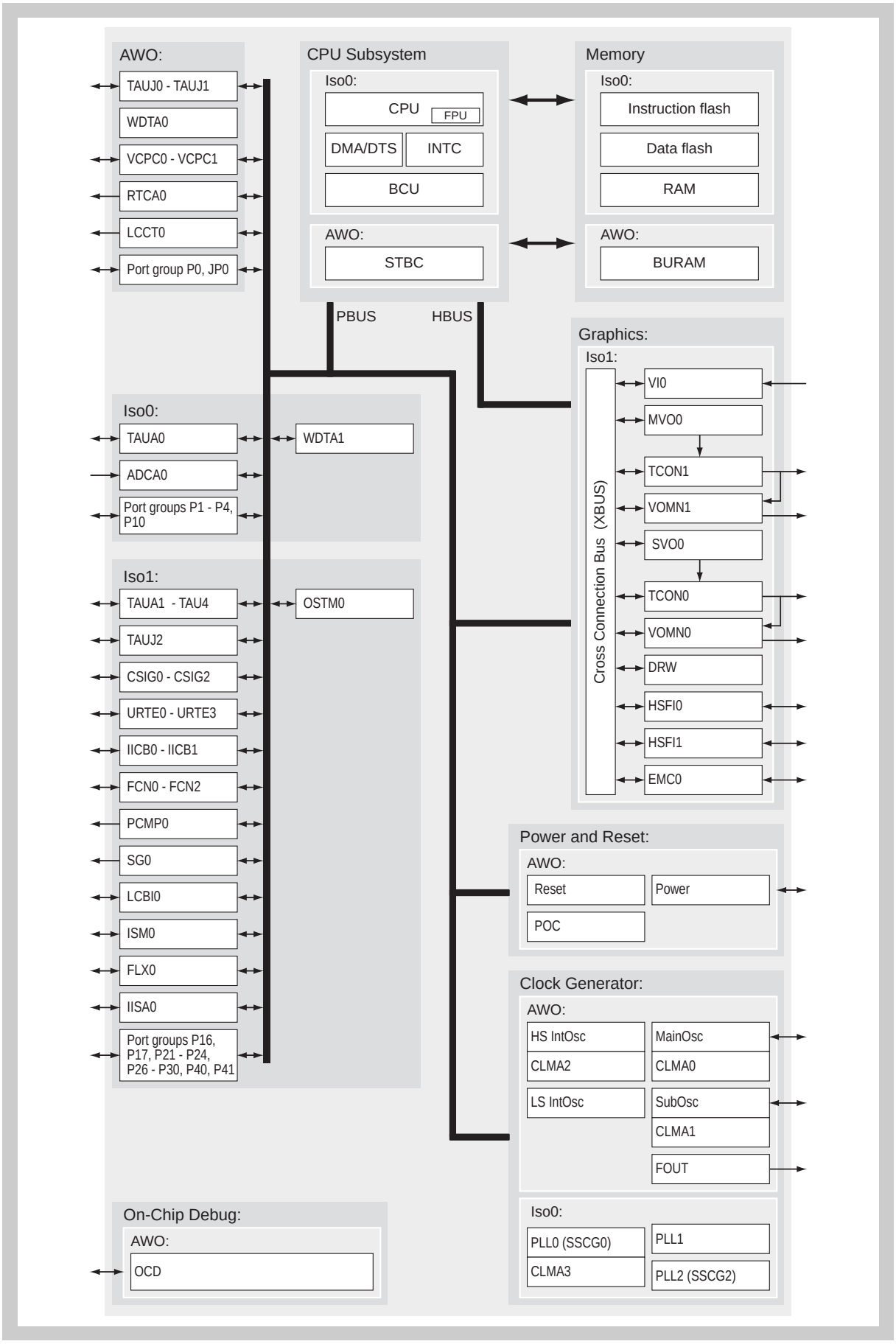

Figure 1-3 V850E2/Dx4(-H) µPD70F3532 block diagram

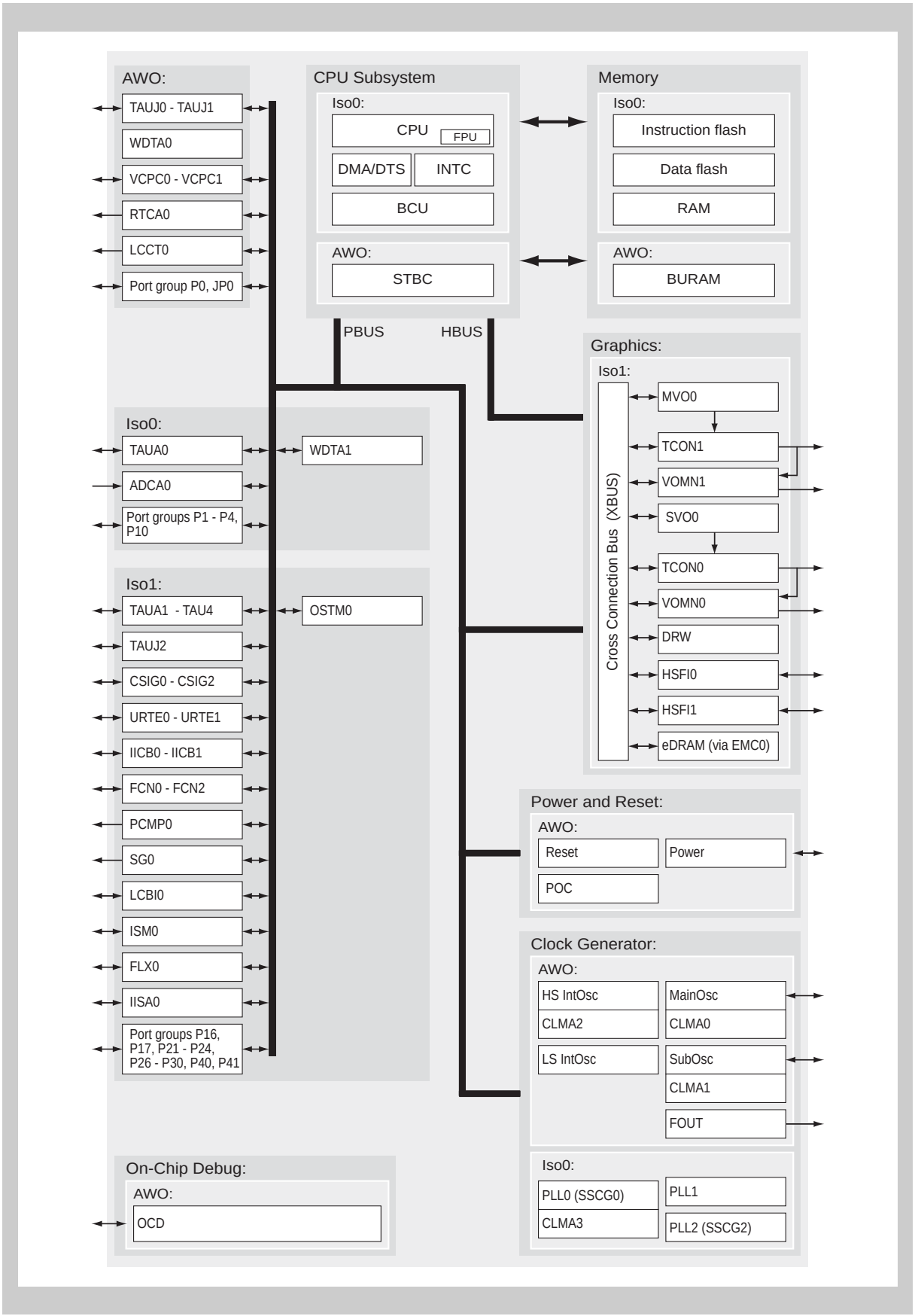

Figure 1-4 V850E2/Dx4(-H) µPD70F3535, µPD70F3536 block diagram

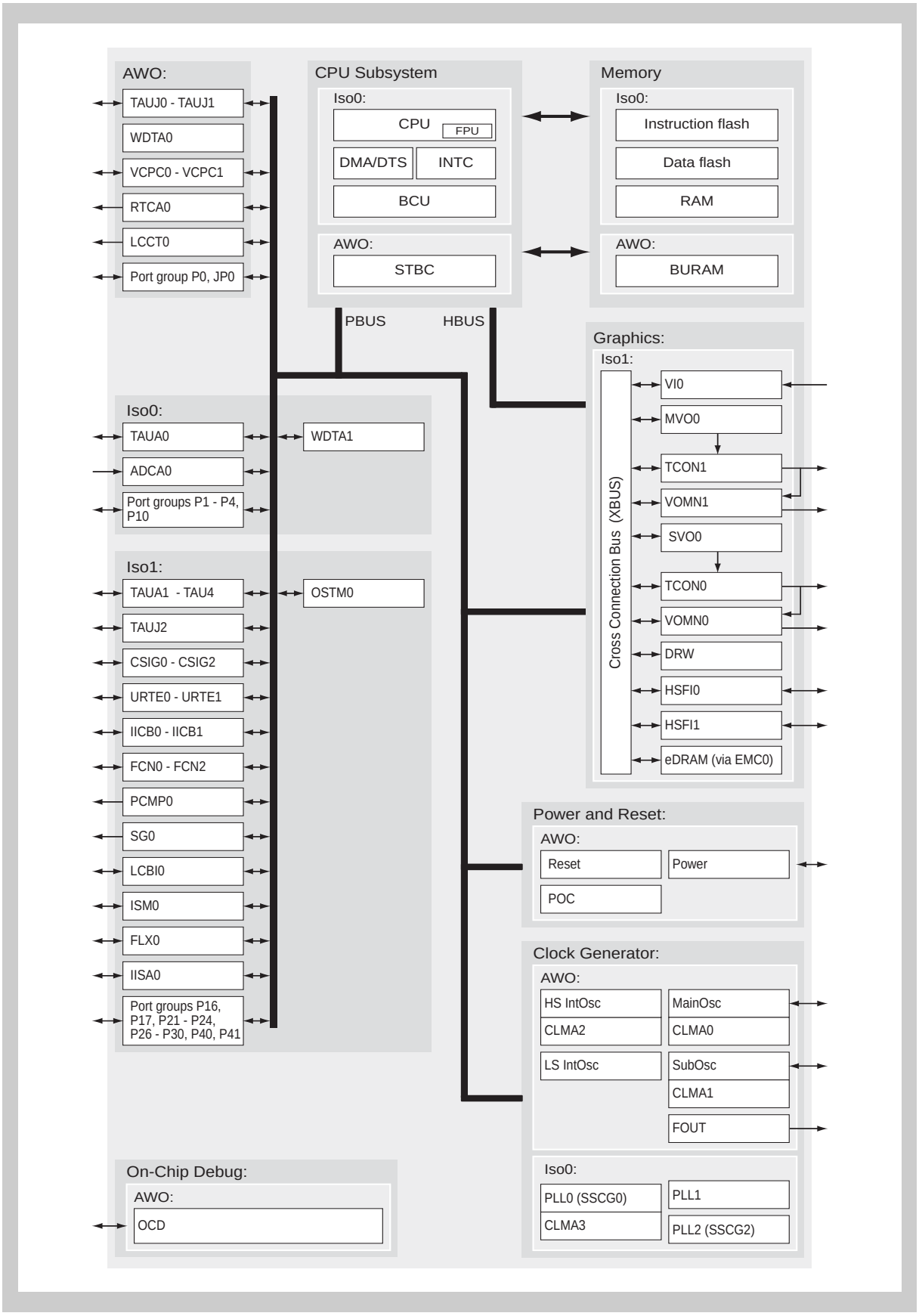

Figure 1-5 V850E2/Dx4(-H) µPD70F3537 block diagram

1.2 Related Documents

Table 1-3 Related documents

Document number	Title
U19949EJxVxUM00	V850E2M 32-bit Microcontroller Core Architecture
EASE-ES-0016-x.x	Data Sheet

1.3 Ordering Information

Table 1-4 V850E2/Dx4(-H) ordering information

Device name	NEC order code[a]	Remarks
μPD70F3522	UPD70F3522xxx	
μPD70F3523	μPD70F3523xxx	
μPD70F3524	μPD70F3524xxx	
μPD70F3525	μPD70F3525xxx	
μPD70F3526	μPD70F3526xxx	
μPD70F3532	μPD70F3532xxx	
μPD70F3535	μPD70F3535xxx	
μPD70F3536	μPD70F3536xxx	
μPD70F3537	μPD70F3537xxx	

a) The final complete order codes are not fixed yet. Thus the placeholders “xxx” will be provided later.