```
/*
 * Intel ACPI Component Architecture
 * AML/ASL+ Disassembler version 20160422-64(RM)
 * Copyright (c) 2000 - 2016 Intel Corporation
 *
 * Disassembling to non-symbolic legacy ASL operators
 *
 * Disassembly of iASL6sdmyH.aml, Fri Oct 28 04:29:29 2016
 *
 * Original Table Header:
 *     Signature        "DSDT"
 *     Length           0x00007AEC (31468)
 *     Revision         0x01 **** 32-bit table (V1), no 64-bit math support
 *     Checksum         0x6A
 *     OEM ID           "A1282"
 *     OEM Table ID     "A1282000"
 *     OEM Revision     0x00000000 (0)
 *     Compiler ID      "INTL"
 *     Compiler Version 0x20060113 (537264403)
 */
DefinitionBlock ("", "DSDT", 1, "A1282", "A1282000", 0x00000000)
{
    Scope (_PR)
    {
        Processor (P001, 0x01, 0x00000810, 0x06) {}
        Alias (P001, CPU1)
        Processor (P002, 0x02, 0x00000000, 0x00) {}
        Alias (P002, CPU2)
        Processor (P003, 0x03, 0x00000000, 0x00) {}
        Alias (P003, CPU3)
        Processor (P004, 0x04, 0x00000000, 0x00) {}
        Alias (P004, CPU4)
    }

    Name (DP80, 0x80)
    Name (DP90, 0x90)
    Name (SPIO, 0x2E)
    Name (IOHW, 0x0290)
    Name (APIC, One)
    Name (PMBS, 0x0800)
    Name (PMLN, 0x80)
    Name (GPBS, 0x0480)
    Name (GPLN, 0x40)
    Name (SMBL, Zero)
    Name (PM30, 0x0830)
    Name (SUSW, 0xFF)
    Name (SMIO, 0xB2)
    Name (EAQF, One)
    Name (CQST, 0x3C)
```

```
Name (TOBS, 0x0860)
Name (SUCC, One)
Name (NVLD, 0x02)
Name (CRIT, 0x04)
Name (NCRT, 0x06)
Name (LIDS, One)
Name (PCIB, 0xF0000000)
Name (PCIL, 0x04000000)
Name (CPUC, 0x04)
Name (SMBS, 0x0400)
OperationRegion (BIOS, SystemMemory, 0xCFF8E064, 0xFF)
Field (BIOS, ByteAcc, NoLock, Preserve)
{
    SS1,    1,
    SS2,    1,
    SS3,    1,
    SS4,    1,
    Offset (0x01),
    IOST,   16,
    TOPM,   32,
    ROMS,   32,
    MG1B,   32,
    MG1L,   32,
    MG2B,   32,
    MG2L,   32,
    Offset (0x1C),
    CPB0,   32,
    CPB1,   32,
    CPB2,   32,
    CPB3,   32,
    ASSB,   8,
    AOTB,   8,
    AAXB,   32,
    SMIF,   8,
    DTSE,   8,
    DTS1,   8,
    DTS2,   8,
    MPEN,   8,
    TPMF,   8,
    MG3B,   32,
    MG3L,   32,
    MSC1,   32,
    MSC2,   32,
    MSC3,   32,
    MSC4,   32,
    MSC5,   32,
    MSC6,   32,
    MSC7,   32,
    MSC8,   32,
```

```
    DMAX,   8,
    HPTA,   32
}

Method (RRIO, 4, NotSerialized)
{
    Store ("RRIO", Debug)
}

Method (RDMA, 3, NotSerialized)
{
    Store ("rDMA", Debug)
}

Name (PICM, Zero)
Method (_PIC, 1, NotSerialized)  // _PIC: Interrupt Model
{
    If (Arg0)
    {
        Store (0xAA, DBG8)
    }
    Else
    {
        Store (0xAC, DBG8)
    }

    Store (Arg0, PICM)
}

Name (OSVR, Ones)
Method (OSFL, 0, NotSerialized)
{
    If (LNotEqual (OSVR, Ones))
    {
        Return (OSVR)
    }

    If (LEqual (PICM, Zero))
    {
        Store (0xAC, DBG8)
    }

    Store (One, OSVR)
    If (CondRefOf (_OSI, Local1))
    {
        If (_OSI ("Windows 2000"))
        {
            Store (0x04, OSVR)
        }
```

```
    If (_OSI ("Windows 2001"))
    {
        Store (Zero, OSVR)
    }

    If (_OSI ("Windows 2001 SP1"))
    {
        Store (Zero, OSVR)
    }

    If (_OSI ("Windows 2001 SP2"))
    {
        Store (Zero, OSVR)
    }

    If (_OSI ("Windows 2001.1"))
    {
        Store (Zero, OSVR)
    }

    If (_OSI ("Windows 2001.1 SP1"))
    {
        Store (Zero, OSVR)
    }

    If (_OSI ("Windows 2006"))
    {
        Store (Zero, OSVR)
    }
}
ElseIf (MCTH (_OS, "Microsoft Windows NT"))
{
    Store (0x04, OSVR)
}
Else
{
    If (MCTH (_OS, "Microsoft WindowsME: Millennium Edition"))
    {
        Store (0x02, OSVR)
    }

    If (MCTH (_OS, "Linux"))
    {
        Store (0x03, OSVR)
    }
}

Return (OSVR)
```

```
    }

    Method (MCTH, 2, NotSerialized)
    {
        If (LLess (SizeOf (Arg0), SizeOf (Arg1)))
        {
            Return (Zero)
        }

        Add (SizeOf (Arg0), One, Local0)
        Name (BUF0, Buffer (Local0) {})
        Name (BUF1, Buffer (Local0) {})
        Store (Arg0, BUF0)
        Store (Arg1, BUF1)
        While (Local0)
        {
            Decrement (Local0)
            If (LNotEqual (DerefOf (Index (BUF0, Local0)), DerefOf (Index (BUF1,
Local0))))
            {
                Return (Zero)
            }
        }

        Return (One)
    }

    Name (PRWP, Package (0x02)
    {
        Zero,
        Zero
    })
    Method (GPRW, 2, NotSerialized)
    {
        Store (Arg0, Index (PRWP, Zero))
        Store (ShiftLeft (SS1, One), Local0)
        Or (Local0, ShiftLeft (SS2, 0x02), Local0)
        Or (Local0, ShiftLeft (SS3, 0x03), Local0)
        Or (Local0, ShiftLeft (SS4, 0x04), Local0)
        If (And (ShiftLeft (One, Arg1), Local0))
        {
            Store (Arg1, Index (PRWP, One))
        }
        Else
        {
            ShiftRight (Local0, One, Local0)
            If (LOr (LEqual (OSFL (), One), LEqual (OSFL (), 0x02)))
            {
                FindSetLeftBit (Local0, Index (PRWP, One))
```

```
        }
        Else
        {
            FindSetRightBit (Local0, Index (PRWP, One))
        }
    }

    Return (PRWP)
}

Name (WAKP, Package (0x02)
{
    Zero,
    Zero
})
OperationRegion (DEB0, SystemIO, DP80, One)
Field (DEB0, ByteAcc, NoLock, Preserve)
{
    DBG8,   8
}

OperationRegion (DEB1, SystemIO, DP90, 0x02)
Field (DEB1, WordAcc, NoLock, Preserve)
{
    DBG9,   16
}

Scope (_SB)
{
    Name (PR00, Package (0x16)
    {
        Package (0x04)
        {
            0x0001FFFF,
            Zero,
            LNKA,
            Zero
        },

        Package (0x04)
        {
            0x0001FFFF,
            One,
            LNKB,
            Zero
        },

        Package (0x04)
        {
```

```
    0x0001FFFF,
    0x02,
    LNKC,
    Zero
},

Package (0x04)
{
    0x0001FFFF,
    0x03,
    LNKD,
    Zero
},

Package (0x04)
{
    0x0006FFFF,
    Zero,
    LNKA,
    Zero
},

Package (0x04)
{
    0x0006FFFF,
    One,
    LNKB,
    Zero
},

Package (0x04)
{
    0x0006FFFF,
    0x02,
    LNKC,
    Zero
},

Package (0x04)
{
    0x0006FFFF,
    0x03,
    LNKD,
    Zero
},

Package (0x04)
{
    0x001FFFFF,
```

```
    Zero,
    LNKC,
    Zero
},

Package (0x04)
{
    0x001FFFFF,
    One,
    LNKG,
    Zero
},

Package (0x04)
{
    0x001DFFFF,
    Zero,
    LNKH,
    Zero
},

Package (0x04)
{
    0x001DFFFF,
    One,
    LNKD,
    Zero
},

Package (0x04)
{
    0x001DFFFF,
    0x02,
    LNKC,
    Zero
},

Package (0x04)
{
    0x001DFFFF,
    0x03,
    LNKA,
    Zero
},

Package (0x04)
{
    0x001EFFFF,
    Zero,
```

```
    LNKB,
    Zero
},

Package (0x04)
{
    0x001EFFFF,
    One,
    LNKE,
    Zero
},

Package (0x04)
{
    0x001CFFFF,
    Zero,
    LNKA,
    Zero
},

Package (0x04)
{
    0x001CFFFF,
    One,
    LNKB,
    Zero
},

Package (0x04)
{
    0x001CFFFF,
    0x02,
    LNKC,
    Zero
},

Package (0x04)
{
    0x001CFFFF,
    0x03,
    LNKD,
    Zero
},

Package (0x04)
{
    0x0002FFFF,
    Zero,
    LNKA,
```

```
            Zero
        },

        Package (0x04)
        {
            0x001BFFFF,
            Zero,
            LNKF,
            Zero
        }
    })
    Name (AR00, Package (0x16)
    {
        Package (0x04)
        {
            0x0001FFFF,
            Zero,
            Zero,
            0x10
        },

        Package (0x04)
        {
            0x0001FFFF,
            One,
            Zero,
            0x11
        },

        Package (0x04)
        {
            0x0001FFFF,
            0x02,
            Zero,
            0x12
        },

        Package (0x04)
        {
            0x0001FFFF,
            0x03,
            Zero,
            0x13
        },

        Package (0x04)
        {
            0x0006FFFF,
            Zero,
```

```
    Zero,
    0x10
},

Package (0x04)
{
    0x0006FFFF,
    One,
    Zero,
    0x11
},

Package (0x04)
{
    0x0006FFFF,
    0x02,
    Zero,
    0x12
},

Package (0x04)
{
    0x0006FFFF,
    0x03,
    Zero,
    0x13
},

Package (0x04)
{
    0x001FFFFF,
    Zero,
    Zero,
    0x12
},

Package (0x04)
{
    0x001FFFFF,
    One,
    Zero,
    0x16
},

Package (0x04)
{
    0x001DFFFF,
    Zero,
    Zero,
```

```
    0x17
},

Package (0x04)
{
    0x001DFFFF,
    One,
    Zero,
    0x13
},

Package (0x04)
{
    0x001DFFFF,
    0x02,
    Zero,
    0x12
},

Package (0x04)
{
    0x001DFFFF,
    0x03,
    Zero,
    0x10
},

Package (0x04)
{
    0x001EFFFF,
    Zero,
    Zero,
    0x11
},

Package (0x04)
{
    0x001EFFFF,
    One,
    Zero,
    0x14
},

Package (0x04)
{
    0x001CFFFF,
    Zero,
    Zero,
    0x10
```

```
    },

    Package (0x04)
    {
        0x001CFFFF,
        One,
        Zero,
        0x11
    },

    Package (0x04)
    {
        0x001CFFFF,
        0x02,
        Zero,
        0x12
    },

    Package (0x04)
    {
        0x001CFFFF,
        0x03,
        Zero,
        0x13
    },

    Package (0x04)
    {
        0x0002FFFF,
        Zero,
        Zero,
        0x10
    },

    Package (0x04)
    {
        0x001BFFFF,
        Zero,
        Zero,
        0x15
    }
})
Name (PR02, Package (0x04)
{
    Package (0x04)
    {
        0xFFFF,
        Zero,
        LNKA,
```

```
            Zero
        },

        Package (0x04)
        {
            0xFFFF,
            One,
            LNKB,
            Zero
        },

        Package (0x04)
        {
            0xFFFF,
            0x02,
            LNKC,
            Zero
        },

        Package (0x04)
        {
            0xFFFF,
            0x03,
            LNKD,
            Zero
        }
    })
    Name (AR02, Package (0x04)
    {
        Package (0x04)
        {
            0xFFFF,
            Zero,
            Zero,
            0x10
        },

        Package (0x04)
        {
            0xFFFF,
            One,
            Zero,
            0x11
        },

        Package (0x04)
        {
            0xFFFF,
            0x02,
```

```
            Zero,
            0x12
        },

        Package (0x04)
        {
            0xFFFF,
            0x03,
            Zero,
            0x13
        }
    })
    Name (PR03, Package (0x04)
    {
        Package (0x04)
        {
            0xFFFF,
            Zero,
            LNKA,
            Zero
        },

        Package (0x04)
        {
            0xFFFF,
            One,
            LNKB,
            Zero
        },

        Package (0x04)
        {
            0xFFFF,
            0x02,
            LNKC,
            Zero
        },

        Package (0x04)
        {
            0xFFFF,
            0x03,
            LNKD,
            Zero
        }
    })
    Name (AR03, Package (0x04)
    {
        Package (0x04)
```

```
    {
        0xFFFF,
        Zero,
        Zero,
        0x10
    },

    Package (0x04)
    {
        0xFFFF,
        One,
        Zero,
        0x11
    },

    Package (0x04)
    {
        0xFFFF,
        0x02,
        Zero,
        0x12
    },

    Package (0x04)
    {
        0xFFFF,
        0x03,
        Zero,
        0x13
    }
})
Name (PR01, Package (0x08)
{
    Package (0x04)
    {
        0xFFFF,
        Zero,
        LNKD,
        Zero
    },

    Package (0x04)
    {
        0xFFFF,
        One,
        LNKA,
        Zero
    },
```

```
    Package (0x04)
    {
        0xFFFF,
        0x02,
        LNKB,
        Zero
    },

    Package (0x04)
    {
        0xFFFF,
        0x03,
        LNKC,
        Zero
    },

    Package (0x04)
    {
        0x0001FFFF,
        Zero,
        LNKA,
        Zero
    },

    Package (0x04)
    {
        0x0001FFFF,
        One,
        LNKB,
        Zero
    },

    Package (0x04)
    {
        0x0001FFFF,
        0x02,
        LNKC,
        Zero
    },

    Package (0x04)
    {
        0x0001FFFF,
        0x03,
        LNKD,
        Zero
    }
})
Name (AR01, Package (0x08)
```

```
{
    Package (0x04)
    {
        0xFFFF,
        Zero,
        Zero,
        0x13
    },

    Package (0x04)
    {
        0xFFFF,
        One,
        Zero,
        0x10
    },

    Package (0x04)
    {
        0xFFFF,
        0x02,
        Zero,
        0x11
    },

    Package (0x04)
    {
        0xFFFF,
        0x03,
        Zero,
        0x12
    },

    Package (0x04)
    {
        0x0001FFFF,
        Zero,
        Zero,
        0x10
    },

    Package (0x04)
    {
        0x0001FFFF,
        One,
        Zero,
        0x11
    },
```

```
    Package (0x04)
    {
        0x0001FFFF,
        0x02,
        Zero,
        0x12
    },

    Package (0x04)
    {
        0x0001FFFF,
        0x03,
        Zero,
        0x13
    }
})
Name (PR04, Package (0x04)
{
    Package (0x04)
    {
        0xFFFF,
        Zero,
        LNKA,
        Zero
    },

    Package (0x04)
    {
        0xFFFF,
        One,
        LNKB,
        Zero
    },

    Package (0x04)
    {
        0xFFFF,
        0x02,
        LNKC,
        Zero
    },

    Package (0x04)
    {
        0xFFFF,
        0x03,
        LNKD,
        Zero
    }
```

```
})
Name (AR04, Package (0x04)
{
    Package (0x04)
    {
        0xFFFF,
        Zero,
        Zero,
        0x10
    },

    Package (0x04)
    {
        0xFFFF,
        One,
        Zero,
        0x11
    },

    Package (0x04)
    {
        0xFFFF,
        0x02,
        Zero,
        0x12
    },

    Package (0x04)
    {
        0xFFFF,
        0x03,
        Zero,
        0x13
    }
})
Name (PR05, Package (0x04)
{
    Package (0x04)
    {
        0xFFFF,
        Zero,
        LNKB,
        Zero
    },

    Package (0x04)
    {
        0xFFFF,
        One,
```

```
            LNKC,
            Zero
        },

        Package (0x04)
        {
            0xFFFF,
            0x02,
            LNKD,
            Zero
        },

        Package (0x04)
        {
            0xFFFF,
            0x03,
            LNKA,
            Zero
        }
    })
    Name (AR05, Package (0x04)
    {
        Package (0x04)
        {
            0xFFFF,
            Zero,
            Zero,
            0x11
        },

        Package (0x04)
        {
            0xFFFF,
            One,
            Zero,
            0x12
        },

        Package (0x04)
        {
            0xFFFF,
            0x02,
            Zero,
            0x13
        },

        Package (0x04)
        {
            0xFFFF,
```

```
        0x03,
        Zero,
        0x10
    }
})
Name (PRSA, ResourceTemplate ()
{
    IRQ (Level, ActiveLow, Shared, )
        {3,4,5,6,7,10,11,12,14,15}
})
Alias (PRSA, PRSB)
Alias (PRSA, PRSC)
Alias (PRSA, PRSD)
Alias (PRSA, PRSE)
Alias (PRSA, PRSF)
Alias (PRSA, PRSG)
Alias (PRSA, PRSH)
Device (PCI0)
{
    Name (_HID, EisaId ("PNP0A08"))  // _HID: Hardware ID
    Name (_ADR, Zero)  // _ADR: Address
    Method (^BN00, 0, NotSerialized)
    {
        Return (Zero)
    }

    Method (_BBN, 0, NotSerialized)  // _BBN: BIOS Bus Number
    {
        Return (BN00 ())
    }

    Name (_UID, Zero)  // _UID: Unique ID
    Method (_PRT, 0, NotSerialized)  // _PRT: PCI Routing Table
    {
        If (PICM)
        {
            Return (AR00)
        }

        Return (PR00)
    }

    Method (_S3D, 0, NotSerialized)  // _S3D: S3 Device State
    {
        If (LOr (LEqual (OSFL (), One), LEqual (OSFL (), 0x02)))
        {
            Return (0x02)
        }
        Else
```

```
    {
        Return (0x03)
    }
}

Name (_CID, EisaId ("PNP0A03"))  // _CID: Compatible ID
Device (MCH)
{
    Name (_HID, EisaId ("PNP0C01"))  // _HID: Hardware ID
    Name (_UID, 0x0A)  // _UID: Unique ID
    Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource Settings
    {
        Memory32Fixed (ReadWrite,
            0xFED14000,         // Address Base
            0x00006000,         // Address Length
            )
    })
}

Method (NPTS, 1, NotSerialized)
{
}

Method (NWAK, 1, NotSerialized)
{
}

Device (P0P2)
{
    Name (_ADR, 0x00010000)  // _ADR: Address
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x09, 0x04))
    }

    Method (_PRT, 0, NotSerialized)  // _PRT: PCI Routing Table
    {
        If (PICM)
        {
            Return (AR02)
        }

        Return (PR02)
    }
}

Device (P0P3)
{
    Name (_ADR, 0x00060000)  // _ADR: Address
```

```
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x09, 0x04))
    }

    Method (_PRT, 0, NotSerialized)  // _PRT: PCI Routing Table
    {
        If (PICM)
        {
            Return (AR03)
        }

        Return (PR03)
    }
}

Device (P0P1)
{
    Name (_ADR, 0x001E0000)  // _ADR: Address
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x0B, 0x04))
    }

    Method (_PRT, 0, NotSerialized)  // _PRT: PCI Routing Table
    {
        If (PICM)
        {
            Return (AR01)
        }

        Return (PR01)
    }
}

Device (SBRG)
{
    Name (_ADR, 0x001F0000)  // _ADR: Address
    Device (IELK)
    {
        Name (_HID, "AWY0001")  // _HID: Hardware ID
        OperationRegion (RXA0, PCI_Config, 0xA0, 0x20)
        Field (RXA0, ByteAcc, NoLock, Preserve)
        {
                ,   9,
            PBLV,   1,
            Offset (0x10),
                ,   1,
            PBMS,   1,
```

```
            ,   1,
        PMCS,   1,
        ECNS,   1,
        Offset (0x11),
        ECT1,   16,
        ELEN,   1,
        Offset (0x14)
    }

    Method (\_GPE._L0A, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
    {
        Notify (\_SB.PCI0.SBRG.IELK, 0x81)
        Store (One, \_SB.PCI0.SBRG.IELK.PMCS)
    }

    Method (_STA, 0, NotSerialized)  // _STA: Status
    {
        If (ELEN)
        {
            Return (0x0F)
        }
        Else
        {
            Return (Zero)
        }
    }

    Method (SMOD, 1, NotSerialized)
    {
    }

    Method (GPBS, 0, NotSerialized)
    {
        Return (XOr (PBLV, One))
    }
}

Method (SPTS, 1, NotSerialized)
{
    Store (One, PS1S)
    Store (One, PS1E)
    Store (One, SLPS)
}

Method (SWAK, 1, NotSerialized)
{
    Store (Zero, SLPS)
    Store (Zero, PS1E)
    If (LAnd (LEqual (Arg0, One), RTCS)) {}
```

```
    ElseIf (LAnd (LEqual (Arg0, 0x03), BRTC)) {}
    Else
    {
        Notify (PWRB, 0x02)
    }
}

OperationRegion (APMP, SystemIO, SMIO, 0x02)
Field (APMP, ByteAcc, NoLock, Preserve)
{
    APMC,   8,
    APMS,   8
}

Field (APMP, ByteAcc, NoLock, Preserve)
{
    Offset (0x01),
        ,   1,
    BRTC,   1
}

OperationRegion (PMS0, SystemIO, PMBS, 0x04)
Field (PMS0, ByteAcc, NoLock, Preserve)
{
        ,   10,
    RTCS,   1,
        ,   4,
    WAKS,   1,
    Offset (0x03),
    PWBT,   1,
    Offset (0x04)
}

OperationRegion (SMIE, SystemIO, PM30, 0x08)
Field (SMIE, ByteAcc, NoLock, Preserve)
{
        ,   4,
    PS1E,   1,
        ,   31,
    PS1S,   1,
    Offset (0x08)
}

Scope (\_SB)
{
    Name (SLPS, Zero)
    Device (SLPB)
    {
        Name (_HID, EisaId ("PNP0C0E"))  // _HID: Hardware ID
```

```
            Method (_STA, 0, NotSerialized)  // _STA: Status
            {
                If (LNotEqual (SUSW, 0xFF))
                {
                    Return (0x0F)
                }

                Return (Zero)
            }

            Method (SBEV, 0, NotSerialized)
            {
                If (SLPS)
                {
                    Notify (SLPB, 0x02)
                }
                Else
                {
                    Notify (SLPB, 0x80)
                }
            }

            Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for
Wake
            {
                Return (Package (0x02)
                {
                    0x1B,
                    0x04
                })
            }
        }
    }

    Device (PIC)
    {
        Name (_HID, EisaId ("PNP0000"))  // _HID: Hardware ID
        Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource
Settings
        {
            IO (Decode16,
                0x0020,             // Range Minimum
                0x0020,             // Range Maximum
                0x00,               // Alignment
                0x02,               // Length
                )
            IO (Decode16,
                0x00A0,             // Range Minimum
                0x00A0,             // Range Maximum
```

```
                0x00,             // Alignment
                0x02,             // Length
                )
            IRQNoFlags ()
                {2}
        })
    }

    Device (DMAD)
    {
        Name (_HID, EisaId ("PNP0200"))  // _HID: Hardware ID
        Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource
Settings
        {
            DMA (Compatibility, BusMaster, Transfer8, )
                {4}
            IO (Decode16,
                0x0000,           // Range Minimum
                0x0000,           // Range Maximum
                0x00,             // Alignment
                0x10,             // Length
                )
            IO (Decode16,
                0x0081,           // Range Minimum
                0x0081,           // Range Maximum
                0x00,             // Alignment
                0x03,             // Length
                )
            IO (Decode16,
                0x0087,           // Range Minimum
                0x0087,           // Range Maximum
                0x00,             // Alignment
                0x01,             // Length
                )
            IO (Decode16,
                0x0089,           // Range Minimum
                0x0089,           // Range Maximum
                0x00,             // Alignment
                0x03,             // Length
                )
            IO (Decode16,
                0x008F,           // Range Minimum
                0x008F,           // Range Maximum
                0x00,             // Alignment
                0x01,             // Length
                )
            IO (Decode16,
                0x00C0,           // Range Minimum
                0x00C0,           // Range Maximum
```

```
                0x00,               // Alignment
                0x20,               // Length
                )
            })
        }

        Device (TMR)
        {
            Name (_HID, EisaId ("PNP0100"))  // _HID: Hardware ID
            Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource
Settings
            {
                IO (Decode16,
                    0x0040,             // Range Minimum
                    0x0040,             // Range Maximum
                    0x00,               // Alignment
                    0x04,               // Length
                    )
                IRQNoFlags ()
                    {0}
            })
        }

        Device (RTC0)
        {
            Name (_HID, EisaId ("PNP0B00"))  // _HID: Hardware ID
            Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource
Settings
            {
                IO (Decode16,
                    0x0070,             // Range Minimum
                    0x0070,             // Range Maximum
                    0x00,               // Alignment
                    0x02,               // Length
                    )
                IRQNoFlags ()
                    {8}
            })
        }

        Device (SPKR)
        {
            Name (_HID, EisaId ("PNP0800"))  // _HID: Hardware ID
            Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource
Settings
            {
                IO (Decode16,
                    0x0061,             // Range Minimum
                    0x0061,             // Range Maximum
```

```
            0x00,               // Alignment
            0x01,               // Length
            )
        })
    }

    Device (COPR)
    {
        Name (_HID, EisaId ("PNP0C04"))  // _HID: Hardware ID
        Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource
Settings
        {
            IO (Decode16,
                0x00F0,             // Range Minimum
                0x00F0,             // Range Maximum
                0x00,               // Alignment
                0x10,               // Length
                )
            IRQNoFlags ()
                {13}
        })
    }

    Device (LPTE)
    {
        Method (_HID, 0, NotSerialized)  // _HID: Hardware ID
        {
            If (LPTM (0x02))
            {
                Return (0x0104D041)
            }
            Else
            {
                Return (0x0004D041)
            }
        }

        Method (_STA, 0, NotSerialized)  // _STA: Status
        {
            Return (DSTA (0x02))
        }

        Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
        {
            DCNT (0x02, Zero)
        }

        Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
        {
```

```
    DCRS (0x02, One)
    If (LPTM (0x02))
    {
        Store (IRQM, IRQE)
        Store (DMAM, DMAE)
        Store (IO11, IO21)
        Store (IO12, IO22)
        Store (LEN1, LEN2)
        Add (IO21, 0x0400, IO31)
        Store (IO31, IO32)
        Store (LEN2, LEN3)
        Return (CRS2)
    }
    Else
    {
        Return (CRS1)
    }
}

Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
{
    DSRS (Arg0, 0x02)
}

Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
{
    If (LPTM (0x02))
    {
        Return (EPPR)
    }
    Else
    {
        Return (LPPR)
    }
}

Name (LPPR, ResourceTemplate ()
{
    StartDependentFnNoPri ()
    {
        IO (Decode16,
            0x0378,             // Range Minimum
            0x0378,             // Range Maximum
            0x01,               // Alignment
            0x08,               // Length
            )
        IRQNoFlags ()
            {3,4,5,6,7,10,11,12}
        DMA (Compatibility, NotBusMaster, Transfer8, )
```

```
            {}
        }
        StartDependentFnNoPri ()
        {
            IO (Decode16,
                0x0278,             // Range Minimum
                0x0278,             // Range Maximum
                0x01,               // Alignment
                0x08,               // Length
                )
            IRQNoFlags ()
                {3,4,5,6,7,10,11,12}
            DMA (Compatibility, NotBusMaster, Transfer8, )
                {}
        }
        StartDependentFnNoPri ()
        {
            IO (Decode16,
                0x03BC,             // Range Minimum
                0x03BC,             // Range Maximum
                0x01,               // Alignment
                0x04,               // Length
                )
            IRQNoFlags ()
                {3,4,5,6,7,10,11,12}
            DMA (Compatibility, NotBusMaster, Transfer8, )
                {}
        }
        EndDependentFn ()
    })
    Name (EPPR, ResourceTemplate ()
    {
        StartDependentFn (0x00, 0x00)
        {
            IO (Decode16,
                0x0378,             // Range Minimum
                0x0378,             // Range Maximum
                0x01,               // Alignment
                0x08,               // Length
                )
            IO (Decode16,
                0x0778,             // Range Minimum
                0x0778,             // Range Maximum
                0x01,               // Alignment
                0x08,               // Length
                )
            IRQNoFlags ()
                {7}
            DMA (Compatibility, NotBusMaster, Transfer8, )
```

```
        {3}
    }
    StartDependentFnNoPri ()
    {
        IO (Decode16,
            0x0378,             // Range Minimum
            0x0378,             // Range Maximum
            0x01,               // Alignment
            0x08,               // Length
            )
        IO (Decode16,
            0x0778,             // Range Minimum
            0x0778,             // Range Maximum
            0x01,               // Alignment
            0x08,               // Length
            )
        IRQNoFlags ()
            {3,4,5,6,7,10,11,12}
        DMA (Compatibility, NotBusMaster, Transfer8, )
            {0,1,2,3}
    }
    StartDependentFnNoPri ()
    {
        IO (Decode16,
            0x0278,             // Range Minimum
            0x0278,             // Range Maximum
            0x01,               // Alignment
            0x08,               // Length
            )
        IO (Decode16,
            0x0678,             // Range Minimum
            0x0678,             // Range Maximum
            0x01,               // Alignment
            0x08,               // Length
            )
        IRQNoFlags ()
            {3,4,5,6,7,10,11,12}
        DMA (Compatibility, NotBusMaster, Transfer8, )
            {0,1,2,3}
    }
    StartDependentFnNoPri ()
    {
        IO (Decode16,
            0x03BC,             // Range Minimum
            0x03BC,             // Range Maximum
            0x01,               // Alignment
            0x04,               // Length
            )
        IO (Decode16,
```

```
                0x07BC,             // Range Minimum
                0x07BC,             // Range Maximum
                0x01,               // Alignment
                0x04,               // Length
                )
            IRQNoFlags ()
                {3,4,5,6,7,10,11,12}
            DMA (Compatibility, NotBusMaster, Transfer8, )
                {0,1,2,3}
        }
        EndDependentFn ()
    })
}

Device (SIOR)
{
    Name (_HID, EisaId ("PNP0C02"))  // _HID: Hardware ID
    Method (_UID, 0, NotSerialized)  // _UID: Unique ID
    {
        Return (SPIO)
    }

    Name (CRS, ResourceTemplate ()
    {
        IO (Decode16,
            0x0000,             // Range Minimum
            0x0000,             // Range Maximum
            0x00,               // Alignment
            0x00,               // Length
            _Y00)
        IO (Decode16,
            0x0000,             // Range Minimum
            0x0000,             // Range Maximum
            0x00,               // Alignment
            0x00,               // Length
            )
        IO (Decode16,
            0x0290,             // Range Minimum
            0x0290,             // Range Maximum
            0x00,               // Alignment
            0x08,               // Length
            )
    })
    Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
    {
        If (LAnd (LNotEqual (SPIO, 0x03F0), LGreater (SPIO, 0xF0)))
        {
            CreateWordField (CRS, \_SB.PCI0.SBRG.SIOR._Y00._MIN,
GP10)  // _MIN: Minimum Base Address
```

```
                    CreateWordField (CRS, \_SB.PCI0.SBRG.SIOR._Y00._MAX,
GP11)  // _MAX: Maximum Base Address
                    CreateByteField (CRS, \_SB.PCI0.SBRG.SIOR._Y00._LEN, GPL1)
// _LEN: Length
                    Store (SPIO, GP10)
                    Store (SPIO, GP11)
                    Store (0x02, GPL1)
                }

                Return (CRS)
            }
        }

        Name (DCAT, Package (0x15)
        {
            0x02,
            0x03,
            One,
            Zero,
            0xFF,
            0x07,
            0xFF,
            0xFF,
            0x07,
            0xFF,
            0xFF,
            0xFF,
            0xFF,
            0xFF,
            0xFF,
            0xFF,
            0xFF,
            0x07,
            0x08,
            0x09,
            0xFF,
            0xFF
        })
        Mutex (IOCF, 0x00)
        Method (ENFG, 1, NotSerialized)
        {
            Acquire (IOCF, 0xFFFF)
            Store (0x87, INDX)
            Store (0x87, INDX)
            Store (Arg0, LDN)
        }

        Method (EXFG, 0, NotSerialized)
        {
            Store (0xAA, INDX)
```

```
    Release (IOCF)
}

Method (LPTM, 1, NotSerialized)
{
    ENFG (CGLD (Arg0))
    And (OPT0, 0x02, Local0)
    EXFG ()
    Return (Local0)
}

Method (UHID, 1, NotSerialized)
{
    If (LEqual (Arg0, One))
    {
        ENFG (CGLD (Arg0))
        And (OPT1, 0x38, Local0)
        EXFG ()
        If (Local0)
        {
            Return (0x1005D041)
        }
    }

    Return (0x0105D041)
}

Name (KBFG, One)
Name (MSFG, One)
Method (SIOK, 1, NotSerialized)
{
    ENFG (0x0A)
    If (And (LGreater (Arg0, One), LNotEqual (Arg0, 0x05)))
    {
        If (LEqual (Arg0, 0x03))
        {
            Or (CRE4, 0x10, CRE4)
        }

        And (CRE0, 0x04, CRE0)
        Store (CRE3, Local0)
        If (KBFG)
        {
            Or (CRE0, 0x41, CRE0)
        }

        If (MSFG)
        {
            Or (CRE0, 0x20, CRE0)
```

```
            And (CRE0, 0x7F, CRE0)
        }
    }
    ElseIf (LEqual (Arg0, One))
    {
        Store (Zero, OPT6)
        Or (OPT3, 0xFF, OPT3)
        Or (OPT4, 0xFF, OPT4)
        If (KBFG)
        {
            Or (OPT6, 0x10, OPT6)
        }

        If (MSFG)
        {
            Or (OPT6, 0x20, OPT6)
        }
    }

    Store (CRE3, Local0)
    EXFG ()
}

Method (SIOS, 1, NotSerialized)
{
    Store ("SIOS", Debug)
    SIOK (Arg0)
}

Method (SIOW, 1, NotSerialized)
{
    Store ("SIOW", Debug)
    SIOK (Zero)
}

Method (SIOH, 0, NotSerialized)
{
    Store ("SIOH", Debug)
    ENFG (0x0A)
    If (And (OPT3, 0x10))
    {
        Notify (PS2K, 0x02)
    }

    If (And (OPT3, 0x20))
    {
        Notify (PS2M, 0x02)
    }
```

```
    EXFG ()
    SIOK (Zero)
}

OperationRegion (IOID, SystemIO, SPIO, 0x02)
Field (IOID, ByteAcc, NoLock, Preserve)
{
    INDX,   8,
    DATA,   8
}

IndexField (INDX, DATA, ByteAcc, NoLock, Preserve)
{
    Offset (0x07),
    LDN,    8,
    Offset (0x22),
    FDCP,   1,
        ,   2,
    LPTP,   1,
    URAP,   1,
    URBP,   1,
    Offset (0x30),
    ACTR,   8,
    Offset (0x60),
    IOAH,   8,
    IOAL,   8,
    IOH2,   8,
    IOL2,   8,
    Offset (0x70),
    INTR,   4,
    Offset (0x74),
    DMCH,   3,
    Offset (0xE0),
    CRE0,   8,
    CRE1,   8,
    CRE2,   8,
    CRE3,   8,
    CRE4,   8,
    CRE5,   8,
    CRE6,   8,
    Offset (0xF0),
    OPT0,   8,
    OPT1,   8,
    OPT2,   8,
    OPT3,   8,
    OPT4,   8,
    OPT5,   8,
    OPT6,   8,
    Offset (0xF9),
```

```
    OPT9,   8
}

Method (CGLD, 1, NotSerialized)
{
    Return (DerefOf (Index (DCAT, Arg0)))
}

Method (DSTA, 1, NotSerialized)
{
    ENFG (CGLD (Arg0))
    Store (ACTR, Local0)
    EXFG ()
    If (LEqual (Local0, 0xFF))
    {
        Return (Zero)
    }

    If (LEqual (Arg0, 0x05))
    {
        ShiftRight (Local0, 0x02, Local0)
    }

    If (LEqual (Arg0, 0x08))
    {
        ShiftRight (Local0, One, Local0)
    }

    And (Local0, One, Local0)
    Or (IOST, ShiftLeft (Local0, Arg0), IOST)
    If (Local0)
    {
        Return (0x0F)
    }
    ElseIf (And (ShiftLeft (One, Arg0), IOST))
    {
        Return (0x0D)
    }
    Else
    {
        Return (Zero)
    }
}

Method (DCNT, 2, NotSerialized)
{
    ENFG (CGLD (Arg0))
    If (LEqual (Arg0, 0x05))
    {
```

```
                ShiftLeft (IOH2, 0x08, Local1)
                Or (IOL2, Local1, Local1)
            }
            Else
            {
                ShiftLeft (IOAH, 0x08, Local1)
                Or (IOAL, Local1, Local1)
            }

            RRIO (Arg0, Arg1, Local1, 0x08)
            If (LAnd (LLess (DMCH, 0x04), LNotEqual (And (DMCH, 0x03, Local1),
Zero)))
            {
                RDMA (Arg0, Arg1, Increment (Local1))
            }

            Store (Arg1, Local1)
            Store (One, Local2)
            If (LEqual (Arg0, 0x05))
            {
                ShiftLeft (Arg1, 0x02, Local1)
                ShiftLeft (Local2, 0x02, Local2)
            }

            If (LEqual (Arg0, 0x08))
            {
                ShiftLeft (Arg1, One, Local1)
                ShiftLeft (Local2, One, Local2)
            }

            Store (ACTR, Local0)
            Not (Local2, Local3)
            And (Local0, Local3, Local0)
            Or (Local0, Local1, Local0)
            Store (Local0, ACTR)
            EXFG ()
        }

        Name (CRS1, ResourceTemplate ()
        {
            IRQNoFlags ()
                {}
            DMA (Compatibility, NotBusMaster, Transfer8, _Y01)
                {}
            IO (Decode16,
                0x0000,             // Range Minimum
                0x0000,             // Range Maximum
                0x01,               // Alignment
                0x00,               // Length
```

```
                _Y02)
            })
            CreateWordField (CRS1, One, IRQM)
            CreateByteField (CRS1, \_SB.PCI0.SBRG._Y01._DMA, DMAM)  // _DMA:
Direct Memory Access
            CreateWordField (CRS1, \_SB.PCI0.SBRG._Y02._MIN, IO11)  // _MIN:
Minimum Base Address
            CreateWordField (CRS1, \_SB.PCI0.SBRG._Y02._MAX, IO12)  // _MAX:
Maximum Base Address
            CreateByteField (CRS1, \_SB.PCI0.SBRG._Y02._LEN, LEN1)  // _LEN:
Length
            Name (CRS2, ResourceTemplate ()
            {
               IRQNoFlags ()
                  {6}
               DMA (Compatibility, NotBusMaster, Transfer8, _Y03)
                  {2}
               IO (Decode16,
                  0x0000,          // Range Minimum
                  0x0000,          // Range Maximum
                  0x01,            // Alignment
                  0x00,            // Length
                  _Y04)
               IO (Decode16,
                  0x0000,          // Range Minimum
                  0x0000,          // Range Maximum
                  0x01,            // Alignment
                  0x00,            // Length
                  _Y05)
            })
            CreateWordField (CRS2, One, IRQE)
            CreateByteField (CRS2, \_SB.PCI0.SBRG._Y03._DMA, DMAE)  // _DMA:
Direct Memory Access
            CreateWordField (CRS2, \_SB.PCI0.SBRG._Y04._MIN, IO21)  // _MIN:
Minimum Base Address
            CreateWordField (CRS2, \_SB.PCI0.SBRG._Y04._MAX, IO22)  // _MAX:
Maximum Base Address
            CreateByteField (CRS2, \_SB.PCI0.SBRG._Y04._LEN, LEN2)  // _LEN:
Length
            CreateWordField (CRS2, \_SB.PCI0.SBRG._Y05._MIN, IO31)  // _MIN:
Minimum Base Address
            CreateWordField (CRS2, \_SB.PCI0.SBRG._Y05._MAX, IO32)  // _MAX:
Maximum Base Address
            CreateByteField (CRS2, \_SB.PCI0.SBRG._Y05._LEN, LEN3)  // _LEN:
Length
            Method (DCRS, 2, NotSerialized)
            {
               ENFG (CGLD (Arg0))
               ShiftLeft (IOAH, 0x08, IO11)
```

```
    Or (IOAL, IO11, IO11)
    Store (IO11, IO12)
    Subtract (FindSetRightBit (IO11), One, Local0)
    ShiftLeft (One, Local0, LEN1)
    If (INTR)
    {
        ShiftLeft (One, INTR, IRQM)
    }
    Else
    {
        Store (Zero, IRQM)
    }

    If (LOr (LGreater (DMCH, 0x03), LEqual (Arg1, Zero)))
    {
        Store (Zero, DMAM)
    }
    Else
    {
        And (DMCH, 0x03, Local1)
        ShiftLeft (One, Local1, DMAM)
    }

    EXFG ()
    Return (CRS1)
}

Method (DSRS, 2, NotSerialized)
{
    CreateWordField (Arg0, One, IRQM)
    CreateByteField (Arg0, 0x04, DMAM)
    CreateWordField (Arg0, 0x08, IO11)
    ENFG (CGLD (Arg1))
    And (IO11, 0xFF, IOAL)
    ShiftRight (IO11, 0x08, IOAH)
    If (IRQM)
    {
        FindSetRightBit (IRQM, Local0)
        Subtract (Local0, One, INTR)
    }
    Else
    {
        Store (Zero, INTR)
    }

    If (DMAM)
    {
        FindSetRightBit (DMAM, Local0)
        Subtract (Local0, One, DMCH)
```

```
    }
    Else
    {
        Store (0x07, DMCH)
    }

    EXFG ()
    DCNT (Arg1, One)
}

Device (RMSC)
{
    Name (_HID, EisaId ("PNP0C02"))  // _HID: Hardware ID
    Name (_UID, 0x10)  // _UID: Unique ID
    Name (CRS, ResourceTemplate ()
    {
        IO (Decode16,
            0x0010,             // Range Minimum
            0x0010,             // Range Maximum
            0x00,               // Alignment
            0x10,               // Length
            )
        IO (Decode16,
            0x0022,             // Range Minimum
            0x0022,             // Range Maximum
            0x00,               // Alignment
            0x1E,               // Length
            )
        IO (Decode16,
            0x0044,             // Range Minimum
            0x0044,             // Range Maximum
            0x00,               // Alignment
            0x1C,               // Length
            )
        IO (Decode16,
            0x0062,             // Range Minimum
            0x0062,             // Range Maximum
            0x00,               // Alignment
            0x02,               // Length
            )
        IO (Decode16,
            0x0065,             // Range Minimum
            0x0065,             // Range Maximum
            0x00,               // Alignment
            0x0B,               // Length
            )
        IO (Decode16,
            0x0072,             // Range Minimum
            0x0072,             // Range Maximum
```

```
    0x00,               // Alignment
    0x0E,               // Length
    )
IO (Decode16,
    0x0080,             // Range Minimum
    0x0080,             // Range Maximum
    0x00,               // Alignment
    0x01,               // Length
    )
IO (Decode16,
    0x0084,             // Range Minimum
    0x0084,             // Range Maximum
    0x00,               // Alignment
    0x03,               // Length
    )
IO (Decode16,
    0x0088,             // Range Minimum
    0x0088,             // Range Maximum
    0x00,               // Alignment
    0x01,               // Length
    )
IO (Decode16,
    0x008C,             // Range Minimum
    0x008C,             // Range Maximum
    0x00,               // Alignment
    0x03,               // Length
    )
IO (Decode16,
    0x0090,             // Range Minimum
    0x0090,             // Range Maximum
    0x00,               // Alignment
    0x10,               // Length
    )
IO (Decode16,
    0x00A2,             // Range Minimum
    0x00A2,             // Range Maximum
    0x00,               // Alignment
    0x1E,               // Length
    )
IO (Decode16,
    0x00E0,             // Range Minimum
    0x00E0,             // Range Maximum
    0x00,               // Alignment
    0x10,               // Length
    )
IO (Decode16,
    0x0400,             // Range Minimum
    0x0400,             // Range Maximum
    0x00,               // Alignment
```

```
                    0x20,               // Length
                    )
                IO (Decode16,
                    0x04D0,             // Range Minimum
                    0x04D0,             // Range Maximum
                    0x00,               // Alignment
                    0x02,               // Length
                    )
                IO (Decode16,
                    0x0000,             // Range Minimum
                    0x0000,             // Range Maximum
                    0x00,               // Alignment
                    0x00,               // Length
                    _Y06)
                IO (Decode16,
                    0x0000,             // Range Minimum
                    0x0000,             // Range Maximum
                    0x00,               // Alignment
                    0x00,               // Length
                    _Y07)
                IO (Decode16,
                    0x0000,             // Range Minimum
                    0x0000,             // Range Maximum
                    0x00,               // Alignment
                    0x00,               // Length
                    _Y08)
                Memory32Fixed (ReadWrite,
                    0xFED1C000,         // Address Base
                    0x00004000,         // Address Length
                    )
                Memory32Fixed (ReadWrite,
                    0xFED20000,         // Address Base
                    0x00070000,         // Address Length
                    )
            })
            Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
            {
                CreateWordField (CRS, \_SB.PCI0.SBRG.RMSC._Y06._MIN, GP00)
// _MIN: Minimum Base Address
                CreateWordField (CRS, \_SB.PCI0.SBRG.RMSC._Y06._MAX, GP01)
// _MAX: Maximum Base Address
                CreateByteField (CRS, \_SB.PCI0.SBRG.RMSC._Y06._LEN, GP0L)
// _LEN: Length
                Store (PMBS, GP00)
                Store (PMBS, GP01)
                Store (PMLN, GP0L)
                If (SMBS)
                {
                    CreateWordField (CRS, \_SB.PCI0.SBRG.RMSC._Y07._MIN,
```

```
GP10)  // _MIN: Minimum Base Address
                    CreateWordField (CRS, \_SB.PCI0.SBRG.RMSC._Y07._MAX,
GP11)  // _MAX: Maximum Base Address
                    CreateByteField (CRS, \_SB.PCI0.SBRG.RMSC._Y07._LEN,
GP1L)  // _LEN: Length
                    Store (SMBS, GP10)
                    Store (SMBS, GP11)
                    Store (SMBL, GP1L)
                }

                If (GPBS)
                {
                    CreateWordField (CRS, \_SB.PCI0.SBRG.RMSC._Y08._MIN,
GP20)  // _MIN: Minimum Base Address
                    CreateWordField (CRS, \_SB.PCI0.SBRG.RMSC._Y08._MAX,
GP21)  // _MAX: Maximum Base Address
                    CreateByteField (CRS, \_SB.PCI0.SBRG.RMSC._Y08._LEN,
GP2L)  // _LEN: Length
                    Store (GPBS, GP20)
                    Store (GPBS, GP21)
                    Store (GPLN, GP2L)
                }

                Return (CRS)
            }
        }

        Scope (\)
        {
            OperationRegion (RAMW, SystemMemory, 0xCFFF0000, 0x00010000)
            Field (RAMW, ByteAcc, NoLock, Preserve)
            {
                PAR0,   32,
                PAR1,   32,
                PAR2,   32
            }

            OperationRegion (IOB2, SystemIO, 0xB2, 0x02)
            Field (IOB2, ByteAcc, NoLock, Preserve)
            {
                SMIC,   8,
                SMIS,   8
            }

            Method (ISMI, 1, Serialized)
            {
                Store (Arg0, SMIC)
            }
```

```
    Method (GNVS, 1, Serialized)
    {
        Store (Arg0, PAR0)
        ISMI (0x70)
        Return (PAR1)
    }

    Method (SNVS, 2, Serialized)
    {
        Store (Arg0, PAR0)
        Store (Arg1, PAR1)
        ISMI (0x71)
    }

    Method (GMAX, 1, Serialized)
    {
        Store (Arg0, PAR0)
        ISMI (0x74)
        Return (PAR1)
    }

    Method (GMDX, 1, Serialized)
    {
        Store (Arg0, PAR0)
        ISMI (0x75)
        Return (PAR1)
    }

    Method (GCAX, 1, Serialized)
    {
        Store (Arg0, PAR0)
        ISMI (0x76)
        Return (PAR1)
    }

    Method (GCDX, 1, Serialized)
    {
        Store (Arg0, PAR0)
        ISMI (0x77)
        Return (PAR1)
    }
}

Scope (\_SB.PCI0.SBRG)
{
    Device (ASOC)
    {
        Name (_HID, "ATK0110")  // _HID: Hardware ID
        Name (_UID, 0x01010110)  // _UID: Unique ID
```

```
Method (_STA, 0, NotSerialized)  // _STA: Status
{
    Return (0x0F)
}

Method (_INI, 0, NotSerialized)  // _INI: Initialize
{
    FSBI ()
}

Name (MBIF, Package (0x08)
{
    0x03,
    "P5QPL-AM",
    0x01010103,
    0x02000100,
    0xE0000000,
    Zero,
    Zero,
    Zero
})
Name (ASBF, Buffer (0x0100) {})
CreateDWordField (ASBF, Zero, ASB0)
CreateDWordField (ASBF, 0x04, ASB1)
Method (GGRP, 1, Serialized)
{
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (Arg0, _T_0)
    If (LEqual (_T_0, Zero))
    {
        Return (GRP0)
    }
    ElseIf (LEqual (_T_0, 0x03))
    {
        Return (GRP3)
    }
    ElseIf (LEqual (_T_0, 0x04))
    {
        Return (GRP4)
    }
    ElseIf (LEqual (_T_0, 0x05))
    {
        Return (GRP5)
    }
    ElseIf (LEqual (_T_0, 0x06))
    {
        Return (GRP6)
    }
    Else
```

```
    {
        Return (Zero)
    }
}

Method (GITM, 1, Serialized)
{
    CreateDWordField (Arg0, Zero, PRM0)
    CreateDWordField (Arg0, 0x04, PRM1)
    CreateByteField (Arg0, 0x03, GPID)
    Store (One, ASB0)
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (GPID, _T_0)
    If (LEqual (_T_0, Zero))
    {
        GIT0 (PRM0)
    }
    ElseIf (LEqual (_T_0, 0x03))
    {
        GIT3 (PRM0)
    }
    ElseIf (LEqual (_T_0, 0x04))
    {
        GIT4 (PRM0)
    }
    ElseIf (LEqual (_T_0, 0x05))
    {
        GIT5 (PRM0, PRM1)
    }
    ElseIf (LEqual (_T_0, 0x06))
    {
        GIT6 (PRM0)
    }
    Else
    {
        Store (Zero, ASB0)
    }

    Return (ASBF)
}

Method (SITM, 1, Serialized)
{
    CreateDWordField (Arg0, Zero, PRM0)
    CreateDWordField (Arg0, 0x04, PRM1)
    CreateDWordField (Arg0, 0x08, PRM2)
    CreateByteField (Arg0, 0x03, GPID)
    Store (One, ASB0)
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
```

```
            Store (GPID, _T_0)
            If (LEqual (_T_0, Zero))
            {
                SIT0 (PRM0, PRM1, PRM2)
            }
            ElseIf (LEqual (_T_0, 0x03))
            {
                SIT3 (PRM0, PRM1, PRM2)
            }
            ElseIf (LEqual (_T_0, 0x04))
            {
                SIT4 (PRM0, PRM1, PRM2)
            }
            ElseIf (LEqual (_T_0, 0x05))
            {
                SIT5 (PRM0, PRM1, PRM2)
            }
            ElseIf (LEqual (_T_0, 0x06))
            {
                SIT6 (PRM0, PRM1, PRM2)
            }
            Else
            {
                Store (Zero, ASB0)
            }

            Return (ASBF)
        }

        Method (OP2V, 2, NotSerialized)
        {
            Store (DerefOf (Index (Arg1, 0x04)), Local0)
            Store (DerefOf (Index (Arg1, 0x05)), Local1)
            Multiply (Arg0, Local1, Local1)
            Add (Local0, Local1, Local0)
            Return (Local0)
        }

        Method (V2OP, 2, NotSerialized)
        {
            Store (DerefOf (Index (Arg1, 0x04)), Local0)
            Store (DerefOf (Index (Arg1, 0x05)), Local1)
            Subtract (Arg0, Local0, Local0)
            Divide (Local0, Local1, Local1, Local0)
            Return (Local0)
        }
    }
}
```

```
            Device (HPET)
            {
               Name (_HID, EisaId ("PNP0103"))  // _HID: Hardware ID
               Name (CRS, ResourceTemplate ()
               {
                  Memory32Fixed (ReadOnly,
                     0xFED00000,         // Address Base
                     0x00000400,         // Address Length
                     _Y09)
               })
               OperationRegion (^LPCR, SystemMemory, 0xFED1F404, 0x04)
               Field (LPCR, AnyAcc, NoLock, Preserve)
               {
                  HPTS,   2,
                      ,   5,
                  HPTE,   1,
                  Offset (0x04)
               }

               Method (_STA, 0, NotSerialized)  // _STA: Status
               {
                  If (LEqual (OSFL (), Zero))
                  {
                     If (HPTE)
                     {
                        Return (0x0F)
                     }
                  }
                  ElseIf (HPTE)
                  {
                     Return (0x0B)
                  }

                  Return (Zero)
               }

               Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
               {
                  CreateDWordField (CRS, \_SB.PCI0.SBRG.HPET._Y09._BAS, HPT)
// _BAS: Base Address
                  Multiply (HPTS, 0x1000, Local0)
                  Add (Local0, 0xFED00000, HPT)
                  Return (CRS)
               }
            }

            OperationRegion (RX80, PCI_Config, Zero, 0xFF)
            Field (RX80, ByteAcc, NoLock, Preserve)
            {
```

```
    Offset (0x80),
    LPCD,   16,
    LPCE,   16
}

Name (DBPT, Package (0x04)
{
    Package (0x08)
    {
        0x03F8,
        0x02F8,
        0x0220,
        0x0228,
        0x0238,
        0x02E8,
        0x0338,
        0x03E8
    },

    Package (0x08)
    {
        0x03F8,
        0x02F8,
        0x0220,
        0x0228,
        0x0238,
        0x02E8,
        0x0338,
        0x03E8
    },

    Package (0x03)
    {
        0x0378,
        0x0278,
        0x03BC
    },

    Package (0x02)
    {
        0x03F0,
        0x0370
    }
})
Name (DDLT, Package (0x04)
{
    Package (0x02)
    {
        Zero,
```

```
                0xFFF8
            },

            Package (0x02)
            {
                0x04,
                0xFF8F
            },

            Package (0x02)
            {
                0x08,
                0xFCFF
            },

            Package (0x02)
            {
                0x0C,
                0xEFFF
            }
        })
        Method (RRIO, 4, NotSerialized)
        {
            If (LAnd (LLessEqual (Arg0, 0x03), LGreaterEqual (Arg0, Zero)))
            {
                Store (Match (DerefOf (Index (DBPT, Arg0)), MEQ, Arg2, MTR, Zero,
Zero), Local0)
                If (LNotEqual (Local0, Ones))
                {
                    Store (DerefOf (Index (DerefOf (Index (DDLT, Arg0)), Zero)),
Local1)
                    Store (DerefOf (Index (DerefOf (Index (DDLT, Arg0)), One)),
Local2)
                    ShiftLeft (Local0, Local1, Local0)
                    And (LPCD, Local2, LPCD)
                    Or (LPCD, Local0, LPCD)
                    WX82 (Arg0, Arg1)
                }
            }

            If (LEqual (Arg0, 0x08))
            {
                If (LEqual (Arg2, 0x0200))
                {
                    WX82 (0x08, Arg0)
                }
                ElseIf (LEqual (Arg2, 0x0208))
                {
                    WX82 (0x09, Arg0)
```

```
                }
            }

            If (LAnd (LLessEqual (Arg0, 0x0D), LGreaterEqual (Arg0, 0x0A)))
            {
                WX82 (Arg0, Arg1)
            }
        }

        Method (WX82, 2, NotSerialized)
        {
            ShiftLeft (One, Arg0, Local0)
            If (Arg1)
            {
                Or (LPCE, Local0, LPCE)
            }
            Else
            {
                Not (Local0, Local0)
                And (LPCE, Local0, LPCE)
            }
        }

        Method (RDMA, 3, NotSerialized)
        {
        }

        Device (FWH)
        {
            Name (_HID, EisaId ("INT0800"))  // _HID: Hardware ID
            Name (CRS, ResourceTemplate ()
            {
                Memory32Fixed (ReadOnly,
                    0x00000000,         // Address Base
                    0x00000000,         // Address Length
                    _Y0A)
                Memory32Fixed (ReadOnly,
                    0x00000000,         // Address Base
                    0x00000000,         // Address Length
                    _Y0B)
            })
            CreateDWordField (CRS, \_SB.PCI0.SBRG.FWH._Y0A._BAS, BS00)  //
_BAS: Base Address
            CreateDWordField (CRS, \_SB.PCI0.SBRG.FWH._Y0A._LEN, BL00)  //
_LEN: Length
            CreateDWordField (CRS, \_SB.PCI0.SBRG.FWH._Y0B._BAS, BS10)  //
_BAS: Base Address
            CreateDWordField (CRS, \_SB.PCI0.SBRG.FWH._Y0B._LEN, BL10)  //
_LEN: Length
```

```
            Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
            {
                Store (0xFF800000, Local0)
                FindSetRightBit (FHD0, Local1)
                Decrement (Local1)
                If (Local1)
                {
                    Multiply (Local1, 0x00080000, Local1)
                }

                Add (Local0, Local1, Local2)
                Store (Local2, BS00)
                Add (BS00, 0x00400000, BS10)
                Subtract (Zero, BS10, BL00)
                Store (BL00, BL10)
                Return (CRS)
            }
        }

        Device (FWHE)
        {
            Name (_HID, EisaId ("PNP0C02"))  // _HID: Hardware ID
            Name (_UID, 0x03)  // _UID: Unique ID
            Name (CRS, ResourceTemplate ()
            {
                Memory32Fixed (ReadOnly,
                    0x00000000,         // Address Base
                    0x00000000,         // Address Length
                    _Y0C)
            })
            Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
            {
                CreateDWordField (CRS, \_SB.PCI0.SBRG.FWHE._Y0C._BAS,
BS00)  // _BAS: Base Address
                CreateDWordField (CRS, \_SB.PCI0.SBRG.FWHE._Y0C._LEN,
BL00)  // _LEN: Length
                If (LEqual (^^FWH.BS00, Zero))
                {
                    ^^FWH._CRS ()
                }

                Add (^^FWH.BS00, ^^FWH.BL00, BS00)
                Subtract (^^FWH.BS10, BS00, BL00)
                Return (CRS)
            }
        }

        OperationRegion (FHR0, PCI_Config, 0xD8, 0x02)
        Field (FHR0, ByteAcc, NoLock, Preserve)
```

```
            {
                FHD1,   4,
                Offset (0x01),
                FHD0,   8
            }

            Device (^PCIE)
            {
                Name (_HID, EisaId ("PNP0C02"))  // _HID: Hardware ID
                Name (_UID, 0x11)  // _UID: Unique ID
                Name (CRS, ResourceTemplate ()
                {
                    Memory32Fixed (ReadOnly,
                        0xE0000000,         // Address Base
                        0x10000000,         // Address Length
                        _Y0D)
                })
                Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
                {
                    CreateDWordField (CRS, \_SB.PCI0.PCIE._Y0D._BAS, BAS1)  //
_BAS: Base Address
                    CreateDWordField (CRS, \_SB.PCI0.PCIE._Y0D._LEN, LEN1)  //
_LEN: Length
                    Store (PCIB, BAS1)
                    Store (PCIL, LEN1)
                    Return (CRS)
                }
            }

            Scope (\_GPE)
            {
            }

            Scope (ASOC)
            {
                Name (VESL, Zero)
                Method (SPLV, 1, Serialized)
                {
                    And (Arg0, 0xFFFF, VESL)
                    Store (VESL, PAR0)
                    ISMI (0x88)
                    Store (And (PAR0, 0xFFFF), Local0)
                    Return (Local0)
                }

                Method (GPLV, 0, Serialized)
                {
                    Return (VESL)
                }
```

```
}

Device (UAR1)
{
   Name (_UID, One)  // _UID: Unique ID
   Name (_HID, EisaId ("PNP0501"))  // _HID: Hardware ID
   Method (_STA, 0, NotSerialized)  // _STA: Status
   {
      Return (DSTA (Zero))
   }

   Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
   {
      DCNT (Zero, Zero)
   }

   Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
   {
      Return (DCRS (Zero, Zero))
   }

   Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
   {
      DSRS (Arg0, Zero)
   }

   Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
   {
      Return (CMPR)
   }

   Name (CMPR, ResourceTemplate ()
   {
      StartDependentFn (0x00, 0x00)
      {
         IO (Decode16,
            0x03F8,           // Range Minimum
            0x03F8,           // Range Maximum
            0x01,             // Alignment
            0x08,             // Length
            )
         IRQNoFlags ()
            {4}
         DMA (Compatibility, NotBusMaster, Transfer8, )
            {}
      }
      StartDependentFnNoPri ()
      {
         IO (Decode16,
```

```
        0x03F8,             // Range Minimum
        0x03F8,             // Range Maximum
        0x01,               // Alignment
        0x08,               // Length
        )
    IRQNoFlags ()
        {3,4,5,6,7,10,11,12}
    DMA (Compatibility, NotBusMaster, Transfer8, )
        {}
}
StartDependentFnNoPri ()
{
    IO (Decode16,
        0x02F8,             // Range Minimum
        0x02F8,             // Range Maximum
        0x01,               // Alignment
        0x08,               // Length
        )
    IRQNoFlags ()
        {3,4,5,6,7,10,11,12}
    DMA (Compatibility, NotBusMaster, Transfer8, )
        {}
}
StartDependentFnNoPri ()
{
    IO (Decode16,
        0x03E8,             // Range Minimum
        0x03E8,             // Range Maximum
        0x01,               // Alignment
        0x08,               // Length
        )
    IRQNoFlags ()
        {3,4,5,6,7,10,11,12}
    DMA (Compatibility, NotBusMaster, Transfer8, )
        {}
}
StartDependentFnNoPri ()
{
    IO (Decode16,
        0x02E8,             // Range Minimum
        0x02E8,             // Range Maximum
        0x01,               // Alignment
        0x08,               // Length
        )
    IRQNoFlags ()
        {3,4,5,6,7,10,11,12}
    DMA (Compatibility, NotBusMaster, Transfer8, )
        {}
}
```

```
            EndDependentFn ()
        })
    }

    Method (UAR1._PRW, 0, NotSerialized)  // _PRW: Power Resources for
Wake
    {
        Return (GPRW (0x08, 0x04))
    }

    Device (OMSC)
    {
        Name (_HID, EisaId ("PNP0C02"))  // _HID: Hardware ID
        Name (_UID, Zero)  // _UID: Unique ID
        Name (CRS, ResourceTemplate ()
        {
            IO (Decode16,
                0x0000,             // Range Minimum
                0x0000,             // Range Maximum
                0x00,               // Alignment
                0x00,               // Length
                _Y10)
            IO (Decode16,
                0x0000,             // Range Minimum
                0x0000,             // Range Maximum
                0x00,               // Alignment
                0x00,               // Length
                _Y11)
            Memory32Fixed (ReadOnly,
                0x00000000,         // Address Base
                0x00000000,         // Address Length
                _Y0E)
            Memory32Fixed (ReadOnly,
                0x00000000,         // Address Base
                0x00000000,         // Address Length
                _Y0F)
        })
        Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
        {
            If (APIC)
            {
                CreateDWordField (CRS, \_SB.PCI0.SBRG.OMSC._Y0E._LEN,
ML01)  // _LEN: Length
                CreateDWordField (CRS, \_SB.PCI0.SBRG.OMSC._Y0E._BAS,
MB01)  // _BAS: Base Address
                CreateDWordField (CRS, \_SB.PCI0.SBRG.OMSC._Y0F._LEN,
ML02)  // _LEN: Length
                CreateDWordField (CRS, \_SB.PCI0.SBRG.OMSC._Y0F._BAS,
MB02)  // _BAS: Base Address
```

```
                    Store (0xFEC00000, MB01)
                    Store (0x1000, ML01)
                    Store (0xFEE00000, MB02)
                    Store (0x1000, ML02)
                }

                ShiftLeft (One, 0x0A, Local0)
                If (And (IOST, Local0)) {}
                Else
                {
                    ShiftLeft (One, 0x0C, Local1)
                    If (And (IOST, Local1)) {}
                    Else
                    {
                        CreateWordField (CRS, \_SB.PCI0.SBRG.OMSC._Y10._MIN,
P60N)  // _MIN: Minimum Base Address
                        CreateWordField (CRS, \_SB.PCI0.SBRG.OMSC._Y10._MAX,
P60M)  // _MAX: Maximum Base Address
                        CreateByteField (CRS, \_SB.PCI0.SBRG.OMSC._Y10._LEN,
P60L)  // _LEN: Length
                        CreateWordField (CRS, \_SB.PCI0.SBRG.OMSC._Y11._MIN,
P64N)  // _MIN: Minimum Base Address
                        CreateWordField (CRS, \_SB.PCI0.SBRG.OMSC._Y11._MAX,
P64M)  // _MAX: Maximum Base Address
                        CreateByteField (CRS, \_SB.PCI0.SBRG.OMSC._Y11._LEN,
P64L)  // _LEN: Length
                        Store (0x60, P60N)
                        Store (0x60, P60M)
                        Store (One, P60L)
                        Store (0x64, P64N)
                        Store (0x64, P64M)
                        Store (One, P64L)
                    }
                }

                Return (CRS)
            }
        }

        Device (^^RMEM)
        {
            Name (_HID, EisaId ("PNP0C01"))  // _HID: Hardware ID
            Name (_UID, One)  // _UID: Unique ID
            Name (CRS, ResourceTemplate ()
            {
                Memory32Fixed (ReadWrite,
                    0x00000000,         // Address Base
                    0x000A0000,         // Address Length
                    )
```

```
                Memory32Fixed (ReadOnly,
                    0x00000000,         // Address Base
                    0x00000000,         // Address Length
                    _Y12)
                Memory32Fixed (ReadOnly,
                    0x000E0000,         // Address Base
                    0x00020000,         // Address Length
                    _Y13)
                Memory32Fixed (ReadWrite,
                    0x00100000,         // Address Base
                    0x00000000,         // Address Length
                    _Y14)
                Memory32Fixed (ReadOnly,
                    0x00000000,         // Address Base
                    0x00000000,         // Address Length
                    _Y15)
            })
            Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
            {
                CreateDWordField (CRS, \_SB.RMEM._Y12._BAS, BAS1)  // _BAS:
Base Address
                CreateDWordField (CRS, \_SB.RMEM._Y12._LEN, LEN1)  // _LEN:
Length
                CreateDWordField (CRS, \_SB.RMEM._Y13._BAS, BAS2)  // _BAS:
Base Address
                CreateDWordField (CRS, \_SB.RMEM._Y13._LEN, LEN2)  // _LEN:
Length
                CreateDWordField (CRS, \_SB.RMEM._Y14._LEN, LEN3)  // _LEN:
Length
                CreateDWordField (CRS, \_SB.RMEM._Y15._BAS, BAS4)  // _BAS:
Base Address
                CreateDWordField (CRS, \_SB.RMEM._Y15._LEN, LEN4)  // _LEN:
Length
                If (OSFL ()) {}
                ElseIf (MG1B)
                {
                    If (LGreater (MG1B, 0x000C0000))
                    {
                        Store (0x000C0000, BAS1)
                        Subtract (MG1B, BAS1, LEN1)
                    }
                }
                Else
                {
                    Store (0x000C0000, BAS1)
                    Store (0x00020000, LEN1)
                }

                Subtract (MG2B, 0x00100000, LEN3)
```

```
            Add (MG2B, MG2L, BAS4)
            Subtract (Zero, BAS4, LEN4)
            Return (CRS)
        }
    }

    Device (PS2K)
    {
        Name (_HID, EisaId ("PNP0303"))  // _HID: Hardware ID
        Name (_CID, EisaId ("PNP030B"))  // _CID: Compatible ID
        Method (_STA, 0, NotSerialized)  // _STA: Status
        {
            ShiftLeft (One, 0x0A, Local0)
            If (And (IOST, Local0))
            {
                Return (0x0F)
            }

            Return (Zero)
        }

        Name (_CRS, ResourceTemplate ()  // _CRS: Current Resource
Settings
        {
            IO (Decode16,
                0x0060,             // Range Minimum
                0x0060,             // Range Maximum
                0x00,               // Alignment
                0x01,               // Length
                )
            IO (Decode16,
                0x0064,             // Range Minimum
                0x0064,             // Range Maximum
                0x00,               // Alignment
                0x01,               // Length
                )
            IRQNoFlags ()
                {1}
        })
    }

    Method (PS2K._PSW, 1, NotSerialized)  // _PSW: Power State Wake
    {
        If (LNot (LOr (LEqual (OSFL (), One), LEqual (OSFL (), 0x02))))
        {
            Store (Arg0, KBFG)
        }
    }
```

```
            Method (PS2K._PRW, 0, NotSerialized)  // _PRW: Power Resources for
Wake
            {
                Return (GPRW (0x1D, 0x04))
            }

            Device (PS2M)
            {
                Name (_HID, EisaId ("PNP0F03"))  // _HID: Hardware ID
                Name (_CID, EisaId ("PNP0F13"))  // _CID: Compatible ID
                Method (_STA, 0, NotSerialized)  // _STA: Status
                {
                    ShiftLeft (One, 0x0C, Local0)
                    If (And (IOST, Local0))
                    {
                        Return (0x0F)
                    }

                    Return (Zero)
                }

                Name (M2R0, ResourceTemplate ()
                {
                    IRQNoFlags ()
                        {12}
                })
                Name (M2R1, ResourceTemplate ()
                {
                    FixedIO (
                        0x0060,             // Address
                        0x01,               // Length
                        )
                    FixedIO (
                        0x0064,             // Address
                        0x01,               // Length
                        )
                    IRQNoFlags ()
                        {12}
                })
                Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
                {
                    ShiftLeft (One, 0x0A, Local0)
                    If (And (IOST, Local0))
                    {
                        Return (M2R0)
                    }
                    Else
                    {
                        Return (M2R1)
```

```
            }
        }
    }

    Method (PS2M._PSW, 1, NotSerialized)  // _PSW: Power State Wake
    {
        If (LNot (LOr (LEqual (OSFL (), One), LEqual (OSFL (), 0x02))))
        {
            Store (Arg0, MSFG)
        }
    }

    Method (PS2M._PRW, 0, NotSerialized)  // _PRW: Power Resources for
Wake
    {
        Return (GPRW (0x1D, 0x04))
    }
}

Device (IDE0)
{
    Name (_ADR, 0x001F0001)  // _ADR: Address
    Name (^NATA, Package (0x01)
    {
        0x001F0001
    })
    Name (REGF, One)
    Method (_REG, 2, NotSerialized)  // _REG: Region Availability
    {
        If (LEqual (Arg0, 0x02))
        {
            Store (Arg1, REGF)
        }
    }

    Name (TIM0, Package (0x08)
    {
        Package (0x04)
        {
            0x78,
            0xB4,
            0xF0,
            0x0384
        },

        Package (0x04)
        {
            0x23,
            0x21,
```

```
    0x10,
    Zero
},

Package (0x04)
{
    0x0B,
    0x09,
    0x04,
    Zero
},

Package (0x06)
{
    0x78,
    0x50,
    0x3C,
    0x28,
    0x1E,
    0x14
},

Package (0x06)
{
    Zero,
    One,
    0x02,
    One,
    0x02,
    One
},

Package (0x06)
{
    Zero,
    Zero,
    Zero,
    One,
    One,
    One
},

Package (0x04)
{
    0x04,
    0x03,
    0x02,
    Zero
},
```

```
    Package (0x04)
    {
        0x02,
        One,
        Zero,
        Zero
    }
})
Name (TMD0, Buffer (0x14) {})
CreateDWordField (TMD0, Zero, PIO0)
CreateDWordField (TMD0, 0x04, DMA0)
CreateDWordField (TMD0, 0x08, PIO1)
CreateDWordField (TMD0, 0x0C, DMA1)
CreateDWordField (TMD0, 0x10, CHNF)
OperationRegion (CFG2, PCI_Config, 0x40, 0x20)
Field (CFG2, DWordAcc, NoLock, Preserve)
{
    PMPT,   4,
    PSPT,   4,
    PMRI,   6,
    Offset (0x02),
    SMPT,   4,
    SSPT,   4,
    SMRI,   6,
    Offset (0x04),
    PSRI,   4,
    SSRI,   4,
    Offset (0x08),
    PM3E,   1,
    PS3E,   1,
    SM3E,   1,
    SS3E,   1,
    Offset (0x0A),
    PMUT,   2,
        ,   2,
    PSUT,   2,
    Offset (0x0B),
    SMUT,   2,
        ,   2,
    SSUT,   2,
    Offset (0x0C),
    Offset (0x14),
    PM6E,   1,
    PS6E,   1,
    SM6E,   1,
    SS6E,   1,
    PMCR,   1,
    PSCR,   1,
```

```
    SMCR,   1,
    SSCR,   1,
        ,   4,
    PMAE,   1,
    PSAE,   1,
    SMAE,   1,
    SSAE,   1
}

Name (GMPT, Zero)
Name (GMUE, Zero)
Name (GMUT, Zero)
Name (GMCR, Zero)
Name (GSPT, Zero)
Name (GSUE, Zero)
Name (GSUT, Zero)
Name (GSCR, Zero)
Device (CHN0)
{
    Name (_ADR, Zero)  // _ADR: Address
    Method (_GTM, 0, NotSerialized)  // _GTM: Get Timing Mode
    {
        ShiftLeft (PSCR, One, Local1)
        Or (PMCR, Local1, Local0)
        ShiftLeft (PMAE, 0x02, Local3)
        ShiftLeft (PM6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (PM3E, Local3, Local1)
        ShiftLeft (PMPT, 0x04, Local3)
        Or (Local1, Local3, Local1)
        ShiftLeft (PSAE, 0x02, Local3)
        ShiftLeft (PS6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (PS3E, Local3, Local2)
        ShiftLeft (PSPT, 0x04, Local3)
        Or (Local2, Local3, Local2)
        Return (GTM (PMRI, Local1, PMUT, PSRI, Local2, PSUT, Local0))
    }

    Method (_STM, 3, NotSerialized)  // _STM: Set Timing Mode
    {
        Store (Arg0, Debug)
        Store (Arg0, TMD0)
        ShiftLeft (PMAE, 0x02, Local3)
        ShiftLeft (PM6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (PM3E, Local3, Local0)
        ShiftLeft (PMPT, 0x04, Local3)
        Or (Local0, Local3, Local0)
```

```
ShiftLeft (PSAE, 0x02, Local3)
ShiftLeft (PS6E, One, Local4)
Or (Local3, Local4, Local3)
Or (PS3E, Local3, Local1)
ShiftLeft (PSPT, 0x04, Local3)
Or (Local1, Local3, Local1)
Store (PMRI, GMPT)
Store (Local0, GMUE)
Store (PMUT, GMUT)
Store (PMCR, GMCR)
Store (PSRI, GSPT)
Store (Local1, GSUE)
Store (PSUT, GSUT)
Store (PSCR, GSCR)
STM ()
Store (GMPT, PMRI)
Store (GMUE, Local0)
Store (GMUT, PMUT)
Store (GMCR, PMCR)
Store (GSUE, Local1)
Store (GSUT, PSUT)
Store (GSCR, PSCR)
If (And (Local0, One))
{
   Store (One, PM3E)
}
Else
{
   Store (Zero, PM3E)
}

If (And (Local0, 0x02))
{
   Store (One, PM6E)
}
Else
{
   Store (Zero, PM6E)
}

If (And (Local0, 0x04))
{
   Store (One, PMAE)
}
Else
{
   Store (Zero, PMAE)
}
```

```
            If (And (Local1, One))
            {
                Store (One, PS3E)
            }
            Else
            {
                Store (Zero, PS3E)
            }

            If (And (Local1, 0x02))
            {
                Store (One, PS6E)
            }
            Else
            {
                Store (Zero, PS6E)
            }

            If (And (Local1, 0x04))
            {
                Store (One, PSAE)
            }
            Else
            {
                Store (Zero, PSAE)
            }

            Store (GTF (Zero, Arg1), ATA0)
            Store (GTF (One, Arg2), ATA1)
        }

        Device (DRV0)
        {
            Name (_ADR, Zero)  // _ADR: Address
            Method (_GTF, 0, NotSerialized)  // _GTF: Get Task File
            {
                Return (RATA (ATA0))
            }
        }

        Device (DRV1)
        {
            Name (_ADR, One)  // _ADR: Address
            Method (_GTF, 0, NotSerialized)  // _GTF: Get Task File
            {
                Return (RATA (ATA1))
            }
        }
    }
```

```
Device (CHN1)
{
    Name (_ADR, One)  // _ADR: Address
    Method (_GTM, 0, NotSerialized)  // _GTM: Get Timing Mode
    {
        ShiftLeft (SSCR, One, Local1)
        Or (SMCR, Local1, Local0)
        ShiftLeft (SMAE, 0x02, Local3)
        ShiftLeft (SM6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (SM3E, Local3, Local1)
        ShiftLeft (SMPT, 0x04, Local3)
        Or (Local1, Local3, Local1)
        ShiftLeft (SSAE, 0x02, Local3)
        ShiftLeft (SS6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (SS3E, Local3, Local2)
        ShiftLeft (SSPT, 0x04, Local3)
        Or (Local2, Local3, Local2)
        Return (GTM (SMRI, Local1, SMUT, SSRI, Local2, SSUT, Local0))
    }

    Method (_STM, 3, NotSerialized)  // _STM: Set Timing Mode
    {
        Store (Arg0, Debug)
        Store (Arg0, TMD0)
        ShiftLeft (SMAE, 0x02, Local3)
        ShiftLeft (SM6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (SM3E, Local3, Local0)
        ShiftLeft (SMPT, 0x04, Local3)
        Or (Local0, Local3, Local0)
        ShiftLeft (SSAE, 0x02, Local3)
        ShiftLeft (SS6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (SS3E, Local3, Local1)
        ShiftLeft (SSPT, 0x04, Local3)
        Or (Local1, Local3, Local1)
        Store (SMRI, GMPT)
        Store (Local0, GMUE)
        Store (SMUT, GMUT)
        Store (SMCR, GMCR)
        Store (SSRI, GSPT)
        Store (Local1, GSUE)
        Store (SSUT, GSUT)
        Store (SSCR, GSCR)
        STM ()
        Store (GMPT, SMRI)
```

```
Store (GMUE, Local0)
Store (GMUT, SMUT)
Store (GMCR, SMCR)
Store (GSUE, Local1)
Store (GSUT, SSUT)
Store (GSCR, SSCR)
If (And (Local0, One))
{
   Store (One, SM3E)
}
Else
{
   Store (Zero, SM3E)
}

If (And (Local0, 0x02))
{
   Store (One, SM6E)
}
Else
{
   Store (Zero, SM6E)
}

If (And (Local0, 0x04))
{
   Store (One, SMAE)
}
Else
{
   Store (Zero, SMAE)
}

If (And (Local1, One))
{
   Store (One, SS3E)
}
Else
{
   Store (Zero, SS3E)
}

If (And (Local1, 0x02))
{
   Store (One, SS6E)
}
Else
{
   Store (Zero, SS6E)
```

```
        }

        If (And (Local1, 0x04))
        {
            Store (One, SSAE)
        }
        Else
        {
            Store (Zero, SSAE)
        }

        Store (GTF (Zero, Arg1), ATA2)
        Store (GTF (One, Arg2), ATA3)
    }

    Device (DRV0)
    {
        Name (_ADR, Zero)  // _ADR: Address
        Method (_GTF, 0, NotSerialized)  // _GTF: Get Task File
        {
            Return (RATA (ATA2))
        }
    }

    Device (DRV1)
    {
        Name (_ADR, One)  // _ADR: Address
        Method (_GTF, 0, NotSerialized)  // _GTF: Get Task File
        {
            Return (RATA (ATA3))
        }
    }
}

Method (GTM, 7, Serialized)
{
    Store (Ones, PIO0)
    Store (Ones, PIO1)
    Store (Ones, DMA0)
    Store (Ones, DMA1)
    Store (0x10, CHNF)
    If (REGF) {}
    Else
    {
        Return (TMD0)
    }

    If (And (Arg1, 0x20))
    {
```

```
            Or (CHNF, 0x02, CHNF)
        }

        Store (Match (DerefOf (Index (TIM0, One)), MEQ, Arg0, MTR, Zero,
Zero), Local6)
        Store (DerefOf (Index (DerefOf (Index (TIM0, Zero)), Local6)), Local7)
        Store (Local7, DMA0)
        Store (Local7, PIO0)
        If (And (Arg4, 0x20))
        {
            Or (CHNF, 0x08, CHNF)
        }

        Store (Match (DerefOf (Index (TIM0, 0x02)), MEQ, Arg3, MTR, Zero,
Zero), Local6)
        Store (DerefOf (Index (DerefOf (Index (TIM0, Zero)), Local6)), Local7)
        Store (Local7, DMA1)
        Store (Local7, PIO1)
        If (And (Arg1, 0x07))
        {
            Store (Arg2, Local5)
            If (And (Arg1, 0x02))
            {
                Add (Local5, 0x02, Local5)
            }

            If (And (Arg1, 0x04))
            {
                Add (Local5, 0x04, Local5)
            }

            Store (DerefOf (Index (DerefOf (Index (TIM0, 0x03)), Local5)), DMA0)
            Or (CHNF, One, CHNF)
        }

        If (And (Arg4, 0x07))
        {
            Store (Arg5, Local5)
            If (And (Arg4, 0x02))
            {
                Add (Local5, 0x02, Local5)
            }

            If (And (Arg4, 0x04))
            {
                Add (Local5, 0x04, Local5)
            }

            Store (DerefOf (Index (DerefOf (Index (TIM0, 0x03)), Local5)), DMA1)
```

```
            Or (CHNF, 0x04, CHNF)
        }

        Store (TMD0, Debug)
        Return (TMD0)
    }

    Method (STM, 0, Serialized)
    {
        If (REGF)
        {
            Store (Zero, GMUE)
            Store (Zero, GMUT)
            Store (Zero, GSUE)
            Store (Zero, GSUT)
            If (And (CHNF, One))
            {
                Store (Match (DerefOf (Index (TIM0, 0x03)), MLE, DMA0, MTR,
Zero, Zero), Local0)
                If (LGreater (Local0, 0x05))
                {
                    Store (0x05, Local0)
                }

                Store (DerefOf (Index (DerefOf (Index (TIM0, 0x04)), Local0)),
GMUT)
                Or (GMUE, One, GMUE)
                If (LGreater (Local0, 0x02))
                {
                    Or (GMUE, 0x02, GMUE)
                }

                If (LGreater (Local0, 0x04))
                {
                    And (GMUE, 0xFD, GMUE)
                    Or (GMUE, 0x04, GMUE)
                }
            }
            ElseIf (Or (LEqual (PIO0, Ones), LEqual (PIO0, Zero)))
            {
                If (And (LLess (DMA0, Ones), LGreater (DMA0, Zero)))
                {
                    Store (DMA0, PIO0)
                    Or (GMUE, 0x80, GMUE)
                }
            }

            If (And (CHNF, 0x04))
            {
```

```
                    Store (Match (DerefOf (Index (TIM0, 0x03)), MLE, DMA1, MTR,
Zero, Zero), Local0)
                    If (LGreater (Local0, 0x05))
                    {
                        Store (0x05, Local0)
                    }

                    Store (DerefOf (Index (DerefOf (Index (TIM0, 0x04)), Local0)),
GSUT)
                    Or (GSUE, One, GSUE)
                    If (LGreater (Local0, 0x02))
                    {
                        Or (GSUE, 0x02, GSUE)
                    }

                    If (LGreater (Local0, 0x04))
                    {
                        And (GSUE, 0xFD, GSUE)
                        Or (GSUE, 0x04, GSUE)
                    }
                }
                ElseIf (Or (LEqual (PIO1, Ones), LEqual (PIO1, Zero)))
                {
                    If (And (LLess (DMA1, Ones), LGreater (DMA1, Zero)))
                    {
                        Store (DMA1, PIO1)
                        Or (GSUE, 0x80, GSUE)
                    }
                }

                If (And (CHNF, 0x02))
                {
                    Or (GMUE, 0x20, GMUE)
                }

                If (And (CHNF, 0x08))
                {
                    Or (GSUE, 0x20, GSUE)
                }

                And (Match (DerefOf (Index (TIM0, Zero)), MGE, PIO0, MTR, Zero,
Zero), 0x07, Local0)
                Store (DerefOf (Index (DerefOf (Index (TIM0, One)), Local0)), Local1)
                Store (Local1, GMPT)
                If (LLess (Local0, 0x03))
                {
                    Or (GMUE, 0x50, GMUE)
                }
```

```
                And (Match (DerefOf (Index (TIM0, Zero)), MGE, PIO1, MTR, Zero,
Zero), 0x07, Local0)
                Store (DerefOf (Index (DerefOf (Index (TIM0, 0x02)), Local0)),
Local1)
                Store (Local1, GSPT)
                If (LLess (Local0, 0x03))
                {
                    Or (GSUE, 0x50, GSUE)
                }
            }
        }

        Name (AT01, Buffer (0x07)
        {
            0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0xEF
        })
        Name (AT02, Buffer (0x07)
        {
            0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x90
        })
        Name (AT03, Buffer (0x07)
        {
            0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC6
        })
        Name (AT04, Buffer (0x07)
        {
            0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x91
        })
        Name (ATA0, Buffer (0x1D) {})
        Name (ATA1, Buffer (0x1D) {})
        Name (ATA2, Buffer (0x1D) {})
        Name (ATA3, Buffer (0x1D) {})
        Name (ATAB, Buffer (0x1D) {})
        CreateByteField (ATAB, Zero, CMDC)
        Method (GTFB, 3, Serialized)
        {
            Multiply (CMDC, 0x38, Local0)
            Add (Local0, 0x08, Local1)
            CreateField (ATAB, Local1, 0x38, CMDX)
            Multiply (CMDC, 0x07, Local0)
            CreateByteField (ATAB, Add (Local0, 0x02), A001)
            CreateByteField (ATAB, Add (Local0, 0x06), A005)
            Store (Arg0, CMDX)
            Store (Arg1, A001)
            Store (Arg2, A005)
            Increment (CMDC)
        }

        Method (GTF, 2, Serialized)
```

```
{
    Store (Arg1, Debug)
    Store (Zero, CMDC)
    Name (ID49, 0x0C00)
    Name (ID59, Zero)
    Name (ID53, 0x04)
    Name (ID63, 0x0F00)
    Name (ID88, 0x0F00)
    Name (IRDY, One)
    Name (PIOT, Zero)
    Name (DMAT, Zero)
    If (LEqual (SizeOf (Arg1), 0x0200))
    {
        CreateWordField (Arg1, 0x62, IW49)
        Store (IW49, ID49)
        CreateWordField (Arg1, 0x6A, IW53)
        Store (IW53, ID53)
        CreateWordField (Arg1, 0x7E, IW63)
        Store (IW63, ID63)
        CreateWordField (Arg1, 0x76, IW59)
        Store (IW59, ID59)
        CreateWordField (Arg1, 0xB0, IW88)
        Store (IW88, ID88)
    }

    Store (0xA0, Local7)
    If (Arg0)
    {
        Store (0xB0, Local7)
        And (CHNF, 0x08, IRDY)
        If (And (CHNF, 0x10))
        {
            Store (PIO1, PIOT)
        }
        Else
        {
            Store (PIO0, PIOT)
        }

        If (And (CHNF, 0x04))
        {
            If (And (CHNF, 0x10))
            {
                Store (DMA1, DMAT)
            }
            Else
            {
                Store (DMA0, DMAT)
            }
```

```
            }
        }
        Else
        {
            And (CHNF, 0x02, IRDY)
            Store (PIO0, PIOT)
            If (And (CHNF, One))
            {
                Store (DMA0, DMAT)
            }
        }

        If (LAnd (LAnd (And (ID53, 0x04), And (ID88, 0xFF00)), DMAT))
        {
            Store (Match (DerefOf (Index (TIM0, 0x03)), MLE, DMAT, MTR, Zero,
Zero), Local1)
            If (LGreater (Local1, 0x05))
            {
                Store (0x05, Local1)
            }

            GTFB (AT01, Or (0x40, Local1), Local7)
        }
        ElseIf (LAnd (And (ID63, 0xFF00), PIOT))
        {
            And (Match (DerefOf (Index (TIM0, Zero)), MGE, PIOT, MTR, Zero,
Zero), 0x03, Local0)
            Or (0x20, DerefOf (Index (DerefOf (Index (TIM0, 0x07)), Local0)),
Local1)
            GTFB (AT01, Local1, Local7)
        }

        If (IRDY)
        {
            And (Match (DerefOf (Index (TIM0, Zero)), MGE, PIOT, MTR, Zero,
Zero), 0x07, Local0)
            Or (0x08, DerefOf (Index (DerefOf (Index (TIM0, 0x06)), Local0)),
Local1)
            GTFB (AT01, Local1, Local7)
        }
        ElseIf (And (ID49, 0x0400))
        {
            GTFB (AT01, One, Local7)
        }

        If (LAnd (And (ID59, 0x0100), And (ID59, 0xFF)))
        {
            GTFB (AT03, And (ID59, 0xFF), Local7)
        }
```

```
        Store (ATAB, Debug)
        Return (ATAB)
    }

    Method (RATA, 1, NotSerialized)
    {
        CreateByteField (Arg0, Zero, CMDN)
        Multiply (CMDN, 0x38, Local0)
        CreateField (Arg0, 0x08, Local0, RETB)
        Store (RETB, Debug)
        Return (Concatenate (RETB, FZTF))
    }
}

Device (IDE1)
{
    Name (_ADR, 0x001F0002)  // _ADR: Address
    Name (\FZTF, Buffer (0x07)
    {
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF5
    })
    Name (REGF, One)
    Method (_REG, 2, NotSerialized)  // _REG: Region Availability
    {
        If (LEqual (Arg0, 0x02))
        {
            Store (Arg1, REGF)
        }
    }

    Name (TIM0, Package (0x08)
    {
        Package (0x04)
        {
            0x78,
            0xB4,
            0xF0,
            0x0384
        },

        Package (0x04)
        {
            0x23,
            0x21,
            0x10,
            Zero
        },
```

```
Package (0x04)
{
    0x0B,
    0x09,
    0x04,
    Zero
},

Package (0x06)
{
    0x78,
    0x50,
    0x3C,
    0x28,
    0x1E,
    0x14
},

Package (0x06)
{
    Zero,
    One,
    0x02,
    One,
    0x02,
    One
},

Package (0x06)
{
    Zero,
    Zero,
    Zero,
    One,
    One,
    One
},

Package (0x04)
{
    0x04,
    0x03,
    0x02,
    Zero
},

Package (0x04)
{
    0x02,
```

```
            One,
            Zero,
            Zero
        }
    })
    Name (TMD0, Buffer (0x14) {})
    CreateDWordField (TMD0, Zero, PIO0)
    CreateDWordField (TMD0, 0x04, DMA0)
    CreateDWordField (TMD0, 0x08, PIO1)
    CreateDWordField (TMD0, 0x0C, DMA1)
    CreateDWordField (TMD0, 0x10, CHNF)
    OperationRegion (CFG2, PCI_Config, 0x40, 0x20)
    Field (CFG2, DWordAcc, NoLock, Preserve)
    {
        PMPT,   4,
        PSPT,   4,
        PMRI,   6,
        Offset (0x02),
        SMPT,   4,
        SSPT,   4,
        SMRI,   6,
        Offset (0x04),
        PSRI,   4,
        SSRI,   4,
        Offset (0x08),
        PM3E,   1,
        PS3E,   1,
        SM3E,   1,
        SS3E,   1,
        Offset (0x0A),
        PMUT,   2,
            ,   2,
        PSUT,   2,
        Offset (0x0B),
        SMUT,   2,
            ,   2,
        SSUT,   2,
        Offset (0x0C),
        Offset (0x14),
        PM6E,   1,
        PS6E,   1,
        SM6E,   1,
        SS6E,   1,
        PMCR,   1,
        PSCR,   1,
        SMCR,   1,
        SSCR,   1,
            ,   4,
        PMAE,   1,
```

```
    PSAE,   1,
    SMAE,   1,
    SSAE,   1
}

Name (GMPT, Zero)
Name (GMUE, Zero)
Name (GMUT, Zero)
Name (GMCR, Zero)
Name (GSPT, Zero)
Name (GSUE, Zero)
Name (GSUT, Zero)
Name (GSCR, Zero)
Device (CHN0)
{
    Name (_ADR, Zero)  // _ADR: Address
    Method (_GTM, 0, NotSerialized)  // _GTM: Get Timing Mode
    {
        ShiftLeft (PSCR, One, Local1)
        Or (PMCR, Local1, Local0)
        ShiftLeft (PMAE, 0x02, Local3)
        ShiftLeft (PM6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (PM3E, Local3, Local1)
        ShiftLeft (PMPT, 0x04, Local3)
        Or (Local1, Local3, Local1)
        ShiftLeft (PSAE, 0x02, Local3)
        ShiftLeft (PS6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (PS3E, Local3, Local2)
        ShiftLeft (PSPT, 0x04, Local3)
        Or (Local2, Local3, Local2)
        Return (GTM (PMRI, Local1, PMUT, PSRI, Local2, PSUT, Local0))
    }

    Method (_STM, 3, NotSerialized)  // _STM: Set Timing Mode
    {
        Store (Arg0, Debug)
        Store (Arg0, TMD0)
        ShiftLeft (PMAE, 0x02, Local3)
        ShiftLeft (PM6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (PM3E, Local3, Local0)
        ShiftLeft (PMPT, 0x04, Local3)
        Or (Local0, Local3, Local0)
        ShiftLeft (PSAE, 0x02, Local3)
        ShiftLeft (PS6E, One, Local4)
        Or (Local3, Local4, Local3)
        Or (PS3E, Local3, Local1)
```

```
ShiftLeft (PSPT, 0x04, Local3)
Or (Local1, Local3, Local1)
Store (PMRI, GMPT)
Store (Local0, GMUE)
Store (PMUT, GMUT)
Store (PMCR, GMCR)
Store (PSRI, GSPT)
Store (Local1, GSUE)
Store (PSUT, GSUT)
Store (PSCR, GSCR)
STM ()
Store (GMPT, PMRI)
Store (GMUE, Local0)
Store (GMUT, PMUT)
Store (GMCR, PMCR)
Store (GSUE, Local1)
Store (GSUT, PSUT)
Store (GSCR, PSCR)
If (And (Local0, One))
{
   Store (One, PM3E)
}
Else
{
   Store (Zero, PM3E)
}

If (And (Local0, 0x02))
{
   Store (One, PM6E)
}
Else
{
   Store (Zero, PM6E)
}

If (And (Local0, 0x04))
{
   Store (One, PMAE)
}
Else
{
   Store (Zero, PMAE)
}

If (And (Local1, One))
{
   Store (One, PS3E)
}
```

```
            Else
            {
                Store (Zero, PS3E)
            }

            If (And (Local1, 0x02))
            {
                Store (One, PS6E)
            }
            Else
            {
                Store (Zero, PS6E)
            }

            If (And (Local1, 0x04))
            {
                Store (One, PSAE)
            }
            Else
            {
                Store (Zero, PSAE)
            }

            Store (GTF (Zero, Arg1), ATA0)
            Store (GTF (One, Arg2), ATA1)
        }

        Device (DRV0)
        {
            Name (_ADR, Zero)  // _ADR: Address
            Method (_GTF, 0, NotSerialized)  // _GTF: Get Task File
            {
                Return (RATA (ATA0))
            }
        }

        Device (DRV1)
        {
            Name (_ADR, One)  // _ADR: Address
            Method (_GTF, 0, NotSerialized)  // _GTF: Get Task File
            {
                Return (RATA (ATA1))
            }
        }
    }

    Device (CHN1)
    {
        Name (_ADR, One)  // _ADR: Address
```

```
Method (_GTM, 0, NotSerialized)  // _GTM: Get Timing Mode
{
    ShiftLeft (SSCR, One, Local1)
    Or (SMCR, Local1, Local0)
    ShiftLeft (SMAE, 0x02, Local3)
    ShiftLeft (SM6E, One, Local4)
    Or (Local3, Local4, Local3)
    Or (SM3E, Local3, Local1)
    ShiftLeft (SMPT, 0x04, Local3)
    Or (Local1, Local3, Local1)
    ShiftLeft (SSAE, 0x02, Local3)
    ShiftLeft (SS6E, One, Local4)
    Or (Local3, Local4, Local3)
    Or (SS3E, Local3, Local2)
    ShiftLeft (SSPT, 0x04, Local3)
    Or (Local2, Local3, Local2)
    Return (GTM (SMRI, Local1, SMUT, SSRI, Local2, SSUT, Local0))
}

Method (_STM, 3, NotSerialized)  // _STM: Set Timing Mode
{
    Store (Arg0, Debug)
    Store (Arg0, TMD0)
    ShiftLeft (SMAE, 0x02, Local3)
    ShiftLeft (SM6E, One, Local4)
    Or (Local3, Local4, Local3)
    Or (SM3E, Local3, Local0)
    ShiftLeft (SMPT, 0x04, Local3)
    Or (Local0, Local3, Local0)
    ShiftLeft (SSAE, 0x02, Local3)
    ShiftLeft (SS6E, One, Local4)
    Or (Local3, Local4, Local3)
    Or (SS3E, Local3, Local1)
    ShiftLeft (SSPT, 0x04, Local3)
    Or (Local1, Local3, Local1)
    Store (SMRI, GMPT)
    Store (Local0, GMUE)
    Store (SMUT, GMUT)
    Store (SMCR, GMCR)
    Store (SSRI, GSPT)
    Store (Local1, GSUE)
    Store (SSUT, GSUT)
    Store (SSCR, GSCR)
    STM ()
    Store (GMPT, SMRI)
    Store (GMUE, Local0)
    Store (GMUT, SMUT)
    Store (GMCR, SMCR)
    Store (GSUE, Local1)
```

```
Store (GSUT, SSUT)
Store (GSCR, SSCR)
If (And (Local0, One))
{
   Store (One, SM3E)
}
Else
{
   Store (Zero, SM3E)
}

If (And (Local0, 0x02))
{
   Store (One, SM6E)
}
Else
{
   Store (Zero, SM6E)
}

If (And (Local0, 0x04))
{
   Store (One, SMAE)
}
Else
{
   Store (Zero, SMAE)
}

If (And (Local1, One))
{
   Store (One, SS3E)
}
Else
{
   Store (Zero, SS3E)
}

If (And (Local1, 0x02))
{
   Store (One, SS6E)
}
Else
{
   Store (Zero, SS6E)
}

If (And (Local1, 0x04))
{
```

```
            Store (One, SSAE)
        }
        Else
        {
            Store (Zero, SSAE)
        }

        Store (GTF (Zero, Arg1), ATA2)
        Store (GTF (One, Arg2), ATA3)
    }

    Device (DRV0)
    {
        Name (_ADR, Zero)  // _ADR: Address
        Method (_GTF, 0, NotSerialized)  // _GTF: Get Task File
        {
            Return (RATA (ATA2))
        }
    }

    Device (DRV1)
    {
        Name (_ADR, One)  // _ADR: Address
        Method (_GTF, 0, NotSerialized)  // _GTF: Get Task File
        {
            Return (RATA (ATA3))
        }
    }
}

Method (GTM, 7, Serialized)
{
    Store (Ones, PIO0)
    Store (Ones, PIO1)
    Store (Ones, DMA0)
    Store (Ones, DMA1)
    Store (0x10, CHNF)
    If (REGF) {}
    Else
    {
        Return (TMD0)
    }

    If (And (Arg1, 0x20))
    {
        Or (CHNF, 0x02, CHNF)
    }

    Store (Match (DerefOf (Index (TIM0, One)), MEQ, Arg0, MTR, Zero,
```

```
Zero), Local6)
                Store (DerefOf (Index (DerefOf (Index (TIM0, Zero)), Local6)), Local7)
                Store (Local7, DMA0)
                Store (Local7, PIO0)
                If (And (Arg4, 0x20))
                {
                    Or (CHNF, 0x08, CHNF)
                }

                Store (Match (DerefOf (Index (TIM0, 0x02)), MEQ, Arg3, MTR, Zero,
Zero), Local6)
                Store (DerefOf (Index (DerefOf (Index (TIM0, Zero)), Local6)), Local7)
                Store (Local7, DMA1)
                Store (Local7, PIO1)
                If (And (Arg1, 0x07))
                {
                    Store (Arg2, Local5)
                    If (And (Arg1, 0x02))
                    {
                        Add (Local5, 0x02, Local5)
                    }

                    If (And (Arg1, 0x04))
                    {
                        Add (Local5, 0x04, Local5)
                    }

                    Store (DerefOf (Index (DerefOf (Index (TIM0, 0x03)), Local5)), DMA0)
                    Or (CHNF, One, CHNF)
                }

                If (And (Arg4, 0x07))
                {
                    Store (Arg5, Local5)
                    If (And (Arg4, 0x02))
                    {
                        Add (Local5, 0x02, Local5)
                    }

                    If (And (Arg4, 0x04))
                    {
                        Add (Local5, 0x04, Local5)
                    }

                    Store (DerefOf (Index (DerefOf (Index (TIM0, 0x03)), Local5)), DMA1)
                    Or (CHNF, 0x04, CHNF)
                }

                Store (TMD0, Debug)
```

```
            Return (TMD0)
        }

        Method (STM, 0, Serialized)
        {
            If (REGF)
            {
                Store (Zero, GMUE)
                Store (Zero, GMUT)
                Store (Zero, GSUE)
                Store (Zero, GSUT)
                If (And (CHNF, One))
                {
                    Store (Match (DerefOf (Index (TIM0, 0x03)), MLE, DMA0, MTR,
Zero, Zero), Local0)
                    If (LGreater (Local0, 0x05))
                    {
                        Store (0x05, Local0)
                    }

                    Store (DerefOf (Index (DerefOf (Index (TIM0, 0x04)), Local0)),
GMUT)
                    Or (GMUE, One, GMUE)
                    If (LGreater (Local0, 0x02))
                    {
                        Or (GMUE, 0x02, GMUE)
                    }

                    If (LGreater (Local0, 0x04))
                    {
                        And (GMUE, 0xFD, GMUE)
                        Or (GMUE, 0x04, GMUE)
                    }
                }
                ElseIf (Or (LEqual (PIO0, Ones), LEqual (PIO0, Zero)))
                {
                    If (And (LLess (DMA0, Ones), LGreater (DMA0, Zero)))
                    {
                        Store (DMA0, PIO0)
                        Or (GMUE, 0x80, GMUE)
                    }
                }

                If (And (CHNF, 0x04))
                {
                    Store (Match (DerefOf (Index (TIM0, 0x03)), MLE, DMA1, MTR,
Zero, Zero), Local0)
                    If (LGreater (Local0, 0x05))
                    {
```

```
                    Store (0x05, Local0)
                }

                Store (DerefOf (Index (DerefOf (Index (TIM0, 0x04)), Local0)),
GSUT)
                Or (GSUE, One, GSUE)
                If (LGreater (Local0, 0x02))
                {
                    Or (GSUE, 0x02, GSUE)
                }

                If (LGreater (Local0, 0x04))
                {
                    And (GSUE, 0xFD, GSUE)
                    Or (GSUE, 0x04, GSUE)
                }
            }
            ElseIf (Or (LEqual (PIO1, Ones), LEqual (PIO1, Zero)))
            {
                If (And (LLess (DMA1, Ones), LGreater (DMA1, Zero)))
                {
                    Store (DMA1, PIO1)
                    Or (GSUE, 0x80, GSUE)
                }
            }

            If (And (CHNF, 0x02))
            {
                Or (GMUE, 0x20, GMUE)
            }

            If (And (CHNF, 0x08))
            {
                Or (GSUE, 0x20, GSUE)
            }

            And (Match (DerefOf (Index (TIM0, Zero)), MGE, PIO0, MTR, Zero,
Zero), 0x07, Local0)
            Store (DerefOf (Index (DerefOf (Index (TIM0, One)), Local0)), Local1)
            Store (Local1, GMPT)
            If (LLess (Local0, 0x03))
            {
                Or (GMUE, 0x50, GMUE)
            }

            And (Match (DerefOf (Index (TIM0, Zero)), MGE, PIO1, MTR, Zero,
Zero), 0x07, Local0)
            Store (DerefOf (Index (DerefOf (Index (TIM0, 0x02)), Local0)),
Local1)
```

```
        Store (Local1, GSPT)
        If (LLess (Local0, 0x03))
        {
            Or (GSUE, 0x50, GSUE)
        }
    }
}

Name (AT01, Buffer (0x07)
{
    0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0xEF
})
Name (AT02, Buffer (0x07)
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x90
})
Name (AT03, Buffer (0x07)
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC6
})
Name (AT04, Buffer (0x07)
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x91
})
Name (ATA0, Buffer (0x1D) {})
Name (ATA1, Buffer (0x1D) {})
Name (ATA2, Buffer (0x1D) {})
Name (ATA3, Buffer (0x1D) {})
Name (ATAB, Buffer (0x1D) {})
CreateByteField (ATAB, Zero, CMDC)
Method (GTFB, 3, Serialized)
{
    Multiply (CMDC, 0x38, Local0)
    Add (Local0, 0x08, Local1)
    CreateField (ATAB, Local1, 0x38, CMDX)
    Multiply (CMDC, 0x07, Local0)
    CreateByteField (ATAB, Add (Local0, 0x02), A001)
    CreateByteField (ATAB, Add (Local0, 0x06), A005)
    Store (Arg0, CMDX)
    Store (Arg1, A001)
    Store (Arg2, A005)
    Increment (CMDC)
}

Method (GTF, 2, Serialized)
{
    Store (Arg1, Debug)
    Store (Zero, CMDC)
    Name (ID49, 0x0C00)
```

```
Name (ID59, Zero)
Name (ID53, 0x04)
Name (ID63, 0x0F00)
Name (ID88, 0x0F00)
Name (IRDY, One)
Name (PIOT, Zero)
Name (DMAT, Zero)
If (LEqual (SizeOf (Arg1), 0x0200))
{
   CreateWordField (Arg1, 0x62, IW49)
   Store (IW49, ID49)
   CreateWordField (Arg1, 0x6A, IW53)
   Store (IW53, ID53)
   CreateWordField (Arg1, 0x7E, IW63)
   Store (IW63, ID63)
   CreateWordField (Arg1, 0x76, IW59)
   Store (IW59, ID59)
   CreateWordField (Arg1, 0xB0, IW88)
   Store (IW88, ID88)
}

Store (0xA0, Local7)
If (Arg0)
{
   Store (0xB0, Local7)
   And (CHNF, 0x08, IRDY)
   If (And (CHNF, 0x10))
   {
      Store (PIO1, PIOT)
   }
   Else
   {
      Store (PIO0, PIOT)
   }

   If (And (CHNF, 0x04))
   {
      If (And (CHNF, 0x10))
      {
         Store (DMA1, DMAT)
      }
      Else
      {
         Store (DMA0, DMAT)
      }
   }
}
Else
{
```

```
                And (CHNF, 0x02, IRDY)
                Store (PIO0, PIOT)
                If (And (CHNF, One))
                {
                    Store (DMA0, DMAT)
                }
            }

            If (LAnd (LAnd (And (ID53, 0x04), And (ID88, 0xFF00)), DMAT))
            {
                Store (Match (DerefOf (Index (TIM0, 0x03)), MLE, DMAT, MTR, Zero,
Zero), Local1)
                If (LGreater (Local1, 0x05))
                {
                    Store (0x05, Local1)
                }

                GTFB (AT01, Or (0x40, Local1), Local7)
            }
            ElseIf (LAnd (And (ID63, 0xFF00), PIOT))
            {
                And (Match (DerefOf (Index (TIM0, Zero)), MGE, PIOT, MTR, Zero,
Zero), 0x03, Local0)
                Or (0x20, DerefOf (Index (DerefOf (Index (TIM0, 0x07)), Local0)),
Local1)
                GTFB (AT01, Local1, Local7)
            }

            If (IRDY)
            {
                And (Match (DerefOf (Index (TIM0, Zero)), MGE, PIOT, MTR, Zero,
Zero), 0x07, Local0)
                Or (0x08, DerefOf (Index (DerefOf (Index (TIM0, 0x06)), Local0)),
Local1)
                GTFB (AT01, Local1, Local7)
            }
            ElseIf (And (ID49, 0x0400))
            {
                GTFB (AT01, One, Local7)
            }

            If (LAnd (And (ID59, 0x0100), And (ID59, 0xFF)))
            {
                GTFB (AT03, And (ID59, 0xFF), Local7)
            }

            Store (ATAB, Debug)
            Return (ATAB)
        }
```

```
    Method (RATA, 1, NotSerialized)
    {
        CreateByteField (Arg0, Zero, CMDN)
        Multiply (CMDN, 0x38, Local0)
        CreateField (Arg0, 0x08, Local0, RETB)
        Store (RETB, Debug)
        Return (Concatenate (RETB, FZTF))
    }
}

Device (USB0)
{
    Name (_ADR, 0x001D0000)  // _ADR: Address
    OperationRegion (BAR0, PCI_Config, 0xC4, One)
    Field (BAR0, ByteAcc, NoLock, Preserve)
    {
        USBW,   2,
        Offset (0x01)
    }

    Method (_S3D, 0, NotSerialized)  // _S3D: S3 Device State
    {
        If (LOr (LEqual (OSFL (), One), LEqual (OSFL (), 0x02)))
        {
            Return (0x02)
        }
        Else
        {
            Return (0x03)
        }
    }

    Method (_PSW, 1, NotSerialized)  // _PSW: Power State Wake
    {
        If (Arg0)
        {
            Store (0x03, USBW)
        }
        Else
        {
            Store (Zero, USBW)
        }
    }

    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x03, 0x04))
    }
```

```
}

Device (USB1)
{
   Name (_ADR, 0x001D0001)  // _ADR: Address
   OperationRegion (BAR0, PCI_Config, 0xC4, One)
   Field (BAR0, ByteAcc, NoLock, Preserve)
   {
      USBW,   2,
      Offset (0x01)
   }

   Method (_S3D, 0, NotSerialized)  // _S3D: S3 Device State
   {
      If (LOr (LEqual (OSFL (), One), LEqual (OSFL (), 0x02)))
      {
         Return (0x02)
      }
      Else
      {
         Return (0x03)
      }
   }

   Method (_PSW, 1, NotSerialized)  // _PSW: Power State Wake
   {
      If (Arg0)
      {
         Store (0x03, USBW)
      }
      Else
      {
         Store (Zero, USBW)
      }
   }

   Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
   {
      Return (GPRW (0x04, 0x04))
   }
}

Device (USB2)
{
   Name (_ADR, 0x001D0002)  // _ADR: Address
   OperationRegion (BAR0, PCI_Config, 0xC4, One)
   Field (BAR0, ByteAcc, NoLock, Preserve)
   {
      USBW,   2,
```

```
        Offset (0x01)
    }

    Method (_S3D, 0, NotSerialized)  // _S3D: S3 Device State
    {
        If (LOr (LEqual (OSFL (), One), LEqual (OSFL (), 0x02)))
        {
            Return (0x02)
        }
        Else
        {
            Return (0x03)
        }
    }

    Method (_PSW, 1, NotSerialized)  // _PSW: Power State Wake
    {
        If (Arg0)
        {
            Store (0x03, USBW)
        }
        Else
        {
            Store (Zero, USBW)
        }
    }

    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x0C, 0x04))
    }
}

Device (USB3)
{
    Name (_ADR, 0x001D0003)  // _ADR: Address
    OperationRegion (BAR0, PCI_Config, 0xC4, One)
    Field (BAR0, ByteAcc, NoLock, Preserve)
    {
        USBW,   2,
        Offset (0x01)
    }

    Method (_S3D, 0, NotSerialized)  // _S3D: S3 Device State
    {
        If (LOr (LEqual (OSFL (), One), LEqual (OSFL (), 0x02)))
        {
            Return (0x02)
        }
```

```
        Else
        {
            Return (0x03)
        }
    }

    Method (_PSW, 1, NotSerialized)  // _PSW: Power State Wake
    {
        If (Arg0)
        {
            Store (0x03, USBW)
        }
        Else
        {
            Store (Zero, USBW)
        }
    }

    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x0E, 0x04))
    }
}

Device (EUSB)
{
    Name (_ADR, 0x001D0007)  // _ADR: Address
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x0D, 0x04))
    }
}

Device (MC97)
{
    Name (_ADR, 0x001E0003)  // _ADR: Address
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x05, 0x04))
    }
}

Device (P0P4)
{
    Name (_ADR, 0x001C0000)  // _ADR: Address
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x09, 0x04))
    }
```

```
    Method (_PRT, 0, NotSerialized)  // _PRT: PCI Routing Table
    {
        If (PICM)
        {
            Return (AR04)
        }

        Return (PR04)
    }
}

Device (P0P5)
{
    Name (_ADR, 0x001C0001)  // _ADR: Address
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x09, 0x04))
    }

    Method (_PRT, 0, NotSerialized)  // _PRT: PCI Routing Table
    {
        If (PICM)
        {
            Return (AR05)
        }

        Return (PR05)
    }
}

Device (P0P6)
{
    Name (_ADR, 0x001C0002)  // _ADR: Address
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x09, 0x04))
    }
}

Device (P0P7)
{
    Name (_ADR, 0x001C0003)  // _ADR: Address
    Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
    {
        Return (GPRW (0x09, 0x04))
    }
}
```

```
    Device (P0P8)
    {
        Name (_ADR, 0x001C0004)  // _ADR: Address
        Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
        {
            Return (GPRW (0x09, 0x04))
        }
    }

    Device (P0P9)
    {
        Name (_ADR, 0x001C0005)  // _ADR: Address
        Method (_PRW, 0, NotSerialized)  // _PRW: Power Resources for Wake
        {
            Return (GPRW (0x09, 0x04))
        }
    }
}

Scope (\_GPE)
{
    Method (_L09, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
    {
        Notify (\_SB.PCI0.P0P2, 0x02)
        Notify (\_SB.PCI0.P0P3, 0x02)
        Notify (\_SB.PCI0.P0P4, 0x02)
        Notify (\_SB.PCI0.P0P5, 0x02)
        Notify (\_SB.PCI0.P0P6, 0x02)
        Notify (\_SB.PCI0.P0P7, 0x02)
        Notify (\_SB.PCI0.P0P8, 0x02)
        Notify (\_SB.PCI0.P0P9, 0x02)
        Notify (\_SB.PWRB, 0x02)
    }

    Method (_L0B, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
    {
        Notify (\_SB.PCI0.P0P1, 0x02)
        Notify (\_SB.PWRB, 0x02)
    }

    Method (_L08, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
    {
        \_SB.PCI0.SBRG.SIOH ()
    }

    Method (_L1D, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
    {
        \_SB.PCI0.SBRG.SIOH ()
    }
```

```
        Method (_L03, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
        {
            Notify (\_SB.PCI0.USB0, 0x02)
            Notify (\_SB.PWRB, 0x02)
        }

        Method (_L04, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
        {
            Notify (\_SB.PCI0.USB1, 0x02)
            Notify (\_SB.PWRB, 0x02)
        }

        Method (_L0C, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
        {
            Notify (\_SB.PCI0.USB2, 0x02)
            Notify (\_SB.PWRB, 0x02)
        }

        Method (_L0E, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
        {
            Notify (\_SB.PCI0.USB3, 0x02)
            Notify (\_SB.PWRB, 0x02)
        }

        Method (_L0D, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
        {
            Notify (\_SB.PCI0.EUSB, 0x02)
            Notify (\_SB.PWRB, 0x02)
        }

        Method (_L05, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
        {
            Notify (\_SB.PCI0.MC97, 0x02)
            Notify (\_SB.PWRB, 0x02)
        }
    }

    Device (PWRB)
    {
        Name (_HID, EisaId ("PNP0C0C"))  // _HID: Hardware ID
        Name (_UID, 0xAA)  // _UID: Unique ID
        Name (_STA, 0x0B)  // _STA: Status
    }
}

Scope (_SB.PCI0.SBRG.ASOC)
{
    Name (G0T3, Package (0x07)
```

```
    {
        0x00070005,
        "New CPU Installed",
        One,
        Zero,
        Zero,
        One,
        0x02
    })
    Name (GRP0, Package (0x01)
    {
        G0T3
    })
    Method (GIT0, 1, NotSerialized)
    {
        Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
        Store (And (Arg0, 0xFFFF), _T_0)
        If (LEqual (_T_0, 0x05))
        {
            Store (GNVS (0x15B2), ASB1)
        }
        Else
        {
            Store (Zero, ASB0)
        }
    }

    Method (SIT0, 3, NotSerialized)
    {
        If (And (Arg2, 0xFFFF))
        {
            Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
            Store (And (Arg0, 0xFFFF), _T_0)
            If (LEqual (_T_0, 0x05))
            {
                SNVS (0x15B2, Arg1)
                Store (0x03, ASB0)
            }
            Else
            {
                Store (Zero, ASB0)
            }
        }
        Else
        {
            Store (0x03, ASB0)
        }
    }
}
```

```
Scope (_GPE)
{
    Method (_L1A, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
    {
        Notify (\_SB.PCI0.SBRG.ASOC, One)
        Sleep (0x03E8)
    }
}

Scope (_SB.PCI0.SBRG.ASOC)
{
    Name (G3T0, Package (0x07)
    {
        0x03010011,
        "CPU Frequency",
        Zero,
        Zero,
        0x4E20,
        0x64,
        0x0259
    })
    Name (G3DS, Package (0x07)
    {
        0x030600F0,
        "Device Select",
        0x40000000,
        Zero,
        Zero,
        Zero,
        Zero
    })
    Name (GRP3, Package (0x02)
    {
        G3T0,
        G3DS
    })
    Name (GODS, Zero)
    Name (IDEX, Zero)
    Name (FSBB, 0xFFFF)
    Name (PEBD, 0xFFFF)
    Name (FSBS, 0xFFFF)
    Name (MN60, 0xFF)
    Name (CPUM, 0xFF)
    Name (PCEM, 0xFF)
    Name (RT02, 0xFF)
    Name (RT04, 0xFF)
    Name (CLKI, Zero)
    Name (OCPL, 0xFFFF)
```

```
Name (PEC3, 0x66)
Name (PEC5, 0x67)
Name (PED5, 0x67)
Method (GIT3, 1, NotSerialized)
{
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (And (Arg0, 0xFFFF), _T_0)
    If (LEqual (_T_0, 0x11))
    {
        If (LEqual (FSBB, 0xFFFF))
        {
            If (LNotEqual (GNVS (0x34A0), One))
            {
                If (LEqual (GNVS (0x34A0), Zero))
                {
                    Subtract (GNVS (0xA2C0), 0xC8, ASB1)
                }
                Else
                {
                    Subtract (GNVS (0xA760), 0xC8, ASB1)
                }
            }
            Else
            {
                Store (DerefOf (Index (G3T0, 0x03)), ASB1)
            }
        }
        Else
        {
            Store (FSBB, ASB1)
        }
    }
    ElseIf (LEqual (_T_0, 0xF0)) {}
    Else
    {
        Store (Zero, ASB0)
    }
}

Method (SIT3, 3, NotSerialized)
{
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (And (Arg0, 0xFFFF), _T_0)
    If (LEqual (_T_0, 0x11))
    {
        Store (Arg1, FSBB)
        Add (Arg1, 0xC8, Local2)
        If (LEqual (GNVS (0x251E), One))
        {
```

```
Store (One, CLKI)
Store (RBLK (0xD2, Zero, 0x0B), RTLR)
CreateByteField (RTLR, 0x02, BR02)
CreateByteField (RTLR, 0x03, BR03)
CreateByteField (RTLR, 0x04, BR04)
CreateByteField (RTLR, 0x08, BR08)
Multiply (Arg1, 0x02, Local0)
CreateByteField (RTLB, Local0, MN02)
Increment (Local0)
CreateByteField (RTLB, Local0, MN04)
If (LEqual (GNVS (0x34A0), One))
{
   If (LNotEqual (FSBB, DerefOf (Index (G3T0, 0x03))))
   {
      And (BR08, 0x7F, BR08)
   }

   WBLK (0xD2, Zero, 0x0B, RTLR)
}

If (LEqual (OCPL, 0x05))
{
   Multiply (Add (0xC8, DerefOf (Index (G3T0, 0x03))), 0x67, Local1)
   Divide (Local1, 0x64, Local5, Local1)
   Subtract (Local1, 0xC8, Local1)
   Multiply (Local1, 0x02, Local1)
   CreateByteField (RTLB, Local1, RT02)
   Increment (Local1)
   CreateByteField (RTLB, Local1, RT04)
   If (LEqual (Arg1, 0x95))
   {
      OVPE (PEC5)
   }

   If (LEqual (Arg1, 0xDC))
   {
      OVPE (PED5)
   }

   OCMN (RT02, RT04)
   OCMN (MN02, MN04)
   If (LEqual (Arg1, 0x85))
   {
      RVPE ()
   }

   If (LEqual (Arg1, 0xC8))
   {
      RVPE ()
```

```
            }
        }
        ElseIf (LEqual (OCPL, 0x03))
        {
            If (LEqual (Arg1, 0x8E))
            {
                OVPE (PEC3)
            }

            OCMN (MN02, MN04)
            If (LEqual (Arg1, 0x85))
            {
                RVPE ()
            }
        }
        Else
        {
            OCMN (MN02, MN04)
        }

        If (LEqual (GNVS (0x34A0), One))
        {
            If (LEqual (FSBB, DerefOf (Index (G3T0, 0x03))))
            {
                Or (BR08, 0x80, BR08)
            }

            WBLK (0xD2, Zero, 0x0B, RTLR)
        }
    }
    Else
    {
        Store (Zero, CLKI)
        Store (RBLK (0xD2, Zero, 0x09), CLKR)
        CreateByteField (CLKR, One, CB01)
        CreateByteField (CLKR, 0x02, CBM)
        CreateByteField (CLKR, 0x03, CBN)
        CreateWordField (CLKR, 0x02, CBMN)
        CreateByteField (CLKR, 0x04, CB04)
        CreateByteField (CLKR, 0x05, CB05)
        CreateByteField (CLKR, 0x08, CB08)
        Multiply (Arg1, 0x03, Local0)
        CreateWordField (MNBF, Local0, MNVL)
        Increment (Local0)
        CreateByteField (MNBF, Increment (Local0), NVL0)
        And (NVL0, One, NVL0)
        ShiftLeft (NVL0, 0x07, MN60)
        If (LEqual (GNVS (0x34A0), One))
        {
```

```
    If (LNotEqual (FSBB, DerefOf (Index (G3T0, 0x03))))
    {
        And (CB08, 0x7F, CB08)
    }

    WBLK (0xD2, Zero, 0x09, CLKR)
}

If (LEqual (OCPL, 0x05))
{
    Multiply (Add (0xC8, DerefOf (Index (G3T0, 0x03))), 0x67, Local1)
    Divide (Local1, 0x64, Local5, Local1)
    Subtract (Local1, 0xC8, Local1)
    Multiply (Local1, 0x03, Local1)
    CreateByteField (MNBF, Local1, P3I2)
    Increment (Local1)
    CreateByteField (MNBF, Local1, P3I3)
    If (LEqual (Arg1, 0x95))
    {
        OVPE (PEC5)
    }

    If (LEqual (Arg1, 0xDC))
    {
        OVPE (PED5)
    }

    OCN0 ()
    OCMN (P3I2, P3I3)
    Store (MNVL, CBMN)
    OCMN (CBM, CBN)
    If (LEqual (Arg1, 0x85))
    {
        RVPE ()
    }

    If (LEqual (Arg1, 0xC8))
    {
        RVPE ()
    }
}
ElseIf (LEqual (OCPL, 0x03))
{
    If (LEqual (Arg1, 0x8E))
    {
        OVPE (PEC3)
    }

    OCN0 ()
```

```
                Store (MNVL, CBMN)
                OCMN (CBM, CBN)
                If (LEqual (Arg1, 0x85))
                {
                    RVPE ()
                }
            }
            Else
            {
                OCN0 ()
                Store (MNVL, CBMN)
                OCMN (CBM, CBN)
            }

            If (LEqual (GNVS (0x34A0), One))
            {
                If (LEqual (FSBB, DerefOf (Index (G3T0, 0x03))))
                {
                    Or (CB08, 0x80, CB08)
                }

                WBLK (0xD2, Zero, 0x09, CLKR)
            }
        }
    }
    ElseIf (LEqual (_T_0, 0xF0))
    {
        Store (Arg1, GODS)
    }
    Else
    {
        Store (Zero, ASB0)
    }
}

Method (GWS3, 1, NotSerialized)
{
    If (LEqual (Arg0, 0x04))
    {
        Store (0xFFFF, FSBB)
    }
}

Method (OVPE, 1, NotSerialized)
{
    Subtract (Arg0, 0x64, Local0)
    Multiply (Local0, 0x02, Local1)
    If (LEqual (CLKI, Zero))
    {
```

```
            Store (RBLK (0xD2, Zero, 0x09), CLKE)
            CreateByteField (CLKE, One, PCE1)
            CreateByteField (CLKE, 0x04, PCE4)
            CreateByteField (CLKE, 0x05, PCE5)
            Store (PCE4, Index (PEBB, Zero))
            Store (PCE5, Index (PEBB, One))
            Store (DerefOf (Index (PCEB, Local1)), PCE4)
            Store (DerefOf (Index (PCEB, Add (Local1, One, Local1))), PCE5)
            Or (PCE1, 0x40, PCE1)
            WBLK (0xD2, Zero, 0x09, CLKE)
        }
        ElseIf (LEqual (CLKI, One))
        {
            Store (RBLK (0xD2, Zero, 0x0B), RTLE)
            Store (DerefOf (Index (RTLE, 0x03)), Index (PEBB, Zero))
            Store (DerefOf (Index (RTLE, 0x04)), Index (PEBB, One))
            Store (DerefOf (Index (PERT, Local1)), Index (RTLE, 0x03))
            Increment (Local1)
            And (DerefOf (Index (RTLE, 0x04)), 0xF0, Local2)
            Or (Local2, DerefOf (Index (PERT, Local1)), Index (RTLE, 0x04))
            Or (DerefOf (Index (RTLE, 0x0A)), 0x04, Index (RTLE, 0x0A))
            WBLK (0xD2, Zero, 0x0B, RTLE)
        }
    }

    Method (RVPE, 0, NotSerialized)
    {
        If (LEqual (CLKI, Zero))
        {
            Store (RBLK (0xD2, Zero, 0x09), CLKR)
            Store (DerefOf (Index (PEBB, Zero)), Index (CLKR, 0x04))
            Store (DerefOf (Index (PEBB, One)), Index (CLKR, 0x05))
            WBLK (0xD2, Zero, 0x09, CLKR)
        }
        ElseIf (LEqual (CLKI, One))
        {
            Store (RBLK (0xD2, Zero, 0x0B), RTLR)
            Store (DerefOf (Index (PEBB, Zero)), Index (RTLR, 0x03))
            Or (And (DerefOf (Index (RTLR, 0x04)), 0xF0), DerefOf (Index (PEBB,
One)), Index (RTLR, 0x04))
            Or (DerefOf (Index (RTLE, 0x0A)), 0x04, Index (RTLE, 0x0A))
            WBLK (0xD2, Zero, 0x0B, RTLR)
        }
    }

    Method (OCN0, 0, NotSerialized)
    {
        Store (RBLK (0xD2, 0x3C, 0x09), CLKB)
        CreateByteField (CLKB, Zero, CB60)
```

```
    And (CB60, 0x7F, CB60)
    Or (CB60, MN60, CB60)
    WBLK (0xD2, 0x3C, 0x09, CLKB)
}

Method (OCMN, 2, NotSerialized)
{
    If (LEqual (CLKI, Zero))
    {
        Store (RBLK (0xD2, Zero, 0x09), CLKR)
        Store (Arg0, Index (CLKR, 0x02))
        Store (Arg1, Index (CLKR, 0x03))
        Or (DerefOf (Index (CLKR, One)), 0x80, Index (CLKR, One))
        WBLK (0xD2, Zero, 0x09, CLKR)
    }
    ElseIf (LEqual (CLKI, One))
    {
        Store (RBLK (0xD2, Zero, 0x0B), RTLR)
        Store (Arg0, Index (RTLR, 0x02))
        Or (And (DerefOf (Index (RTLR, 0x04)), 0x0F), Arg1, Index (RTLR, 0x04))
        Or (DerefOf (Index (RTLE, 0x0A)), One, Index (RTLE, 0x0A))
        WBLK (0xD2, Zero, 0x0B, RTLR)
    }
}

Method (FSBI, 0, NotSerialized)
{
    If (LGreater (And (GCAX (One), 0x0FF0, Local0), 0x06F0))
    {
        And (ShiftRight (GMAX (0x2C), 0x10), 0x07, Local0)
    }
    Else
    {
        And (GMAX (0xCD), 0x07, Local0)
    }

    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (Local0, _T_0)
    If (LEqual (_T_0, Zero))
    {
        Store (0x42, Index (G3T0, 0x03))
    }
    ElseIf (LEqual (_T_0, 0x02))
    {
        Store (0xC9, Index (G3T0, 0x06))
        Store (Zero, Index (G3T0, 0x03))
    }
    ElseIf (LEqual (_T_0, 0x04))
    {
```

```
        Store (0x85, Index (G3T0, 0x03))
    }
    ElseIf (LEqual (_T_0, 0x06))
    {
        Store (0xC8, Index (G3T0, 0x03))
    }
}

Name (CLKR, Buffer (0x0A)
{
    /* 0000 */  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0008 */  0x00, 0x00
})
Name (CLKB, Buffer (0x0A)
{
    /* 0000 */  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0008 */  0x00, 0x00
})
Name (CLKE, Buffer (0x0A)
{
    /* 0000 */  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0008 */  0x00, 0x00
})
Name (PEBB, Buffer (0x02)
{
     0x00, 0x00
})
Name (RTLR, Buffer (0x0C)
{
    /* 0000 */  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0008 */  0x00, 0x00, 0x00, 0x00
})
Name (RTLE, Buffer (0x0C)
{
    /* 0000 */  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    /* 0008 */  0x00, 0x00, 0x00, 0x00
})
Name (PCEB, Buffer (0x0A)
{
    /* 0000 */  0xC9, 0x2E, 0x8A, 0x34, 0x0A, 0x35, 0x8A, 0x35,
    /* 0008 */  0x0A, 0x36
})
Name (PERT, Buffer (0x0A)
{
    /* 0000 */  0x50, 0x00, 0x50, 0x08, 0x51, 0x06, 0x52, 0x04,
    /* 0008 */  0x53, 0x02
})
Name (MNBF, Buffer (0x04B3)
{
```

```
/* 0000 */  0xC9, 0x12, 0x00, 0xC8, 0x10, 0x00, 0x86, 0x0C,
/* 0008 */  0x01, 0x0A, 0x15, 0x01, 0x4A, 0x15, 0x00, 0x4A,
/* 0010 */  0x15, 0x01, 0xC6, 0x0C, 0x01, 0x48, 0x11, 0x00,
/* 0018 */  0x89, 0x13, 0x00, 0x47, 0x0F, 0x00, 0xCA, 0x15,
/* 0020 */  0x01, 0x47, 0x0F, 0x01, 0xC9, 0x13, 0x01, 0xC8,
/* 0028 */  0x11, 0x00, 0x46, 0x0D, 0x01, 0x4A, 0x16, 0x01,
/* 0030 */  0x8A, 0x16, 0x00, 0x8A, 0x16, 0x01, 0x86, 0x0D,
/* 0038 */  0x01, 0x48, 0x12, 0x00, 0x89, 0x14, 0x01, 0x07,
/* 0040 */  0x10, 0x01, 0x0A, 0x17, 0x01, 0x47, 0x10, 0x00,
/* 0048 */  0x09, 0x15, 0x00, 0xC8, 0x12, 0x00, 0x06, 0x0E,
/* 0050 */  0x01, 0x8A, 0x17, 0x01, 0xCA, 0x17, 0x00, 0xCA,
/* 0058 */  0x17, 0x01, 0x46, 0x0E, 0x01, 0x48, 0x13, 0x00,
/* 0060 */  0xC9, 0x15, 0x00, 0x07, 0x11, 0x00, 0x4A, 0x18,
/* 0068 */  0x01, 0x07, 0x11, 0x01, 0x09, 0x16, 0x01, 0xC8,
/* 0070 */  0x13, 0x00, 0xC6, 0x0E, 0x01, 0xCA, 0x18, 0x01,
/* 0078 */  0x0A, 0x19, 0x00, 0x0A, 0x19, 0x01, 0x06, 0x0F,
/* 0080 */  0x01, 0x48, 0x14, 0x00, 0xC9, 0x16, 0x01, 0xC7,
/* 0088 */  0x11, 0x01, 0x8A, 0x19, 0x01, 0x07, 0x12, 0x00,
/* 0090 */  0x49, 0x17, 0x00, 0xC8, 0x14, 0x00, 0x86, 0x0F,
/* 0098 */  0x01, 0x0A, 0x1A, 0x01, 0x4A, 0x1A, 0x00, 0x4A,
/* 00A0 */  0x1A, 0x01, 0xC6, 0x0F, 0x01, 0x48, 0x15, 0x00,
/* 00A8 */  0x09, 0x18, 0x00, 0xC7, 0x12, 0x00, 0xCA, 0x1A,
/* 00B0 */  0x01, 0xC7, 0x12, 0x01, 0x49, 0x18, 0x01, 0xC8,
/* 00B8 */  0x15, 0x00, 0x46, 0x10, 0x01, 0x4A, 0x1B, 0x01,
/* 00C0 */  0x8A, 0x1B, 0x00, 0x8A, 0x1B, 0x01, 0x89, 0x0C,
/* 00C8 */  0x00, 0x08, 0x0B, 0x01, 0x46, 0x08, 0x01, 0x0A,
/* 00D0 */  0x0E, 0x00, 0x48, 0x0B, 0x00, 0xC7, 0x09, 0x01,
/* 00D8 */  0xC9, 0x0C, 0x00, 0x48, 0x0B, 0x01, 0x07, 0x0A,
/* 00E0 */  0x00, 0xC9, 0x0C, 0x01, 0x4A, 0x0E, 0x01, 0x09,
/* 00E8 */  0x0D, 0x00, 0x07, 0x0A, 0x01, 0x88, 0x0B, 0x01,
/* 00F0 */  0x09, 0x0D, 0x01, 0x47, 0x0A, 0x00, 0xC8, 0x0B,
/* 00F8 */  0x00, 0xCA, 0x0E, 0x00, 0xC6, 0x08, 0x01, 0xC8,
/* 0100 */  0x0B, 0x01, 0x49, 0x0D, 0x01, 0xCA, 0x0E, 0x01,
/* 0108 */  0x0A, 0x0F, 0x00, 0x0A, 0x0F, 0x01, 0x89, 0x0D,
/* 0110 */  0x01, 0x08, 0x0C, 0x01, 0x06, 0x09, 0x01, 0x4A,
/* 0118 */  0x0F, 0x00, 0x48, 0x0C, 0x00, 0xC7, 0x0A, 0x00,
/* 0120 */  0xC9, 0x0D, 0x01, 0x48, 0x0C, 0x01, 0xC7, 0x0A,
/* 0128 */  0x01, 0x09, 0x0E, 0x00, 0x8A, 0x0F, 0x01, 0x09,
/* 0130 */  0x0E, 0x01, 0x07, 0x0B, 0x00, 0x88, 0x0C, 0x01,
/* 0138 */  0x49, 0x0E, 0x00, 0x07, 0x0B, 0x01, 0xC8, 0x0C,
/* 0140 */  0x00, 0x0A, 0x10, 0x00, 0x86, 0x09, 0x01, 0xC8,
/* 0148 */  0x0C, 0x01, 0x89, 0x0E, 0x00, 0x0A, 0x10, 0x01,
/* 0150 */  0x4A, 0x10, 0x00, 0x4A, 0x10, 0x01, 0xC9, 0x0E,
/* 0158 */  0x00, 0x08, 0x0D, 0x01, 0xC6, 0x09, 0x01, 0x8A,
/* 0160 */  0x10, 0x00, 0x48, 0x0D, 0x00, 0x87, 0x0B, 0x01,
/* 0168 */  0x09, 0x0F, 0x00, 0x48, 0x0D, 0x01, 0xC7, 0x0B,
/* 0170 */  0x00, 0x09, 0x0F, 0x01, 0xCA, 0x10, 0x01, 0x49,
/* 0178 */  0x0F, 0x00, 0xC7, 0x0B, 0x01, 0x88, 0x0D, 0x01,
/* 0180 */  0x49, 0x0F, 0x01, 0x07, 0x0C, 0x00, 0xC8, 0x0D,
```

```
/* 0188 */  0x00, 0x4A, 0x11, 0x00, 0x46, 0x0A, 0x01, 0x89,
/* 0190 */  0x0F, 0x01, 0x89, 0x0F, 0x01, 0x4A, 0x11, 0x01,
/* 0198 */  0x8A, 0x11, 0x00, 0x8A, 0x11, 0x01, 0xC9, 0x0F,
/* 01A0 */  0x01, 0x08, 0x0E, 0x01, 0x86, 0x0A, 0x01, 0xCA,
/* 01A8 */  0x11, 0x00, 0xCA, 0x11, 0x01, 0x87, 0x0C, 0x00,
/* 01B0 */  0x09, 0x10, 0x01, 0x48, 0x0E, 0x01, 0x87, 0x0C,
/* 01B8 */  0x01, 0x49, 0x10, 0x00, 0x0A, 0x12, 0x01, 0x49,
/* 01C0 */  0x10, 0x01, 0xC7, 0x0C, 0x00, 0x88, 0x0E, 0x01,
/* 01C8 */  0x89, 0x10, 0x00, 0xC7, 0x0C, 0x01, 0xC8, 0x0E,
/* 01D0 */  0x00, 0x8A, 0x12, 0x00, 0x06, 0x0B, 0x01, 0xC8,
/* 01D8 */  0x0E, 0x01, 0xC9, 0x10, 0x00, 0x8A, 0x12, 0x01,
/* 01E0 */  0xCA, 0x12, 0x00, 0xCA, 0x12, 0x01, 0x09, 0x11,
/* 01E8 */  0x00, 0x08, 0x0F, 0x01, 0x46, 0x0B, 0x01, 0x0A,
/* 01F0 */  0x13, 0x00, 0x48, 0x0F, 0x00, 0x47, 0x0D, 0x01,
/* 01F8 */  0x49, 0x11, 0x00, 0x48, 0x0F, 0x01, 0x87, 0x0D,
/* 0200 */  0x00, 0x49, 0x11, 0x01, 0x4A, 0x13, 0x01, 0x89,
/* 0208 */  0x11, 0x00, 0x87, 0x0D, 0x01, 0x88, 0x0F, 0x01,
/* 0210 */  0x89, 0x11, 0x01, 0xC7, 0x0D, 0x00, 0xC8, 0x0F,
/* 0218 */  0x00, 0xCA, 0x13, 0x00, 0xC6, 0x0B, 0x01, 0xC8,
/* 0220 */  0x0F, 0x01, 0xC9, 0x11, 0x01, 0xCA, 0x13, 0x01,
/* 0228 */  0x0A, 0x14, 0x00, 0x0A, 0x14, 0x01, 0x09, 0x12,
/* 0230 */  0x01, 0x08, 0x10, 0x01, 0x06, 0x0C, 0x01, 0x4A,
/* 0238 */  0x14, 0x00, 0x48, 0x10, 0x00, 0x47, 0x0E, 0x00,
/* 0240 */  0x49, 0x12, 0x01, 0x48, 0x10, 0x01, 0x47, 0x0E,
/* 0248 */  0x01, 0x89, 0x12, 0x00, 0x8A, 0x14, 0x01, 0x89,
/* 0250 */  0x12, 0x01, 0x87, 0x0E, 0x00, 0x88, 0x10, 0x01,
/* 0258 */  0xC9, 0x12, 0x00, 0x87, 0x0E, 0x01, 0xC8, 0x10,
/* 0260 */  0x00, 0x0A, 0x15, 0x00, 0x86, 0x0C, 0x01, 0xC8,
/* 0268 */  0x10, 0x01, 0x09, 0x13, 0x00, 0x0A, 0x15, 0x01,
/* 0270 */  0x4A, 0x15, 0x00, 0x4A, 0x15, 0x01, 0x49, 0x13,
/* 0278 */  0x00, 0x08, 0x11, 0x01, 0xC6, 0x0C, 0x01, 0x8A,
/* 0280 */  0x15, 0x00, 0x48, 0x11, 0x00, 0x07, 0x0F, 0x01,
/* 0288 */  0x89, 0x13, 0x00, 0x48, 0x11, 0x01, 0x47, 0x0F,
/* 0290 */  0x00, 0x89, 0x13, 0x01, 0xCA, 0x15, 0x01, 0xC9,
/* 0298 */  0x13, 0x00, 0x47, 0x0F, 0x01, 0x88, 0x11, 0x01,
/* 02A0 */  0xC9, 0x13, 0x01, 0x87, 0x0F, 0x00, 0xC8, 0x11,
/* 02A8 */  0x00, 0x4A, 0x16, 0x00, 0x46, 0x0D, 0x01, 0xC8,
/* 02B0 */  0x11, 0x01, 0x09, 0x14, 0x01, 0x4A, 0x16, 0x01,
/* 02B8 */  0x8A, 0x16, 0x00, 0x8A, 0x16, 0x01, 0x49, 0x14,
/* 02C0 */  0x01, 0x08, 0x12, 0x01, 0x86, 0x0D, 0x01, 0xCA,
/* 02C8 */  0x16, 0x00, 0x48, 0x12, 0x00, 0x07, 0x10, 0x00,
/* 02D0 */  0x89, 0x14, 0x01, 0x48, 0x12, 0x01, 0x07, 0x10,
/* 02D8 */  0x01, 0xC9, 0x14, 0x00, 0x0A, 0x17, 0x01, 0xC9,
/* 02E0 */  0x14, 0x01, 0x47, 0x10, 0x00, 0x88, 0x12, 0x01,
/* 02E8 */  0x09, 0x15, 0x00, 0x47, 0x10, 0x01, 0xC8, 0x12,
/* 02F0 */  0x00, 0x8A, 0x17, 0x00, 0x06, 0x0E, 0x01, 0xC8,
/* 02F8 */  0x12, 0x01, 0x49, 0x15, 0x00, 0x8A, 0x17, 0x01,
/* 0300 */  0xCA, 0x17, 0x00, 0xCA, 0x17, 0x01, 0x89, 0x15,
/* 0308 */  0x00, 0x08, 0x13, 0x01, 0x46, 0x0E, 0x01, 0x0A,
```

```
/* 0310 */  0x18, 0x00, 0x48, 0x13, 0x00, 0xC7, 0x10, 0x01,
/* 0318 */  0xC9, 0x15, 0x00, 0x48, 0x13, 0x01, 0x07, 0x11,
/* 0320 */  0x00, 0xC9, 0x15, 0x01, 0x4A, 0x18, 0x01, 0x09,
/* 0328 */  0x16, 0x00, 0x07, 0x11, 0x01, 0x88, 0x13, 0x01,
/* 0330 */  0x09, 0x16, 0x01, 0x47, 0x11, 0x00, 0xC8, 0x13,
/* 0338 */  0x00, 0xCA, 0x18, 0x00, 0xC6, 0x0E, 0x01, 0xC8,
/* 0340 */  0x13, 0x01, 0x49, 0x16, 0x01, 0xCA, 0x18, 0x01,
/* 0348 */  0x0A, 0x19, 0x00, 0x0A, 0x19, 0x01, 0x89, 0x16,
/* 0350 */  0x01, 0x08, 0x14, 0x01, 0x06, 0x0F, 0x01, 0x4A,
/* 0358 */  0x19, 0x00, 0x48, 0x14, 0x00, 0xC7, 0x11, 0x00,
/* 0360 */  0xC9, 0x16, 0x01, 0x48, 0x14, 0x01, 0xC7, 0x11,
/* 0368 */  0x01, 0x09, 0x17, 0x00, 0x8A, 0x19, 0x01, 0x09,
/* 0370 */  0x17, 0x01, 0x07, 0x12, 0x00, 0x88, 0x14, 0x01,
/* 0378 */  0x49, 0x17, 0x00, 0x07, 0x12, 0x01, 0xC8, 0x14,
/* 0380 */  0x00, 0x0A, 0x1A, 0x00, 0x86, 0x0F, 0x01, 0xC8,
/* 0388 */  0x14, 0x01, 0x89, 0x17, 0x00, 0x0A, 0x1A, 0x01,
/* 0390 */  0x4A, 0x1A, 0x00, 0x4A, 0x1A, 0x01, 0xC9, 0x17,
/* 0398 */  0x00, 0x08, 0x15, 0x01, 0xC6, 0x0F, 0x01, 0x8A,
/* 03A0 */  0x1A, 0x00, 0x48, 0x15, 0x00, 0x87, 0x12, 0x01,
/* 03A8 */  0x09, 0x18, 0x00, 0x48, 0x15, 0x01, 0xC7, 0x12,
/* 03B0 */  0x00, 0x09, 0x18, 0x01, 0xCA, 0x1A, 0x01, 0x49,
/* 03B8 */  0x18, 0x00, 0xC7, 0x12, 0x01, 0x88, 0x15, 0x01,
/* 03C0 */  0x49, 0x18, 0x01, 0x07, 0x13, 0x00, 0xC8, 0x15,
/* 03C8 */  0x00, 0x4A, 0x1B, 0x00, 0x46, 0x10, 0x01, 0xC8,
/* 03D0 */  0x15, 0x01, 0x89, 0x18, 0x01, 0x4A, 0x1B, 0x01,
/* 03D8 */  0x8A, 0x1B, 0x00, 0x8A, 0x1B, 0x01, 0xC9, 0x18,
/* 03E0 */  0x01, 0x08, 0x16, 0x01, 0x86, 0x10, 0x01, 0xCA,
/* 03E8 */  0x1B, 0x00, 0x48, 0x16, 0x00, 0x87, 0x13, 0x00,
/* 03F0 */  0x09, 0x19, 0x01, 0x48, 0x16, 0x01, 0x87, 0x13,
/* 03F8 */  0x01, 0x49, 0x19, 0x00, 0x0A, 0x1C, 0x01, 0x49,
/* 0400 */  0x19, 0x01, 0xC7, 0x13, 0x00, 0x88, 0x16, 0x01,
/* 0408 */  0x89, 0x19, 0x00, 0xC7, 0x13, 0x01, 0xC8, 0x16,
/* 0410 */  0x00, 0x8A, 0x1C, 0x00, 0x06, 0x11, 0x01, 0xC8,
/* 0418 */  0x16, 0x01, 0xC9, 0x19, 0x00, 0x8A, 0x1C, 0x01,
/* 0420 */  0xCA, 0x1C, 0x00, 0xCA, 0x1C, 0x01, 0x09, 0x1A,
/* 0428 */  0x00, 0x08, 0x17, 0x01, 0x46, 0x11, 0x01, 0x0A,
/* 0430 */  0x1D, 0x00, 0x48, 0x17, 0x00, 0x47, 0x14, 0x01,
/* 0438 */  0x49, 0x1A, 0x00, 0x48, 0x17, 0x01, 0x87, 0x14,
/* 0440 */  0x00, 0x49, 0x1A, 0x01, 0x4A, 0x1D, 0x01, 0x89,
/* 0448 */  0x1A, 0x00, 0x87, 0x14, 0x01, 0x88, 0x17, 0x01,
/* 0450 */  0x89, 0x1A, 0x01, 0xC7, 0x14, 0x00, 0xC8, 0x17,
/* 0458 */  0x00, 0xCA, 0x1D, 0x00, 0xC6, 0x11, 0x01, 0xC8,
/* 0460 */  0x17, 0x01, 0xC9, 0x1A, 0x01, 0xCA, 0x1D, 0x01,
/* 0468 */  0x0A, 0x1E, 0x00, 0x0A, 0x1E, 0x01, 0x09, 0x1B,
/* 0470 */  0x01, 0x08, 0x18, 0x01, 0x06, 0x12, 0x01, 0x4A,
/* 0478 */  0x1E, 0x00, 0x48, 0x18, 0x00, 0x47, 0x15, 0x00,
/* 0480 */  0x49, 0x1B, 0x01, 0x48, 0x18, 0x01, 0x47, 0x15,
/* 0488 */  0x01, 0x89, 0x1B, 0x00, 0x8A, 0x1E, 0x01, 0x89,
/* 0490 */  0x1B, 0x01, 0x87, 0x15, 0x00, 0x88, 0x18, 0x01,
```

```
    /* 0498 */  0xC9, 0x1B, 0x00, 0x87, 0x15, 0x01, 0xC8, 0x18,
    /* 04A0 */  0x00, 0x0A, 0x1F, 0x00, 0x86, 0x12, 0x01, 0xC8,
    /* 04A8 */  0x18, 0x01, 0x09, 0x1C, 0x00, 0x0A, 0x1F, 0x01,
    /* 04B0 */  0x4A, 0x1F, 0x00
})
Name (RTLB, Buffer (0x0322)
{
    /* 0000 */  0x28, 0x00, 0x28, 0x20, 0x28, 0x40, 0x28, 0x60,
    /* 0008 */  0x28, 0x80, 0x29, 0x00, 0x29, 0x20, 0x29, 0x40,
    /* 0010 */  0x29, 0x60, 0x29, 0x80, 0x2A, 0x00, 0x2A, 0x20,
    /* 0018 */  0x2A, 0x40, 0x2A, 0x60, 0x2A, 0x80, 0x2B, 0x00,
    /* 0020 */  0x2B, 0x20, 0x2B, 0x40, 0x2B, 0x60, 0x2B, 0x80,
    /* 0028 */  0x2C, 0x00, 0x2C, 0x20, 0x2C, 0x40, 0x2C, 0x60,
    /* 0030 */  0x2C, 0x80, 0x2D, 0x00, 0x2D, 0x20, 0x2D, 0x40,
    /* 0038 */  0x2D, 0x60, 0x2D, 0x80, 0x2E, 0x00, 0x2E, 0x20,
    /* 0040 */  0x2E, 0x40, 0x2E, 0x60, 0x2E, 0x80, 0x2F, 0x00,
    /* 0048 */  0x2F, 0x20, 0x2F, 0x40, 0x2F, 0x60, 0x2F, 0x80,
    /* 0050 */  0x30, 0x00, 0x30, 0x20, 0x30, 0x40, 0x30, 0x60,
    /* 0058 */  0x30, 0x80, 0x31, 0x00, 0x31, 0x20, 0x31, 0x40,
    /* 0060 */  0x31, 0x60, 0x31, 0x80, 0x32, 0x00, 0x32, 0x20,
    /* 0068 */  0x32, 0x40, 0x32, 0x60, 0x32, 0x80, 0x33, 0x00,
    /* 0070 */  0x33, 0x20, 0x33, 0x40, 0x33, 0x60, 0x33, 0x80,
    /* 0078 */  0x34, 0x00, 0x34, 0x20, 0x34, 0x40, 0x34, 0x60,
    /* 0080 */  0x34, 0x80, 0x35, 0x00, 0x35, 0x30, 0x35, 0x40,
    /* 0088 */  0x35, 0x60, 0x35, 0x80, 0x36, 0x00, 0x36, 0x20,
    /* 0090 */  0x36, 0x40, 0x36, 0x60, 0x36, 0x80, 0x37, 0x00,
    /* 0098 */  0x37, 0x20, 0x37, 0x40, 0x37, 0x60, 0x37, 0x80,
    /* 00A0 */  0x38, 0x00, 0x38, 0x20, 0x38, 0x40, 0x38, 0x60,
    /* 00A8 */  0x38, 0x80, 0x39, 0x00, 0x39, 0x20, 0x39, 0x40,
    /* 00B0 */  0x39, 0x60, 0x39, 0x80, 0x3A, 0x00, 0x3A, 0x20,
    /* 00B8 */  0x3A, 0x40, 0x3A, 0x60, 0x3A, 0x80, 0x3B, 0x00,
    /* 00C0 */  0x3B, 0x20, 0x3B, 0x40, 0x3B, 0x60, 0x3B, 0x80,
    /* 00C8 */  0x3C, 0x00, 0x3C, 0x20, 0x3C, 0x40, 0x3C, 0x60,
    /* 00D0 */  0x3C, 0x80, 0x3D, 0x00, 0x3D, 0x20, 0x3D, 0x40,
    /* 00D8 */  0x3D, 0x60, 0x3D, 0x80, 0x3E, 0x00, 0x3E, 0x20,
    /* 00E0 */  0x3E, 0x40, 0x3E, 0x60, 0x3E, 0x80, 0x3F, 0x00,
    /* 00E8 */  0x3F, 0x20, 0x3F, 0x40, 0x3F, 0x60, 0x3F, 0x80,
    /* 00F0 */  0x40, 0x00, 0x40, 0x20, 0x40, 0x40, 0x40, 0x60,
    /* 00F8 */  0x40, 0x80, 0x41, 0x00, 0x41, 0x20, 0x41, 0x40,
    /* 0100 */  0x41, 0x60, 0x41, 0x80, 0x42, 0x00, 0x42, 0x20,
    /* 0108 */  0x42, 0x40, 0x42, 0x60, 0x42, 0x80, 0x43, 0x00,
    /* 0110 */  0x43, 0x20, 0x43, 0x40, 0x43, 0x60, 0x43, 0x80,
    /* 0118 */  0x44, 0x00, 0x44, 0x20, 0x44, 0x40, 0x44, 0x60,
    /* 0120 */  0x44, 0x80, 0x45, 0x00, 0x45, 0x20, 0x45, 0x40,
    /* 0128 */  0x45, 0x60, 0x45, 0x80, 0x46, 0x00, 0x46, 0x20,
    /* 0130 */  0x46, 0x40, 0x46, 0x60, 0x46, 0x80, 0x47, 0x00,
    /* 0138 */  0x47, 0x20, 0x47, 0x40, 0x47, 0x60, 0x47, 0x80,
    /* 0140 */  0x48, 0x00, 0x48, 0x20, 0x48, 0x40, 0x48, 0x60,
    /* 0148 */  0x48, 0x80, 0x49, 0x00, 0x49, 0x20, 0x49, 0x40,
```

```
/* 0150 */  0x49, 0x60, 0x49, 0x80, 0x4A, 0x00, 0x4A, 0x20,
/* 0158 */  0x4A, 0x40, 0x4A, 0x60, 0x4A, 0x80, 0x4B, 0x00,
/* 0160 */  0x4B, 0x20, 0x4B, 0x40, 0x4B, 0x60, 0x4B, 0x80,
/* 0168 */  0x4C, 0x00, 0x4C, 0x20, 0x4C, 0x40, 0x4C, 0x60,
/* 0170 */  0x4C, 0x80, 0x4D, 0x00, 0x4D, 0x20, 0x4D, 0x40,
/* 0178 */  0x4D, 0x60, 0x4D, 0x80, 0x4E, 0x00, 0x4E, 0x20,
/* 0180 */  0x4E, 0x40, 0x4E, 0x60, 0x4E, 0x80, 0x4F, 0x00,
/* 0188 */  0x4F, 0x20, 0x4F, 0x40, 0x4F, 0x60, 0x4F, 0x80,
/* 0190 */  0x50, 0x00, 0x50, 0x20, 0x50, 0x40, 0x50, 0x60,
/* 0198 */  0x50, 0x80, 0x51, 0x00, 0x51, 0x20, 0x51, 0x40,
/* 01A0 */  0x51, 0x60, 0x51, 0x80, 0x52, 0x00, 0x52, 0x20,
/* 01A8 */  0x52, 0x40, 0x52, 0x60, 0x52, 0x80, 0x53, 0x00,
/* 01B0 */  0x53, 0x20, 0x53, 0x40, 0x53, 0x60, 0x53, 0x80,
/* 01B8 */  0x54, 0x00, 0x54, 0x20, 0x54, 0x40, 0x54, 0x60,
/* 01C0 */  0x54, 0x80, 0x55, 0x00, 0x55, 0x20, 0x55, 0x40,
/* 01C8 */  0x55, 0x60, 0x55, 0x80, 0x56, 0x00, 0x56, 0x20,
/* 01D0 */  0x56, 0x40, 0x56, 0x60, 0x56, 0x80, 0x57, 0x00,
/* 01D8 */  0x57, 0x20, 0x57, 0x40, 0x57, 0x60, 0x57, 0x80,
/* 01E0 */  0x58, 0x00, 0x58, 0x20, 0x58, 0x40, 0x58, 0x60,
/* 01E8 */  0x58, 0x80, 0x59, 0x00, 0x59, 0x20, 0x59, 0x40,
/* 01F0 */  0x59, 0x60, 0x59, 0x80, 0x5A, 0x00, 0x5A, 0x20,
/* 01F8 */  0x5A, 0x40, 0x5A, 0x60, 0x5A, 0x80, 0x5B, 0x00,
/* 0200 */  0x5B, 0x20, 0x5B, 0x40, 0x5B, 0x60, 0x5B, 0x80,
/* 0208 */  0x5C, 0x00, 0x5C, 0x20, 0x5C, 0x40, 0x5C, 0x60,
/* 0210 */  0x5C, 0x80, 0x5D, 0x00, 0x5D, 0x20, 0x5D, 0x40,
/* 0218 */  0x5D, 0x60, 0x5D, 0x80, 0x5E, 0x00, 0x5E, 0x20,
/* 0220 */  0x5E, 0x40, 0x5E, 0x60, 0x5E, 0x80, 0x5F, 0x00,
/* 0228 */  0x5F, 0x20, 0x5F, 0x40, 0x5F, 0x60, 0x5F, 0x80,
/* 0230 */  0x60, 0x00, 0x60, 0x20, 0x60, 0x40, 0x60, 0x60,
/* 0238 */  0x60, 0x80, 0x61, 0x00, 0x61, 0x20, 0x61, 0x40,
/* 0240 */  0x61, 0x60, 0x61, 0x80, 0x62, 0x00, 0x62, 0x20,
/* 0248 */  0x62, 0x40, 0x62, 0x60, 0x62, 0x80, 0x63, 0x00,
/* 0250 */  0x63, 0x20, 0x63, 0x40, 0x63, 0x60, 0x63, 0x80,
/* 0258 */  0x64, 0x00, 0x64, 0x20, 0x64, 0x40, 0x64, 0x60,
/* 0260 */  0x64, 0x80, 0x65, 0x00, 0x65, 0x20, 0x65, 0x40,
/* 0268 */  0x65, 0x60, 0x65, 0x80, 0x66, 0x00, 0x66, 0x20,
/* 0270 */  0x66, 0x40, 0x66, 0x60, 0x66, 0x80, 0x67, 0x00,
/* 0278 */  0x67, 0x20, 0x67, 0x40, 0x67, 0x60, 0x67, 0x80,
/* 0280 */  0x68, 0x00, 0x68, 0x20, 0x68, 0x40, 0x68, 0x60,
/* 0288 */  0x68, 0x80, 0x69, 0x00, 0x69, 0x20, 0x69, 0x40,
/* 0290 */  0x69, 0x60, 0x69, 0x80, 0x6A, 0x00, 0x6A, 0x20,
/* 0298 */  0x6A, 0x40, 0x6A, 0x60, 0x6A, 0x80, 0x6B, 0x00,
/* 02A0 */  0x6B, 0x20, 0x6B, 0x40, 0x6B, 0x60, 0x6B, 0x80,
/* 02A8 */  0x6C, 0x00, 0x6C, 0x20, 0x6C, 0x40, 0x6C, 0x60,
/* 02B0 */  0x6C, 0x80, 0x6D, 0x00, 0x6D, 0x20, 0x6D, 0x40,
/* 02B8 */  0x6D, 0x60, 0x6D, 0x80, 0x6E, 0x00, 0x6E, 0x20,
/* 02C0 */  0x6E, 0x40, 0x6E, 0x60, 0x6E, 0x80, 0x6F, 0x00,
/* 02C8 */  0x6F, 0x20, 0x6F, 0x40, 0x6F, 0x60, 0x6F, 0x80,
/* 02D0 */  0x70, 0x00, 0x70, 0x20, 0x70, 0x40, 0x70, 0x60,
```

```
        /* 02D8 */  0x70, 0x80, 0x71, 0x00, 0x71, 0x20, 0x71, 0x40,
        /* 02E0 */  0x71, 0x60, 0x71, 0x80, 0x72, 0x00, 0x72, 0x20,
        /* 02E8 */  0x72, 0x40, 0x72, 0x60, 0x72, 0x80, 0x73, 0x00,
        /* 02F0 */  0x73, 0x20, 0x73, 0x40, 0x73, 0x60, 0x73, 0x80,
        /* 02F8 */  0x74, 0x00, 0x74, 0x20, 0x74, 0x40, 0x74, 0x60,
        /* 0300 */  0x74, 0x80, 0x75, 0x00, 0x75, 0x20, 0x75, 0x40,
        /* 0308 */  0x75, 0x60, 0x75, 0x80, 0x76, 0x00, 0x76, 0x20,
        /* 0310 */  0x76, 0x40, 0x76, 0x60, 0x76, 0x80, 0x77, 0x00,
        /* 0318 */  0x77, 0x20, 0x77, 0x40, 0x77, 0x60, 0x77, 0x80,
        /* 0320 */  0x78, 0x00
    })
}

Scope (_SB.PCI0.SBRG.ASOC)
{
    Name (G4T0, Package (0x04)
    {
        0x04070010,
        "CPU Q-FAN Control",
        0x80000000,
        Zero
    })
    Name (G4T1, Package (0x08)
    {
        0x04080011,
        "CPU Q-FAN Profile",
        0x00100001,
        Zero,
        0x03,
        "Optimal",
        "Silent",
        "Performance"
    })
    Name (GRP4, Package (0x02)
    {
        G4T0,
        G4T1
    })
    Method (GIT4, 1, NotSerialized)
    {
        Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
        Store (And (Arg0, 0xFFFF), _T_0)
        If (LEqual (_T_0, 0x10))
        {
            Store (GNVS (0x157B), ASB1)
        }
        ElseIf (LEqual (_T_0, 0x11))
        {
            Store (GNVS (0x24F4), ASB1)
```

```
        }
        ElseIf (LEqual (_T_0, 0x17)) {}
        Else
        {
            Store (Zero, ASB0)
        }
    }

    Method (SIT4, 3, NotSerialized)
    {
        Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
        Store (And (Arg0, 0xFFFF), _T_0)
        If (LEqual (_T_0, 0x10))
        {
            If (LNotEqual (GNVS (0x157B), Arg1))
            {
                If (And (Arg2, One))
                {
                    SNVS (0x157B, Arg1)
                }

                Or (ASB0, 0x02, ASB0)
            }
        }
        ElseIf (LEqual (_T_0, 0x11))
        {
            If (LNotEqual (GNVS (0x24F4), Arg1))
            {
                If (And (Arg2, One))
                {
                    SNVS (0x24F4, Arg1)
                }

                Or (ASB0, 0x02, ASB0)
            }
        }
        Else
        {
            Store (Zero, ASB0)
        }
    }
}

Scope (_SB.PCI0.SBRG.ASOC)
{
    Name (G5T2, Package (0x08)
    {
        0x05080002,
        "AI Profile",
```

```
    Zero,
    Zero,
    0x03,
    "Race Car",
    "Jet Plane",
    "Rocket"
})
Name (GRP5, Package (0x01)
{
    G5T2
})
Name (BUF0, Package (0x02)
{
    0x03010011,
    Ones
})
Name (BUF1, Package (0x02)
{
    0x03010011,
    Ones
})
Name (BUF2, Package (0x02)
{
    0x03010011,
    Ones
})
Method (GIT5, 2, NotSerialized)
{
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (And (Arg0, 0xFFFF), _T_0)
    If (LEqual (_T_0, 0x02))
    {
        UDBF (Arg1)
        MVBF (Arg1)
        Store (0x05, ASB1)
    }
    Else
    {
        Store (Zero, ASB0)
    }
}

Method (SIT5, 3, NotSerialized)
{
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (And (Arg0, 0xFFFF), _T_0)
    If (LEqual (_T_0, Zero))
    {
        Store (0x03, ASB0)
```

```
    }
    Else
    {
        Store (Zero, ASB0)
    }
}

Method (UDBF, 1, NotSerialized)
{
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (And (Arg0, 0xFFFF), _T_0)
    If (LEqual (_T_0, Zero))
    {
        Store (OCFS (One), Index (BUF0, One))
    }
    ElseIf (LEqual (_T_0, One))
    {
        Store (OCFS (0x03), Index (BUF1, One))
    }
    ElseIf (LEqual (_T_0, 0x02))
    {
        Store (OCFS (0x05), Index (BUF2, One))
    }
}

Method (MVBF, 1, NotSerialized)
{
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (And (Arg0, 0xFFFF), _T_0)
    If (LEqual (_T_0, Zero))
    {
        Store (Zero, Local0)
        Store (0x08, Local2)
        Store (SizeOf (BUF0), Local3)
        While (LNotEqual (Local0, Local3))
        {
            Add (Local0, One, Local1)
            If (LNotEqual (DerefOf (Index (BUF0, Local1)), Ones))
            {
                STBF (DerefOf (Index (BUF0, Local0)), Local2)
                Add (Local2, 0x04, Local2)
                STBF (DerefOf (Index (BUF0, Local1)), Local2)
                Add (Local2, 0x04, Local2)
            }

            STBF (Ones, Local2)
            Add (Local0, 0x02, Local0)
        }
    }
```

```
    ElseIf (LEqual (_T_0, One))
    {
        Store (Zero, Local0)
        Store (0x08, Local2)
        Store (SizeOf (BUF1), Local3)
        While (LNotEqual (Local0, Local3))
        {
            Add (Local0, One, Local1)
            If (LNotEqual (DerefOf (Index (BUF1, Local1)), Ones))
            {
                STBF (DerefOf (Index (BUF1, Local0)), Local2)
                Add (Local2, 0x04, Local2)
                STBF (DerefOf (Index (BUF1, Local1)), Local2)
                Add (Local2, 0x04, Local2)
            }

            STBF (Ones, Local2)
            Add (Local0, 0x02, Local0)
        }
    }
    ElseIf (LEqual (_T_0, 0x02))
    {
        Store (Zero, Local0)
        Store (0x08, Local2)
        Store (SizeOf (BUF2), Local3)
        While (LNotEqual (Local0, Local3))
        {
            Add (Local0, One, Local1)
            If (LNotEqual (DerefOf (Index (BUF2, Local1)), Ones))
            {
                STBF (DerefOf (Index (BUF2, Local0)), Local2)
                Add (Local2, 0x04, Local2)
                STBF (DerefOf (Index (BUF2, Local1)), Local2)
                Add (Local2, 0x04, Local2)
            }

            STBF (Ones, Local2)
            Add (Local0, 0x02, Local0)
        }
    }
}

Method (STBF, 2, NotSerialized)
{
    Store (Arg0, Local0)
    Store (Arg1, Local1)
    Store (Zero, Local2)
    While (LNotEqual (Local2, 0x20))
    {
```

```
            Store (ShiftRight (Local0, Local2), Index (ASBF, Local1))
            Add (Local2, 0x08, Local2)
            Add (Local1, One, Local1)
        }
    }

    Method (OCFS, 1, NotSerialized)
    {
        Store (Arg0, OCPL)
        Store (DerefOf (Index (G3T0, 0x03)), FSBS)
        Add (Arg0, 0x64, Local0)
        Multiply (Add (0xC8, DerefOf (Index (G3T0, 0x03))), Local0, Local0)
        Divide (Local0, 0x64, Local5, Local0)
        Subtract (Local0, 0xC8, Local0)
        Return (Local0)
    }
}

Scope (_SB.PCI0.SBRG.ASOC)
{
    Name (G6T1, Package (0x07)
    {
        0x06020011,
        "Vcore Voltage",
        0x20000000,
        Zero,
        0x0352,
        0x02EE,
        0x02
    })
    Name (G6T2, Package (0x07)
    {
        0x06030012,
        "CPU Temperature",
        0x20000000,
        Zero,
        0x0258,
        0x015E,
        0x02
    })
    Name (G6T3, Package (0x07)
    {
        0x06040013,
        "CPU FAN Speed",
        0x20000000,
        Zero,
        0x0320,
        0x1900,
        0x02
```

```
})
Name (G6T6, Package (0x07)
{
    0x06020061,
    "+12V Voltage",
    0x20000000,
    0x2EE0,
    0x27D8,
    0x0E10,
    0x02
})
Name (G6T7, Package (0x07)
{
    0x06020062,
    "+5V Voltage",
    0x20000000,
    0x1388,
    0x1194,
    0x03E8,
    0x02
})
Name (G6T8, Package (0x07)
{
    0x06020063,
    "+3.3V Voltage",
    0x20000000,
    0x0CE4,
    0x0B9A,
    0x0294,
    0x02
})
Name (G6T9, Package (0x07)
{
    0x06030074,
    "MB Temperature",
    0x20000000,
    Zero,
    0x01C2,
    0x01F4,
    0x02
})
Name (GRP6, Package (0x07)
{
    G6T1,
    G6T2,
    G6T3,
    G6T6,
    G6T7,
    G6T8,
```

```
    G6T9
})
Method (GIT6, 1, NotSerialized)
{
    Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
    Store (And (Arg0, 0xFFFF), _T_0)
    If (LEqual (_T_0, 0x11))
    {
        Store (0x0600, ASB1)
    }
    ElseIf (LEqual (_T_0, 0x12))
    {
        Store (0x0601, ASB1)
    }
    ElseIf (LEqual (_T_0, 0x13))
    {
        Store (0x0602, ASB1)
    }
    ElseIf (LEqual (_T_0, 0x73))
    {
        Store (0x0603, ASB1)
    }
    ElseIf (LEqual (_T_0, 0xC3))
    {
        Store (0x0604, ASB1)
    }
    ElseIf (LEqual (_T_0, 0x61))
    {
        Store (0x0605, ASB1)
    }
    ElseIf (LEqual (_T_0, 0x62))
    {
        Store (0x0606, ASB1)
    }
    ElseIf (LEqual (_T_0, 0x63))
    {
        Store (0x0607, ASB1)
    }
    ElseIf (LEqual (_T_0, 0x74))
    {
        Store (0x0608, ASB1)
    }
    Else
    {
        Store (Zero, ASB0)
    }
}

Method (SIT6, 3, NotSerialized)
```

```
    {
        Name (_T_0, Zero)  // _T_x: Emitted by ASL Compiler
        Store (And (Arg0, 0xFFFF), _T_0)
        If (LEqual (_T_0, 0x11))
        {
            Store (0x0600, DBG8)
        }
        ElseIf (LEqual (_T_0, 0x12))
        {
            Store (0x0601, DBG8)
        }
        ElseIf (LEqual (_T_0, 0x13))
        {
            Store (0x0602, DBG8)
        }
        ElseIf (LEqual (_T_0, 0x73))
        {
            Store (0x0603, DBG8)
        }
        ElseIf (LEqual (_T_0, 0xC3))
        {
            Store (0x0604, DBG8)
        }
        ElseIf (LEqual (_T_0, 0x61))
        {
            Store (0x0605, DBG8)
        }
        ElseIf (LEqual (_T_0, 0x62))
        {
            Store (0x0606, DBG8)
        }
        ElseIf (LEqual (_T_0, 0x63))
        {
            Store (0x0607, DBG8)
        }
        ElseIf (LEqual (_T_0, 0x74))
        {
            Store (0x0608, DBG8)
        }
        Else
        {
            Store (Zero, ASB0)
        }
    }
}

OperationRegion (_SB.PCI0.SBRG.PIX0, PCI_Config, 0x60, 0x0C)
Field (\_SB.PCI0.SBRG.PIX0, ByteAcc, NoLock, Preserve)
{
```

```
    PIRA,   8,
    PIRB,   8,
    PIRC,   8,
    PIRD,   8,
    Offset (0x08),
    PIRE,   8,
    PIRF,   8,
    PIRG,   8,
    PIRH,   8
}

Scope (_SB)
{
    Name (BUFA, ResourceTemplate ()
    {
        IRQ (Level, ActiveLow, Shared, )
            {15}
    })
    CreateWordField (BUFA, One, IRA0)
    Device (LNKA)
    {
        Name (_HID, EisaId ("PNP0C0F"))  // _HID: Hardware ID
        Name (_UID, One)  // _UID: Unique ID
        Method (_STA, 0, NotSerialized)  // _STA: Status
        {
            And (PIRA, 0x80, Local0)
            If (Local0)
            {
                Return (0x09)
            }
            Else
            {
                Return (0x0B)
            }
        }

        Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
        {
            Return (PRSA)
        }

        Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
        {
            Or (PIRA, 0x80, PIRA)
        }

        Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
        {
            And (PIRA, 0x0F, Local0)
```

```
        ShiftLeft (One, Local0, IRA0)
        Return (BUFA)
    }

    Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
    {
        CreateWordField (Arg0, One, IRA)
        FindSetRightBit (IRA, Local0)
        Decrement (Local0)
        Store (Local0, PIRA)
    }
}

Device (LNKB)
{
    Name (_HID, EisaId ("PNP0C0F"))  // _HID: Hardware ID
    Name (_UID, 0x02)  // _UID: Unique ID
    Method (_STA, 0, NotSerialized)  // _STA: Status
    {
        And (PIRB, 0x80, Local0)
        If (Local0)
        {
            Return (0x09)
        }
        Else
        {
            Return (0x0B)
        }
    }

    Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
    {
        Return (PRSB)
    }

    Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
    {
        Or (PIRB, 0x80, PIRB)
    }

    Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
    {
        And (PIRB, 0x0F, Local0)
        ShiftLeft (One, Local0, IRA0)
        Return (BUFA)
    }

    Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
    {
```

```
        CreateWordField (Arg0, One, IRA)
        FindSetRightBit (IRA, Local0)
        Decrement (Local0)
        Store (Local0, PIRB)
    }
}

Device (LNKC)
{
    Name (_HID, EisaId ("PNP0C0F"))  // _HID: Hardware ID
    Name (_UID, 0x03)  // _UID: Unique ID
    Method (_STA, 0, NotSerialized)  // _STA: Status
    {
        And (PIRC, 0x80, Local0)
        If (Local0)
        {
            Return (0x09)
        }
        Else
        {
            Return (0x0B)
        }
    }

    Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
    {
        Return (PRSC)
    }

    Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
    {
        Or (PIRC, 0x80, PIRC)
    }

    Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
    {
        And (PIRC, 0x0F, Local0)
        ShiftLeft (One, Local0, IRA0)
        Return (BUFA)
    }

    Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
    {
        CreateWordField (Arg0, One, IRA)
        FindSetRightBit (IRA, Local0)
        Decrement (Local0)
        Store (Local0, PIRC)
    }
}
```

```
Device (LNKD)
{
   Name (_HID, EisaId ("PNP0C0F"))  // _HID: Hardware ID
   Name (_UID, 0x04)  // _UID: Unique ID
   Method (_STA, 0, NotSerialized)  // _STA: Status
   {
      And (PIRD, 0x80, Local0)
      If (Local0)
      {
         Return (0x09)
      }
      Else
      {
         Return (0x0B)
      }
   }

   Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
   {
      Return (PRSD)
   }

   Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
   {
      Or (PIRD, 0x80, PIRD)
   }

   Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
   {
      And (PIRD, 0x0F, Local0)
      ShiftLeft (One, Local0, IRA0)
      Return (BUFA)
   }

   Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
   {
      CreateWordField (Arg0, One, IRA)
      FindSetRightBit (IRA, Local0)
      Decrement (Local0)
      Store (Local0, PIRD)
   }
}

Device (LNKE)
{
   Name (_HID, EisaId ("PNP0C0F"))  // _HID: Hardware ID
   Name (_UID, 0x05)  // _UID: Unique ID
   Method (_STA, 0, NotSerialized)  // _STA: Status
```

```
    {
        And (PIRE, 0x80, Local0)
        If (Local0)
        {
            Return (0x09)
        }
        Else
        {
            Return (0x0B)
        }
    }

    Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
    {
        Return (PRSE)
    }

    Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
    {
        Or (PIRE, 0x80, PIRE)
    }

    Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
    {
        And (PIRE, 0x0F, Local0)
        ShiftLeft (One, Local0, IRA0)
        Return (BUFA)
    }

    Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
    {
        CreateWordField (Arg0, One, IRA)
        FindSetRightBit (IRA, Local0)
        Decrement (Local0)
        Store (Local0, PIRE)
    }
}

Device (LNKF)
{
    Name (_HID, EisaId ("PNP0C0F"))  // _HID: Hardware ID
    Name (_UID, 0x06)  // _UID: Unique ID
    Method (_STA, 0, NotSerialized)  // _STA: Status
    {
        And (PIRF, 0x80, Local0)
        If (Local0)
        {
            Return (0x09)
        }
```

```
        Else
        {
            Return (0x0B)
        }
    }

    Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
    {
        Return (PRSF)
    }

    Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
    {
        Or (PIRF, 0x80, PIRF)
    }

    Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
    {
        And (PIRF, 0x0F, Local0)
        ShiftLeft (One, Local0, IRA0)
        Return (BUFA)
    }

    Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
    {
        CreateWordField (Arg0, One, IRA)
        FindSetRightBit (IRA, Local0)
        Decrement (Local0)
        Store (Local0, PIRF)
    }
}

Device (LNKG)
{
    Name (_HID, EisaId ("PNP0C0F"))  // _HID: Hardware ID
    Name (_UID, 0x07)  // _UID: Unique ID
    Method (_STA, 0, NotSerialized)  // _STA: Status
    {
        And (PIRG, 0x80, Local0)
        If (Local0)
        {
            Return (0x09)
        }
        Else
        {
            Return (0x0B)
        }
    }
```

```
    Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
    {
        Return (PRSG)
    }

    Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
    {
        Or (PIRG, 0x80, PIRG)
    }

    Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
    {
        And (PIRG, 0x0F, Local0)
        ShiftLeft (One, Local0, IRA0)
        Return (BUFA)
    }

    Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
    {
        CreateWordField (Arg0, One, IRA)
        FindSetRightBit (IRA, Local0)
        Decrement (Local0)
        Store (Local0, PIRG)
    }
}

Device (LNKH)
{
    Name (_HID, EisaId ("PNP0C0F"))  // _HID: Hardware ID
    Name (_UID, 0x08)  // _UID: Unique ID
    Method (_STA, 0, NotSerialized)  // _STA: Status
    {
        And (PIRH, 0x80, Local0)
        If (Local0)
        {
            Return (0x09)
        }
        Else
        {
            Return (0x0B)
        }
    }

    Method (_PRS, 0, NotSerialized)  // _PRS: Possible Resource Settings
    {
        Return (PRSH)
    }

    Method (_DIS, 0, NotSerialized)  // _DIS: Disable Device
```

```
      {
         Or (PIRH, 0x80, PIRH)
      }

      Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
      {
         And (PIRH, 0x0F, Local0)
         ShiftLeft (One, Local0, IRA0)
         Return (BUFA)
      }

      Method (_SRS, 1, NotSerialized)  // _SRS: Set Resource Settings
      {
         CreateWordField (Arg0, One, IRA)
         FindSetRightBit (IRA, Local0)
         Decrement (Local0)
         Store (Local0, PIRH)
      }
   }
}

Scope (_SB)
{
   Name (XCPD, Zero)
   Name (XNPT, One)
   Name (XCAP, 0x02)
   Name (XDCP, 0x04)
   Name (XDCT, 0x08)
   Name (XDST, 0x0A)
   Name (XLCP, 0x0C)
   Name (XLCT, 0x10)
   Name (XLST, 0x12)
   Name (XSCP, 0x14)
   Name (XSCT, 0x18)
   Name (XSST, 0x1A)
   Name (XRCT, 0x1C)
   Mutex (MUTE, 0x00)
   Method (RBPE, 1, NotSerialized)
   {
      Acquire (MUTE, 0x03E8)
      Add (Arg0, PCIB, Local0)
      OperationRegion (PCFG, SystemMemory, Local0, One)
      Field (PCFG, ByteAcc, NoLock, Preserve)
      {
         XCFG,   8
      }

      Release (MUTE)
      Return (XCFG)
```

```
}

Method (RWPE, 1, NotSerialized)
{
   Acquire (MUTE, 0x03E8)
   And (Arg0, 0xFFFFFFFE, Arg0)
   Add (Arg0, PCIB, Local0)
   OperationRegion (PCFG, SystemMemory, Local0, 0x02)
   Field (PCFG, WordAcc, NoLock, Preserve)
   {
      XCFG,   16
   }

   Release (MUTE)
   Return (XCFG)
}

Method (RDPE, 1, NotSerialized)
{
   Acquire (MUTE, 0x03E8)
   And (Arg0, 0xFFFFFFFC, Arg0)
   Add (Arg0, PCIB, Local0)
   OperationRegion (PCFG, SystemMemory, Local0, 0x04)
   Field (PCFG, DWordAcc, NoLock, Preserve)
   {
      XCFG,   32
   }

   Release (MUTE)
   Return (XCFG)
}

Method (WBPE, 2, NotSerialized)
{
   Acquire (MUTE, 0x0FFF)
   Add (Arg0, PCIB, Local0)
   OperationRegion (PCFG, SystemMemory, Local0, One)
   Field (PCFG, ByteAcc, NoLock, Preserve)
   {
      XCFG,   8
   }

   Store (Arg1, XCFG)
   Release (MUTE)
}

Method (WWPE, 2, NotSerialized)
{
   Acquire (MUTE, 0x03E8)
```

```
    And (Arg0, 0xFFFFFFFE, Arg0)
    Add (Arg0, PCIB, Local0)
    OperationRegion (PCFG, SystemMemory, Local0, 0x02)
    Field (PCFG, WordAcc, NoLock, Preserve)
    {
        XCFG,   16
    }

    Store (Arg1, XCFG)
    Release (MUTE)
}

Method (WDPE, 2, NotSerialized)
{
    Acquire (MUTE, 0x03E8)
    And (Arg0, 0xFFFFFFFC, Arg0)
    Add (Arg0, PCIB, Local0)
    OperationRegion (PCFG, SystemMemory, Local0, 0x04)
    Field (PCFG, DWordAcc, NoLock, Preserve)
    {
        XCFG,   32
    }

    Store (Arg1, XCFG)
    Release (MUTE)
}

Method (RWDP, 3, NotSerialized)
{
    Acquire (MUTE, 0x03E8)
    And (Arg0, 0xFFFFFFFC, Arg0)
    Add (Arg0, PCIB, Local0)
    OperationRegion (PCFG, SystemMemory, Local0, 0x04)
    Field (PCFG, DWordAcc, NoLock, Preserve)
    {
        XCFG,   32
    }

    And (XCFG, Arg2, Local1)
    Or (Local1, Arg1, XCFG)
    Release (MUTE)
}

Method (RPME, 1, NotSerialized)
{
    Add (Arg0, 0x84, Local0)
    Store (RDPE (Local0), Local1)
    If (LEqual (Local1, Ones))
    {
```

```
            Return (Zero)
        }
        Else
        {
            If (LAnd (Local1, 0x00010000))
            {
                WDPE (Local0, And (Local1, 0x00010000))
                Return (One)
            }

            Return (Zero)
        }
    }
}

OperationRegion (SMRG, SystemIO, 0x0400, 0x10)
Field (SMRG, ByteAcc, NoLock, Preserve)
{
    HSTS,   8,
    SSTS,   8,
    HSTC,   8,
    HCMD,   8,
    HADR,   8,
    HDT0,   8,
    HDT1,   8,
    BLKD,   8,
    SLCT,   8,
    SHCM,   8,
    SLEV,   24,
    AUXC,   8
}

Field (SMRG, ByteAcc, NoLock, Preserve)
{
    Offset (0x05),
    HDTW,   16
}

Method (SCMD, 4, Serialized)
{
    Or (AUXC, 0x02, AUXC)
    Store (0x05, Local0)
    While (Decrement (Local0))
    {
        Store (0xFFFF, Local1)
        While (LAnd (HSTS, Decrement (Local1)))
        {
            Store (0xFE, HSTS)
            Stall (0x0A)
```

```
        }

        Store (HSTC, Local2)
        Store (Arg0, HADR)
        Store (Arg1, HCMD)
        Store (Arg2, HDTW)
        Store (Arg3, HSTC)
        Store (0xFFFF, Local1)
        While (Decrement (Local1))
        {
            If (And (HSTS, 0x0C))
            {
                Store (One, Local1)
            }

            If (LEqual (And (HSTS, 0x03), 0x02))
            {
                Return (HDTW)
            }

            Stall (0x0A)
        }

        Store (0x42, HSTC)
        Store (0xFFFF, Local1)
        While (Decrement (Local1))
        {
            If (And (HSTS, 0x10))
            {
                Store (One, Local1)
            }

            Stall (0x0A)
        }

        Store (Zero, HSTC)
    }

    Return (Ones)
}

Method (SBYT, 2, NotSerialized)
{
    SCMD (Arg0, Arg1, Zero, 0x44)
}

Method (WBYT, 3, NotSerialized)
{
    SCMD (Arg0, Arg1, Arg2, 0x48)
```

```
}

Method (WWRD, 3, NotSerialized)
{
   SCMD (Arg0, Arg1, Arg2, 0x4C)
}

Method (RSBT, 2, NotSerialized)
{
   Or (Arg0, One, Arg0)
   Return (And (SCMD (Arg0, Arg1, Zero, 0x44), 0xFF))
}

Method (RBYT, 2, NotSerialized)
{
   Or (Arg0, One, Arg0)
   Return (And (SCMD (Arg0, Arg1, Zero, 0x48), 0xFF))
}

Method (RWRD, 2, NotSerialized)
{
   Or (Arg0, One, Arg0)
   Return (SCMD (Arg0, Arg1, Zero, 0x4C))
}

Method (RBLK, 3, NotSerialized)
{
   Or (Arg0, One, Local0)
   SCMD (Local0, Arg1, Arg2, 0x54)
   Store (HSTC, Local0)
   Store (HDT0, Local0)
   Add (Local0, One, Local7)
   Name (RBUF, Buffer (Local7) {})
   Store (Zero, Local1)
   While (Local0)
   {
      Store (BLKD, Index (RBUF, Local1))
      Decrement (Local0)
      Increment (Local1)
   }

   Return (RBUF)
}

Method (WBLK, 4, NotSerialized)
{
   Store (HSTC, Local0)
   Store (Zero, Local0)
   While (LLessEqual (Local0, Arg2))
```

```
    {
        Store (DerefOf (Index (Arg3, Local0)), BLKD)
        Increment (Local0)
    }

    Store (HSTC, Local0)
    SCMD (Arg0, Arg1, Arg2, 0x54)
}

Scope (_SB.PCI0.SBRG.SIOR)
{
    Method (HWV0, 0, NotSerialized)
    {
        Return (Multiply (VCOR, 0x08))
    }

    Method (HWV1, 0, NotSerialized)
    {
        Multiply (V12V, 0x08, Local0)
        Return (Local0)
    }

    Method (HWV3, 0, NotSerialized)
    {
        Multiply (V33V, 0x08, Local0)
        Return (Local0)
    }

    Method (HWV5, 0, NotSerialized)
    {
        Multiply (V50V, 0x08, Local0)
        Return (Local0)
    }

    Method (HWT0, 0, NotSerialized)
    {
        Store (MBTE, Local1)
        Multiply (Local1, 0x0A, Local1)
        Return (Local1)
    }

    Method (HWT1, 0, NotSerialized)
    {
        Store (One, BSEL)
        Store (TSR1, Local1)
        Multiply (Local1, 0x0A, Local1)
        Store (TSR2, Local2)
        Multiply (Local2, 0x05, Local2)
        Add (Local1, Local2, Local1)
```

```
            Return (Local1)
        }

        Method (HWT2, 0, NotSerialized)
        {
            Store (0x02, BSEL)
            Store (TSR1, Local1)
            Multiply (Local1, 0x0A, Local1)
            Store (TSR2, Local2)
            Multiply (Local2, 0x05, Local2)
            Add (Local1, Local2, Local1)
            Return (Local1)
        }

        Method (HWF0, 0, NotSerialized)
        {
            Store (FAN1, Local0)
            Store (Zero, BSEL)
            And (FD21, 0x20, Local1)
            ShiftRight (Local1, 0x05, Local1)
            Multiply (Local1, 0x04, Local1)
            Add (Local1, FD11, Local1)
            While (LOr (LAnd (LGreater (Local0, 0xE6), LLess (Local1, 0x07)), LAnd
(LLess (Local0, 0x64), LGreater (Local1, Zero))))
            {
                If (LAnd (LGreater (Local0, 0xE6), LLess (Local1, 0x07)))
                {
                    Add (Local1, One, Local1)
                    Divide (Local1, 0x04, Local2, Local3)
                    Store (Zero, BSEL)
                    ShiftLeft (Local3, 0x05, Local3)
                    Store (FD21, Local4)
                    And (Local4, 0xDF, Local4)
                    Or (Local3, Local4, FD21)
                    Store (Local2, FD11)
                    Sleep (0x012C)
                    Store (FAN1, Local0)
                }
                Else
                {
                    Subtract (Local1, One, Local1)
                    Divide (Local1, 0x04, Local2, Local3)
                    Store (Zero, BSEL)
                    ShiftLeft (Local3, 0x05, Local3)
                    Store (FD21, Local4)
                    And (Local4, 0xDF, Local4)
                    Or (Local3, Local4, FD21)
                    Store (Local2, FD11)
                    Sleep (0x012C)
```

```
                Store (FAN1, Local0)
            }
        }

        If (LAnd (LEqual (Local0, 0xFF), LEqual (Local1, 0x07)))
        {
            Return (Zero)
        }

        If (LAnd (LEqual (Local0, Zero), LEqual (Local1, Zero)))
        {
            Return (0xFFFF)
        }

        Store (One, Local2)
        While (Local1)
        {
            Multiply (Local2, 0x02, Local2)
            Decrement (Local1)
        }

        Multiply (Local0, Local2, Local0)
        Divide (0x00149970, Local0, Local1, Local0)
        Return (Local0)
    }

    Method (HWF1, 0, NotSerialized)
    {
        Store (FAN2, Local0)
        Store (Zero, BSEL)
        And (FD21, 0x40, Local1)
        ShiftRight (Local1, 0x06, Local1)
        Multiply (Local1, 0x04, Local1)
        Add (Local1, FD12, Local1)
        While (LOr (LAnd (LGreater (Local0, 0xE6), LLess (Local1, 0x07)), LAnd
(LLess (Local0, 0x64), LGreater (Local1, Zero))))
        {
            If (LAnd (LGreater (Local0, 0xE6), LLess (Local1, 0x07)))
            {
                Add (Local1, One, Local1)
                Divide (Local1, 0x04, Local2, Local3)
                Store (Zero, BSEL)
                ShiftLeft (Local3, 0x06, Local3)
                Store (FD21, Local4)
                And (Local4, 0xBF, Local4)
                Or (Local3, Local4, FD21)
                Store (Local2, FD12)
                Sleep (0x0258)
                Store (FAN2, Local0)
```

```
            }
            Else
            {
                Subtract (Local1, One, Local1)
                Divide (Local1, 0x04, Local2, Local3)
                Store (Zero, BSEL)
                ShiftLeft (Local3, 0x06, Local3)
                Store (FD21, Local4)
                And (Local4, 0xBF, Local4)
                Or (Local3, Local4, FD21)
                Store (Local2, FD12)
                Sleep (0x0258)
                Store (FAN2, Local0)
            }
        }

        If (LAnd (LEqual (Local0, 0xFF), LEqual (Local1, 0x07)))
        {
            Return (Zero)
        }

        If (LAnd (LEqual (Local0, Zero), LEqual (Local1, Zero)))
        {
            Return (0xFFFF)
        }

        Store (One, Local2)
        While (Local1)
        {
            Multiply (Local2, 0x02, Local2)
            Decrement (Local1)
        }

        Multiply (Local0, Local2, Local0)
        Divide (0x00149970, Local0, Local1, Local0)
        Return (Local0)
    }

    Method (HWF2, 0, NotSerialized)
    {
        Store (FAN3, Local0)
        Store (Zero, BSEL)
        And (FD21, 0x80, Local1)
        ShiftRight (Local1, 0x07, Local1)
        Multiply (Local1, 0x04, Local1)
        Divide (FD13, 0x40, Local2, Local3)
        Add (Local1, Local3, Local1)
        While (LOr (LAnd (LGreater (Local0, 0xE6), LLess (Local1, 0x07)), LAnd
(LLess (Local0, 0x64), LGreater (Local1, Zero))))
```

```
{
    If (LAnd (LGreater (Local0, 0xE6), LLess (Local1, 0x07)))
    {
        Add (Local1, One, Local1)
        Divide (Local1, 0x04, Local2, Local3)
        Store (Zero, BSEL)
        ShiftLeft (Local3, 0x07, Local3)
        Store (FD21, Local4)
        And (Local4, 0x7F, Local4)
        Or (Local3, Local4, FD21)
        Store (FD13, Local3)
        And (Local3, 0x3F, Local3)
        Multiply (Local2, 0x40, Local2)
        Add (Local3, Local2, Local2)
        Store (Local2, FD13)
        Sleep (0x012C)
        Store (FAN3, Local0)
    }
    Else
    {
        Subtract (Local1, One, Local1)
        Divide (Local1, 0x04, Local2, Local3)
        Store (Zero, BSEL)
        ShiftLeft (Local3, 0x07, Local3)
        Store (FD21, Local4)
        And (Local4, 0x7F, Local4)
        Or (Local3, Local4, FD21)
        Store (FD13, Local3)
        And (Local3, 0x3F, Local3)
        Multiply (Local2, 0x40, Local2)
        Add (Local3, Local2, Local2)
        Store (Local2, FD13)
        Sleep (0x012C)
        Store (FAN3, Local0)
    }
}

If (LAnd (LEqual (Local0, 0xFF), LEqual (Local1, 0x07)))
{
    Return (Zero)
}

If (LAnd (LEqual (Local0, Zero), LEqual (Local1, Zero)))
{
    Return (0xFFFF)
}

Store (One, Local2)
While (Local1)
```

```
    {
        Multiply (Local2, 0x02, Local2)
        Decrement (Local1)
    }

    Multiply (Local0, Local2, Local0)
    Divide (0x00149970, Local0, Local1, Local0)
    Return (Local0)
}

OperationRegion (HWRE, SystemIO, IOHW, 0x0A)
Field (HWRE, ByteAcc, NoLock, Preserve)
{
    Offset (0x05),
    HIDX,   8,
    HDAT,   8
}

IndexField (HIDX, HDAT, ByteAcc, NoLock, Preserve)
{
    Offset (0x04),
    CHNM,   1,
    CFNM,   1,
    CHNS,   2,
    CFNS,   2,
    Offset (0x05),
    SYST,   8,
    TRGT,   8,
    Offset (0x08),
    SSDN,   8,
    CSDN,   8,
    SSUP,   8,
    CSUP,   8,
    Offset (0x20),
    VCOR,   8,
    V12V,   8,
    Offset (0x23),
    V33V,   8,
    Offset (0x25),
    V50V,   8,
    Offset (0x27),
    MBTE,   8,
    FAN1,   8,
    FAN2,   8,
    FAN3,   8,
    Offset (0x40),
    HWST,   1,
    Offset (0x41),
    Offset (0x47),
```

```
            ,   4,
        FD11,   2,
        FD12,   2,
        Offset (0x4B),
        FD13,   8,
        Offset (0x4E),
        BSEL,   3,
        Offset (0x4F),
        Offset (0x50),
        TSR1,   8,
        TSR2,   1,
        Offset (0x52),
        Offset (0x5D),
        FD21,   8
    }
}

Scope (\)
{
    Field (RAMW, ByteAcc, NoLock, Preserve)
    {
        Offset (0x20),
        CPUQ,   8,
        CPVL,   16,
        CPVH,   16,
        CPVC,   1
    }
}

Scope (_SB.PCI0.SBRG.ASOC)
{
    Mutex (AMTX, 0x00)
    Name (CORV, Package (0x05)
    {
        0x06020000,
        "Vcore Voltage",
        0x0352,
        0x0640,
        One
    })
    Name (V3VV, Package (0x05)
    {
        0x06020001,
        " +3.3 Voltage",
        0x0B9A,
        0x0E2E,
        One
    })
    Name (V5VV, Package (0x05)
```

```
{
    0x06020002,
    " +5 Voltage",
    0x1194,
    0x157C,
    One
})
Name (VV12, Package (0x05)
{
    0x06020003,
    " +12 Voltage",
    0x27D8,
    0x35E8,
    One
})
Name (VPAR, Package (0x04)
{
    Package (0x03)
    {
        Zero,
        One,
        Zero
    },

    Package (0x03)
    {
        0x22,
        0x22,
        Zero
    },

    Package (0x03)
    {
        0x16,
        0x0A,
        Zero
    },

    Package (0x03)
    {
        0x38,
        0x0A,
        Zero
    }
})
Name (VBUF, Package (0x05)
{
    0x04,
    CORV,
```

```
    V3VV,
    V5VV,
    VV12
})
Method (VGET, 1, NotSerialized)
{
    Store (Zero, Local0)
    If (LEqual (Arg0, Zero))
    {
        Store (^^SIOR.HWV0 (), Local0)
    }

    If (LEqual (Arg0, One))
    {
        Store (^^SIOR.HWV3 (), Local0)
    }

    If (LEqual (Arg0, 0x02))
    {
        Store (^^SIOR.HWV5 (), Local0)
    }

    If (LEqual (Arg0, 0x03))
    {
        Store (^^SIOR.HWV1 (), Local0)
    }

    Return (Local0)
}

Name (CPUT, Package (0x05)
{
    0x06030000,
    "CPU Temperature",
    0x0258,
    0x03B6,
    0x00010001
})
Name (MBTP, Package (0x05)
{
    0x06030001,
    "MB Temperature",
    0x01C2,
    0x03B6,
    0x00010001
})
Name (TBUF, Package (0x03)
{
    0x02,
```

```
    CPUT,
    MBTP
})
Method (TGET, 1, NotSerialized)
{
    Store (Zero, Local0)
    If (LEqual (Arg0, Zero))
    {
        Store (^^SIOR.HWT2 (), Local0)
    }

    If (LEqual (Arg0, One))
    {
        Store (^^SIOR.HWT0 (), Local0)
    }

    Return (Local0)
}

Name (CPUF, Package (0x05)
{
    0x06040000,
    "CPU FAN Speed",
    0x0258,
    0x1C20,
    0x00010001
})
Name (CHAF, Package (0x05)
{
    0x06040001,
    "CHASSIS FAN Speed",
    0x0258,
    0x1C20,
    0x00010001
})
Name (FBUF, Package (0x03)
{
    0x02,
    CPUF,
    CHAF
})
Method (FGET, 1, NotSerialized)
{
    Store (Zero, Local0)
    If (LEqual (Arg0, Zero))
    {
        Store (^^SIOR.HWF1 (), Local0)
    }
```

```
    If (LEqual (Arg0, One))
    {
        Store (^^SIOR.HWF0 (), Local0)
    }

    Return (Local0)
}

Name (QCFN, Package (0x06)
{
    0x04060003,
    "CPU Q-Fan Control",
    Zero,
    One,
    0x02,
    0x00010001
})
Name (QBUF, Package (0x02)
{
    One,
    QCFN
})
Method (VSIF, 0, Serialized)
{
    Return (VBUF)
}

Method (RVLT, 1, Serialized)
{
    Acquire (AMTX, 0xFFFF)
    And (Arg0, 0xFFFF, Local0)
    Store (VGET (Local0), Local1)
    Store (DerefOf (Index (DerefOf (Index (VPAR, Local0)), Zero)), Local2)
    Store (DerefOf (Index (DerefOf (Index (VPAR, Local0)), One)), Local3)
    Store (DerefOf (Index (DerefOf (Index (VPAR, Local0)), 0x02)), Local4)
    Multiply (Local1, Add (Local2, Local3), Local5)
    Divide (Local5, Local3, , Local5)
    Add (Local5, Local4, Local5)
    Release (AMTX)
    Return (Local5)
}

Method (SVLT, 1, Serialized)
{
    Acquire (AMTX, 0xFFFF)
    And (DerefOf (Index (Arg0, Zero)), 0xFFFF, Local0)
    Store (DerefOf (Index (VBUF, Zero)), Local1)
    If (LGreaterEqual (Local0, Local1))
    {
```

```
            Release (AMTX)
            Return (Zero)
        }

        Increment (Local0)
        Store (DerefOf (Index (Arg0, One)), Index (DerefOf (Index (VBUF, Local0)),
One))
        Store (DerefOf (Index (Arg0, 0x02)), Index (DerefOf (Index (VBUF, Local0)),
0x02))
        Store (DerefOf (Index (Arg0, 0x03)), Index (DerefOf (Index (VBUF, Local0)),
0x03))
        Store (DerefOf (Index (Arg0, 0x04)), Index (DerefOf (Index (VBUF, Local0)),
0x04))
        Release (AMTX)
        Return (One)
    }

    Method (TSIF, 0, Serialized)
    {
        Return (TBUF)
    }

    Method (RTMP, 1, NotSerialized)
    {
        Acquire (AMTX, 0xFFFF)
        And (Arg0, 0xFFFF, Local0)
        Store (TGET (Local0), Local1)
        Release (AMTX)
        Return (Local1)
    }

    Method (STMP, 1, Serialized)
    {
        Acquire (AMTX, 0xFFFF)
        Store (And (DerefOf (Index (Arg0, Zero)), 0xFFFF), Local0)
        Store (DerefOf (Index (TBUF, Zero)), Local1)
        If (LGreaterEqual (Local0, Local1))
        {
            Release (AMTX)
            Return (Zero)
        }

        Increment (Local0)
        Store (DerefOf (Index (Arg0, One)), Index (DerefOf (Index (TBUF, Local0)),
One))
        Store (DerefOf (Index (Arg0, 0x02)), Index (DerefOf (Index (TBUF, Local0)),
0x02))
        Store (DerefOf (Index (Arg0, 0x03)), Index (DerefOf (Index (TBUF, Local0)),
0x03))
```

```
            Store (DerefOf (Index (Arg0, 0x04)), Index (DerefOf (Index (TBUF, Local0)),
0x04))
            Release (AMTX)
            Return (One)
        }

        Method (FSIF, 0, Serialized)
        {
            Return (FBUF)
        }

        Method (RFAN, 1, Serialized)
        {
            Acquire (AMTX, 0xFFFF)
            And (Arg0, 0xFFFF, Local0)
            Store (FGET (Local0), Local1)
            Release (AMTX)
            Return (Local1)
        }

        Method (SFAN, 1, Serialized)
        {
            Acquire (AMTX, 0xFFFF)
            And (DerefOf (Index (Arg0, Zero)), 0xFFFF, Local0)
            Store (DerefOf (Index (FBUF, Zero)), Local1)
            If (LGreaterEqual (Local0, Local1))
            {
                Release (AMTX)
                Return (Zero)
            }

            Increment (Local0)
            Store (DerefOf (Index (Arg0, One)), Index (DerefOf (Index (FBUF, Local0)),
One))
            Store (DerefOf (Index (Arg0, 0x02)), Index (DerefOf (Index (FBUF, Local0)),
0x02))
            Store (DerefOf (Index (Arg0, 0x03)), Index (DerefOf (Index (FBUF, Local0)),
0x03))
            Store (DerefOf (Index (Arg0, 0x04)), Index (DerefOf (Index (FBUF, Local0)),
0x04))
            Release (AMTX)
            Return (One)
        }

        Method (QFIF, 0, NotSerialized)
        {
            If (LEqual (CPUQ, Zero))
            {
                And (DerefOf (Index (QCFN, 0x05)), 0xFFFDFFFF, Local0)
```

```
            Store (Local0, Index (QCFN, 0x05))
        }
        Else
        {
            Or (DerefOf (Index (QCFN, 0x05)), 0x00020000, Local0)
            Store (Local0, Index (QCFN, 0x05))
        }

        Return (QBUF)
    }

    Method (GCQV, 1, NotSerialized)
    {
        If (LEqual (Arg0, Zero))
        {
            Return (CPVL)
        }

        If (LEqual (Arg0, One))
        {
            Return (CPVH)
        }

        If (LEqual (Arg0, 0x02))
        {
            Return (CPVC)
        }

        Return (Zero)
    }

    Method (QFST, 1, NotSerialized)
    {
        If (LEqual (Arg0, DerefOf (Index (QCFN, Zero))))
        {
            Return (CQST)
        }

        Return (Zero)
    }
}

Scope (_SB.PCI0)
{
    Device (GFX0)
    {
        Name (_ADR, 0x00020000)  // _ADR: Address
        OperationRegion (IGDM, SystemMemory, 0xCFF8E0F4, 0x2000)
        Field (IGDM, AnyAcc, NoLock, Preserve)
```

```
{
    SIGN,   128,
    SIZE,   32,
    OVER,   32,
    SVER,   256,
    VVER,   128,
    GVER,   128,
    MBOX,   32,
    Offset (0xF0),
    IBTT,   4,
    IPSC,   2,
    IPAT,   4,
    IBIA,   3,
    IBLC,   2,
    Offset (0xF2),
    ITVF,   4,
    ITVM,   4,
    IDVM,   1,
    IDVS,   2,
    ISSC,   1,
    Offset (0xF4),
    Offset (0x100),
    DRDY,   32,
    CSTS,   32,
    CEVT,   32,
    Offset (0x120),
    DIDL,   32,
    DDL2,   32,
    DDL3,   32,
    DDL4,   32,
    DDL5,   32,
    DDL6,   32,
    DDL7,   32,
    DDL8,   32,
    CPDL,   32,
    CPL2,   32,
    CPL3,   32,
    CPL4,   32,
    CPL5,   32,
    CPL6,   32,
    CPL7,   32,
    CPL8,   32,
    CADL,   32,
    CAL2,   32,
    CAL3,   32,
    CAL4,   32,
    CAL5,   32,
    CAL6,   32,
    CAL7,   32,
```

```
    CAL8,   32,
    NADL,   32,
    NDL2,   32,
    NDL3,   32,
    NDL4,   32,
    NDL5,   32,
    NDL6,   32,
    NDL7,   32,
    NDL8,   32,
    ASLP,   32,
    TIDX,   32,
    CHPD,   32,
    CLID,   32,
    CDCK,   32,
    SXSW,   32,
    EVTS,   32,
    CNOT,   32,
    NRDY,   32,
    Offset (0x200),
    SCIE,   1,
    GEFC,   4,
    GXFC,   3,
    GESF,   8,
    Offset (0x204),
    PARM,   32,
    DSLP,   32,
    Offset (0x300),
    ARDY,   32,
    ASLC,   32,
    TCHE,   32,
    ALSI,   32,
    BCLP,   32,
    PFIT,   32,
    Offset (0x400),
    GVD1,   57344
}

OperationRegion (TCOI, SystemIO, TOBS, 0x08)
Field (TCOI, WordAcc, NoLock, Preserve)
{
    Offset (0x04),
        ,   9,
    SCIS,   1,
    Offset (0x06)
}

Name (DBTB, Package (0x15)
{
    Zero,
```

```
    0x07,
    0x38,
    0x01C0,
    0x0E00,
    0x3F,
    0x01C7,
    0x0E07,
    0x01F8,
    0x0E38,
    0x0FC0,
    Zero,
    Zero,
    Zero,
    Zero,
    Zero,
    0x7000,
    0x7007,
    0x7038,
    0x71C0,
    0x7E00
})
Method (GSCI, 0, NotSerialized)
{
    If (LEqual (GEFC, 0x04))
    {
        Store (GBDA (), GXFC)
    }

    If (LEqual (GEFC, 0x06))
    {
        Store (SBCB (), GXFC)
    }

    Store (One, SCIS)
    Store (Zero, GEFC)
    Store (Zero, GSSE)
    Store (Zero, SCIE)
    Return (Zero)
}

Method (GBDA, 0, NotSerialized)
{
    If (LEqual (GESF, Zero))
    {
        Store (0x0279, PARM)
        Store (Zero, GESF)
        Return (SUCC)
    }
```

```
If (LEqual (GESF, One))
{
   Store (0x20, PARM)
   Store (Zero, GESF)
   Return (SUCC)
}

If (LEqual (GESF, 0x04))
{
   And (PARM, 0xEFFF0000, PARM)
   And (PARM, ShiftLeft (DerefOf (Index (DBTB, IBTT)), 0x10), PARM)
   Or (IBTT, PARM, PARM)
   Store (Zero, GESF)
   Return (SUCC)
}

If (LEqual (GESF, 0x05))
{
   Store (IPSC, PARM)
   Or (PARM, ShiftLeft (IPAT, 0x08), PARM)
   Add (PARM, 0x0100, PARM)
   Or (PARM, ShiftLeft (LIDS, 0x10), PARM)
   XOr (PARM, 0x00010000, PARM)
   Or (PARM, ShiftLeft (IBIA, 0x14), PARM)
   Store (Zero, GESF)
   Return (SUCC)
}

If (LEqual (GESF, 0x06))
{
   Store (ITVF, PARM)
   Or (PARM, ShiftLeft (ITVM, 0x04), PARM)
   Store (Zero, GESF)
   Return (SUCC)
}

If (LEqual (GESF, 0x07))
{
   Name (MEMS, 0x0D)
   Store (GIVD, PARM)
   XOr (PARM, One, PARM)
   Or (PARM, ShiftLeft (GMFN, One), PARM)
   Or (PARM, 0x1800, PARM)
   Or (ShiftLeft (CDCT, 0x15), PARM, PARM)
   If (LEqual (IDVM, One))
   {
      Store (0x11, MEMS)
   }
```

```
            If (LLess (TASM, M512))
            {
                Or (PARM, ShiftLeft (One, MEMS), PARM)
            }
            ElseIf (LLess (TASM, M1GB))
            {
                If (LLess (IDVS, 0x03))
                {
                    Or (PARM, ShiftLeft (IDVS, MEMS), PARM)
                }
                Else
                {
                    Or (PARM, ShiftLeft (0x02, MEMS), PARM)
                }
            }
            Else
            {
                Or (PARM, ShiftLeft (IDVS, MEMS), PARM)
            }

            Store (One, GESF)
            Return (SUCC)
        }

        If (LEqual (GESF, 0x0A))
        {
            Store (Zero, PARM)
            If (ISSC)
            {
                Or (PARM, 0x03, PARM)
            }

            Store (Zero, GESF)
            Return (SUCC)
        }

        Store (Zero, GESF)
        Return (CRIT)
    }

    Method (SBCB, 0, NotSerialized)
    {
        If (LEqual (GESF, Zero))
        {
            Store (0x20, PARM)
            Store (Zero, GESF)
            Return (SUCC)
        }
```

```
If (LEqual (GESF, One))
{
    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}

If (LEqual (GESF, 0x03))
{
    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}

If (LEqual (GESF, 0x04))
{
    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}

If (LEqual (GESF, 0x05))
{
    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}

If (LEqual (GESF, 0x06))
{
    ShiftRight (PARM, 0x1C, Local0)
    If (LEqual (Local0, Zero))
    {
        And (PARM, 0x0F, ITVF)
        And (PARM, 0xF0, ITVM)
    }

    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}

If (LEqual (GESF, 0x07))
{
    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}
```

```
If (LEqual (GESF, 0x08))
{
   Store (Zero, GESF)
   Store (Zero, PARM)
   Return (SUCC)
}

If (LEqual (GESF, 0x09))
{
   And (PARM, 0xFF, IBTT)
   Store (Zero, GESF)
   Store (Zero, PARM)
   Return (SUCC)
}

If (LEqual (GESF, 0x0A))
{
   ShiftRight (PARM, 0x1C, Local0)
   If (LEqual (Local0, Zero))
   {
      And (PARM, 0xFF, IPSC)
      Subtract (And (ShiftRight (PARM, 0x08), 0xFF), One, IPAT)
      And (ShiftRight (PARM, 0x12), 0x03, IBLC)
      And (ShiftRight (PARM, 0x14), 0x07, IBIA)
   }

   Store (Zero, GESF)
   Store (Zero, PARM)
   Return (SUCC)
}

If (LEqual (GESF, 0x0B))
{
   If (LEqual (And (ShiftRight (PARM, 0x0B), 0x03), 0x02))
   {
      And (ShiftRight (PARM, 0x0D), 0x0F, Local0)
      And (ShiftRight (PARM, 0x11), 0x0F, Local1)
      If (Local0)
      {
         Store (Zero, IDVM)
         Store (Local0, IDVS)
      }

      If (Local1)
      {
         Store (One, IDVM)
         Store (Local1, IDVS)
      }
   }
```

```
    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}

If (LEqual (GESF, 0x10))
{
    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}

If (LEqual (GESF, 0x11))
{
    Store (ShiftLeft (LIDS, 0x08), PARM)
    Store (Zero, GESF)
    Return (SUCC)
}

If (LEqual (GESF, 0x12))
{
    If (And (PARM, One))
    {
        If (LEqual (ShiftRight (PARM, One), One))
        {
            Store (One, ISSC)
        }
        Else
        {
            Store (Zero, GESF)
            Return (CRIT)
        }
    }
    Else
    {
        Store (Zero, ISSC)
    }

    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
}

If (LEqual (GESF, 0x13))
{
    Store (Zero, GESF)
    Store (Zero, PARM)
    Return (SUCC)
```

```
    }

    Store (Zero, GESF)
    Return (SUCC)
}

Scope (^^PCI0)
{
    OperationRegion (MCHP, PCI_Config, 0x40, 0xC0)
    Field (MCHP, AnyAcc, NoLock, Preserve)
    {
        Offset (0x60),
        TASM,   10,
        Offset (0x62)
    }
}

OperationRegion (IGDP, PCI_Config, 0x40, 0xC0)
Field (IGDP, AnyAcc, NoLock, Preserve)
{
    Offset (0x12),
        ,   1,
    GIVD,   1,
        ,   2,
    GUMA,   3,
    Offset (0x14),
        ,   4,
    GMFN,   1,
    Offset (0x18),
    Offset (0x8C),
    CDCT,   10,
    Offset (0x8E),
    Offset (0xA8),
    GSSE,   1,
    GSSB,   14,
    GSES,   1,
    Offset (0xBC),
    ASLS,   32
}

Name (M512, 0x08)
Name (M1GB, 0x10)
Scope (\_GPE)
{
    Method (_L06, 0, NotSerialized)  // _Lxx: Level-Triggered GPE
    {
        \_SB.PCI0.GFX0.GSCI ()
    }
}
```

```
        Name (OPBS, 0xFFFFFF00)
        Method (OPTS, 1, NotSerialized)
        {
            If (LEqual (Arg0, 0x03))
            {
                Store (ASLS, OPBS)
            }
        }

        Method (OWAK, 1, NotSerialized)
        {
            If (LEqual (Arg0, 0x03))
            {
                Store (OPBS, ASLS)
                Store (One, GSES)
            }
        }
    }
}

Scope (_SB)
{
    Scope (PCI0)
    {
        Name (CRS, ResourceTemplate ()
        {
            WordBusNumber (ResourceProducer, MinFixed, MaxFixed, PosDecode,
                0x0000,             // Granularity
                0x0000,             // Range Minimum
                0x00FF,             // Range Maximum
                0x0000,             // Translation Offset
                0x0100,             // Length
                ,, )
            IO (Decode16,
                0x0CF8,             // Range Minimum
                0x0CF8,             // Range Maximum
                0x01,               // Alignment
                0x08,               // Length
                )
            WordIO (ResourceProducer, MinFixed, MaxFixed, PosDecode,
EntireRange,
                0x0000,             // Granularity
                0x0000,             // Range Minimum
                0x0CF7,             // Range Maximum
                0x0000,             // Translation Offset
                0x0CF8,             // Length
                ,, , TypeStatic)
            WordIO (ResourceProducer, MinFixed, MaxFixed, PosDecode,
```

```
EntireRange,
                0x0000,             // Granularity
                0x0D00,             // Range Minimum
                0xFFFF,             // Range Maximum
                0x0000,             // Translation Offset
                0xF300,             // Length
                ,, , TypeStatic)
            DWordMemory (ResourceProducer, PosDecode, MinFixed, MaxFixed,
Cacheable, ReadWrite,
                0x00000000,         // Granularity
                0x000A0000,         // Range Minimum
                0x000BFFFF,         // Range Maximum
                0x00000000,         // Translation Offset
                0x00020000,         // Length
                ,, , AddressRangeMemory, TypeStatic)
            DWordMemory (ResourceProducer, PosDecode, MinFixed, MaxFixed,
Cacheable, ReadWrite,
                0x00000000,         // Granularity
                0x000C0000,         // Range Minimum
                0x000DFFFF,         // Range Maximum
                0x00000000,         // Translation Offset
                0x00020000,         // Length
                ,, _Y16, AddressRangeMemory, TypeStatic)
            DWordMemory (ResourceProducer, PosDecode, MinFixed, MaxFixed,
Cacheable, ReadWrite,
                0x00000000,         // Granularity
                0x00000000,         // Range Minimum
                0x00000000,         // Range Maximum
                0x00000000,         // Translation Offset
                0x00000000,         // Length
                ,, _Y17, AddressRangeMemory, TypeStatic)
            DWordMemory (ResourceProducer, PosDecode, MinFixed, MaxFixed,
Cacheable, ReadWrite,
                0x00000000,         // Granularity
                0x00000000,         // Range Minimum
                0x00000000,         // Range Maximum
                0x00000000,         // Translation Offset
                0x00000000,         // Length
                ,, _Y18, AddressRangeMemory, TypeStatic)
        })
        CreateDWordField (CRS, \_SB.PCI0._Y16._MIN, MIN5)  // _MIN: Minimum
Base Address
        CreateDWordField (CRS, \_SB.PCI0._Y16._MAX, MAX5)  // _MAX:
Maximum Base Address
        CreateDWordField (CRS, \_SB.PCI0._Y16._LEN, LEN5)  // _LEN: Length
        CreateDWordField (CRS, \_SB.PCI0._Y17._MIN, MIN6)  // _MIN: Minimum
Base Address
        CreateDWordField (CRS, \_SB.PCI0._Y17._MAX, MAX6)  // _MAX:
Maximum Base Address
```

```
            CreateDWordField (CRS, \_SB.PCI0._Y17._LEN, LEN6)  // _LEN: Length
            CreateDWordField (CRS, \_SB.PCI0._Y18._MIN, MIN7)  // _MIN: Minimum
Base Address
            CreateDWordField (CRS, \_SB.PCI0._Y18._MAX, MAX7)  // _MAX:
Maximum Base Address
            CreateDWordField (CRS, \_SB.PCI0._Y18._LEN, LEN7)  // _LEN: Length
            Method (_CRS, 0, NotSerialized)  // _CRS: Current Resource Settings
            {
                Store (MG1L, Local0)
                If (Local0)
                {
                    Store (MG1B, MIN5)
                    Store (MG1L, LEN5)
                    Add (MIN5, Decrement (Local0), MAX5)
                }

                Store (MG2B, MIN6)
                Store (MG2L, LEN6)
                Store (MG2L, Local0)
                Add (MIN6, Decrement (Local0), MAX6)
                Store (MG3B, MIN7)
                Store (MG3L, LEN7)
                Store (MG3L, Local0)
                Add (MIN7, Decrement (Local0), MAX7)
                Return (CRS)
            }
        }
    }

    Name (WOTB, Zero)
    Name (WSSB, Zero)
    Name (WAXB, Zero)
    Method (_PTS, 1, NotSerialized)  // _PTS: Prepare To Sleep
    {
        Store (Arg0, DBG8)
        PTS (Arg0)
        Store (Zero, Index (WAKP, Zero))
        Store (Zero, Index (WAKP, One))
        If (LAnd (LEqual (Arg0, 0x04), LEqual (OSFL (), 0x02)))
        {
            Sleep (0x0BB8)
        }

        Store (ASSB, WSSB)
        Store (AOTB, WOTB)
        Store (AAXB, WAXB)
        Store (Arg0, ASSB)
        Store (OSFL (), AOTB)
        Store (Zero, AAXB)
```

```
}

Method (_WAK, 1, NotSerialized)  // _WAK: Wake
{
   ShiftLeft (Arg0, 0x04, DBG8)
   WAK (Arg0)
   If (IOWK) {}
   Else
   {
      Notify (\_SB.PWRB, 0x02)
   }

   If (ASSB)
   {
      Store (WSSB, ASSB)
      Store (WOTB, AOTB)
      Store (WAXB, AAXB)
   }

   If (DerefOf (Index (WAKP, Zero)))
   {
      Store (Zero, Index (WAKP, One))
   }
   Else
   {
      Store (Arg0, Index (WAKP, One))
   }

   Return (WAKP)
}

OperationRegion (IORK, SystemIO, 0xB3, One)
Field (IORK, ByteAcc, NoLock, Preserve)
{
   IOWK,   8
}

Name (_S0, Package (0x04)  // _S0_: S0 System State
{
   Zero,
   Zero,
   Zero,
   Zero
})
If (SS1)
{
   Name (_S1, Package (0x04)  // _S1_: S1 System State
   {
      One,
```

```
        Zero,
        Zero,
        Zero
    })
}

If (SS3)
{
    Name (_S3, Package (0x04)  // _S3_: S3 System State
    {
        0x05,
        Zero,
        Zero,
        Zero
    })
}

If (SS4)
{
    Name (_S4, Package (0x04)  // _S4_: S4 System State
    {
        0x06,
        Zero,
        Zero,
        Zero
    })
}

Name (_S5, Package (0x04)  // _S5_: S5 System State
{
    0x07,
    Zero,
    Zero,
    Zero
})
Method (PTS, 1, NotSerialized)
{
    If (Arg0)
    {
        \_SB.PCI0.SBRG.SIOS (Arg0)
        \_SB.PCI0.NPTS (Arg0)
        \_SB.PCI0.SBRG.SPTS (Arg0)
        \_SB.PCI0.GFX0.OPTS (Arg0)
    }
}

Method (WAK, 1, NotSerialized)
{
    \_SB.PCI0.SBRG.SIOW (Arg0)
```

```
        \_SB.PCI0.NWAK (Arg0)
        \_SB.PCI0.SBRG.SWAK (Arg0)
        \_SB.PCI0.GFX0.OWAK (Arg0)
        \_SB.PCI0.SBRG.ASOC.GWS3 (Arg0)
    }
}
```