Fachhochschule
Dortmund
University of Applied Sciences and Arts

Bachelor-Thesis

Konfiguration einer GNU RISC-V Toolchain für die Programmierung eines Sipeed Longan Nano Mikrocontroller-Boards in der Eclipse IDE

Fachhochschule Dortmund

Fachbereich Elektrotechnik

Verfasser: Mehmet Kiyak

Erstprüfer: Prof. Dr.-Ing. Micheal Karagounis

Zweitprüfer: Dipl.-Ing. Rolf Paulus

Abgabedatum: 21. Dezember 2021

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen herzlich bedanken, die mich während der Anfertigung meiner Bachelorarbeit unterstützt und motiviert haben.

Zunächst gebührt mein Dank Herrn Prof. Dr.-Ing. Micheal Karagounis, der meine Bachelorarbeit betreut und mit seinem Engagement bereichert hat.

Für die hilfreichen Anregungen und das konstruktive Feedback möchte ich mich herzlich bedanken. Mein Dank gilt auch der Unterstützung von Herrn Dipl.-Ing. Rolf Paulus.

Des Weiteren möchte ich mich bei meinen Eltern und Geschwistern bedanken. Vielen Dank für die mentale Unterstützung und den Beistand während des Anfertigens meiner Bachelorarbeit und meines gesamten Studiums.

Vielen lieben Dank

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Konfiguration der GNU RISC-V Toolchain für die erste Programmierung des Entwicklungsboards Sipeed Longan Nano in der Eclipse Entwicklungsumgebung.

In diesem Zusammenhang wurde der Aufbau eines Mikrocontrollers, der Ablauf der Erstellung von Software und die Konfiguration der GNU RISC-V Toolchain für die Programmierung der RGB LED des Entwicklungsboards beschrieben. Dazu gehört das Linker-Script, die Vektortabelle und der Startcode.

Abstract

This thesis focuses on the configuration of the GNU RISC-V toolchain for the first programming of the development board Sipeed Longan Nano in the Eclipse development environment.

In this context, the structure of a microcontroller, the process of creating a software and the configuration of the GNU RISC-V toolchain for programming the RGB LED of the development board were described. This includes the linker script, the vector table and the start code.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass die von mir vorgelegte Prüfungsleistung selbstständig und ohne unzulässige fremde Hilfe erstellt worden ist. Alle verwendeten Quellen sind in der Arbeit so aufgeführt, dass Art und Umfang der Verwendung nachvollziehbar sind.

Dortmund, den 21. Dezember 2021 _____

Mehmet Kiyak

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ABI ..... *application binary interface*
ADC ..... *Analog to digital converter*
ALU ..... *Arithmetic Logic Unit*
bit ..... Binary digit
Bus ..... *Binary Unit System*
CAN ..... *Controller area network*
CDT ..... *C Development Tool*
CISC ..... *Complex Instruction Set Computer*
CLI ..... *Command Line Interface*
COFF ..... *Common Object File Format*
CPU ..... *Central Processing Unit*
CSR ..... *Control and Status Register*
CU ..... *Control Unit*
DAC ..... *Digital to analog converter*
DFU ..... *Data File Utility*
ELF ..... *Executable and Linkable Format*
GCC ..... *GNU C Compiler*
I2C ..... *Inter-Integrated Circuit*
I2S ..... *Inter-IC Sound*
IDE ..... *Integrated Development Environment*
IPS ..... *In-plane switching*
ISA ..... *Instruction Set Architecture*
ISP ..... *In System Programming*
JRE ..... *Java Runtime Environment*
JTAG ..... *Joint Test Action Group*
LCD ..... *Liquid Crystal Display*
LIFO ..... *Last-In-First-Out*
LTS ..... *Long Term Support*
MIE ..... *Machine-mode Interrupts Enabled*
MISO ..... *Master Input, Slave Output*
MOSI ..... *Master Output, Slave Input*
MSPS ..... *Mega-Samples Per Second*
ODR ..... *Output Data Register*
OpenOCD ..... Open On-Chip Debugger
OTG ..... On-The-Go
PC ..... *Personal Computer*
PWM ..... *Pulse width modulation*
RAM ..... *Random Access Memory*
RGB ..... *Red, Green, Blue*
RISC ..... *Reduced Instruction Set Computer*
ROM ..... *Read Only Memory*
RTC ..... *Real Time Clock*
SCK ..... *Serial Clock*
SD-Karte ..... *Secure Digital Memory Card*
sp ..... *Stack-Pointer*
SPI ..... *Serial Peripheral Interface*
SS ..... *Slave Select Line*

Systick ..... *System Tick Timer*
UART ..... *Universal Asynchronous Receiver Transmitter*
USART..... *Universal Synchronous Asynchronous Receiver Transmitter*
USB .....*Universal Serial Bus*
USBFS..... *Universal Serial Bus full speed*

1 Einleitung

In der heutigen Zeit sind Mikrocontroller unverzichtbare Komponenten für die Steuerung technischer Gebrauchsgegenstände wie z.B. Waschmaschinen, Mikrowellen, aber auch in der Unterhaltungselektronik. So finden sie beispielsweise bei Fernsehern und Fernbedienungen breite Anwendung. Für die jeweiligen Aufgaben der Geräte werden Mikrocontroller speziell technisch angepasst und programmiert. Im Vergleich zu den im Computer eingesetzten Prozessoren sind Mikrocontroller nur für die Erfüllung einer speziellen Aufgabe ausgelegt und müssen somit keine aufwendige Hardware für weitere Funktionen vorhalten. Somit sind die Kosten für die Produktion von Mikrocontrollern geringer als bei den Computer-Prozessoren [1].

Diese Arbeit beschäftigt sich mit der Programmierung eines RISC-V Mikrocontrollers. Wie der Name schon andeutet, handelt es sich bei diesem Mikrocontroller um eine RISC (Reduced Instruction Set Computer) Architektur. RISC Systeme zeichnen sich durch wenige und einfache Instruktionen aus, besitzen dafür aber mehr Register als beispielsweise CISC (Complex Instruction Set Computer) [2]. Bei RISC-V Mikrocontrollern handelt es sich um eine sehr junge Computerarchitektur, die aber deswegen nicht weniger leistungsstark als bereits etablierte Computerarchitekturen ist. Die tatsächliche Leistungsfähigkeit des Mikrocontrollers hängt von der jeweiligen Qualität der mikroarchitektonischen Umsetzung, des Schaltungsdesigns und der Prozesstechnologie ab [3].

Im Folgenden wird die Konfiguration der GNU RISC-V Toolchain in Eclipse, eines Entwicklungsboards mit einem RISC-V Mikrocontroller der GD32VF103 Serie von Giga Device, als Ziel meiner Bachelorarbeit durchgeführt.

Die Entwicklungsumgebung von Eclipse wird mit den Windows-Build-Tools, OpenOCD (Open On-Chip Debugger [4]) und der entsprechenden RISC-V Toolchain installiert. Dazu sind einige Schritte beginnend von der Installation der Software bis zur eigentlichen Programmierung des Entwicklungsboards Sipeed Longan Nano erforderlich. Im Rahmen dieser Arbeit werden nachfolgend die Installationen der zugehörigen Software und die Konfiguration der Toolchain, sowie die erste Programmierung des Entwicklungsboards beschrieben.

2 Die RISC-V Architektur

Die Befehlssatzarchitektur RISC-V ist offen und wurde an der University of California in Berkeley unter anderem von Andrew Watermann, Yunsup Lee und Prof. Krste Asanovic entwickelt, welche zusammen im Mai 2010 das RISC-V-Projekt gründeten [5]. Der offene Standard der RISC-Architektur soll eigene Softwareportierungen, sowie eigene Hardwareentwicklungen möglich machen, um damit verschiedene Aufgaben zu lösen. Das verwendete Lizenzmodell erlaubt auch, dass eigens erstellte Software nicht freigegeben werden muss. Der Mikrocontroller soll mannigfache Aufgaben lösen und zugleich frei oder geschlossen bleiben können.

Aus dem akademischen RISC-V Projekt heraus wurde die RISC-V Foundation im Jahr 2015 gegründet. Die Zusammenarbeit von über 750 Mitgliedern wie z.B. Western Digital, SiFive und Samsung soll zur weiteren Verbreitung der Architektur beitragen [6]. Die Mitgliedschaft ist kostenpflichtig und in die folgenden Kategorien gruppiert: Premier-, Strategic- und Community-Mitglieder. Durch die kostenpflichtige Mitgliedschaft erhält man ein Mitspracherecht bei der Vermarktung und der Betreuung von technischen und operativen Vorgängen. Während die Befehlssatzarchitektur (engl. Instruction Set Architecture, ISA) offen ist und kostenlos für eigene Implementierungen genutzt werden darf, wird für die Verwendung der Marke RISC-V und des Logos der RISC-V Foundation eine Lizenz Instruction SA, die in der Mitgliedschaft beinhaltet ist, benötigt [7, 8].

Ein wichtiges Ziel der RISC-V Foundation bzw. RISC-V International besteht darin, einen geringen Energieverbrauch bei eingebetteten Anwendungen zu erzielen und mithilfe der RISC-V Architektur effiziente Mikroarchietekturen zu entwerfen, welche die dynamische Verzweigungsvorhersage beherrschen, Befehlsvorabrufpufferung nutzen und lokale Caches verwenden [5].

2.1 RISC und CISC

Die Entwicklung von CISC begann zwischen den 60er und 70er Jahren. Die Prozessorarchitektur wurde wegen der damaligen teuren und langsamen Hauptspeicher, die als Zwischenspeicher aufgrund fehlender Cache-Speicher dienten, entwickelt. CISC hatte das Problem der langsamen Speicher durch viele Anweisungen in einem Maschinenbefehl gelöst. Durch die vielen Anweisungen, die mit einem Maschinenbefehl aus dem Hauptspeicher übertragen wurden,

konnten viele Vorgänge den Prozessor beschäftigen. So wurde der langsame Speicher überwunden.

Um die Maschinenbefehle zu steuern, also um die benötigen Maschinenbefehle auszuführen, wurden Mikroprogramme eingesetzt. Die Ausführung der komplexen Maschinenbefehle wurde durch mehrere hintereinander laufende Mikroprogramme gesteuert.

Dieses Steuerwerk brachte jedoch einige Nachteile mit sich. Es benötigte spezielle Speicher, in dem die Mikroprogramme des Prozessors abgelegt wurden. Die Folge war eine größere Chip-Fläche und somit höhere Kosten bei der Herstellung.

Die CISC-Architektur zeichnete sich durch folgende Eigenschaften aus [9, S. 34]:

- Umfangreiche Befehlssätze
- Viele Adressierungsmöglichkeiten
- Spezialisierte Register
- Viele Operationen mit einem Maschinenbefehl

Ende der 70er Jahre wurde die RISC-Architektur entwickelt. Unter anderem bestand der Grund für die Entwicklung von RISC darin, dass einige der Anweisungen im Maschinenbefehl der CISC-Architektur wenig genutzt wurden. Die Mikroprogrammierung benötigte zudem Chip-Fläche und beeinflusste somit die Kosten in der Herstellung und die Effizienz des Prozessors. Die vielen Adressierungsmöglichkeiten bzw. Befehlsformate und -längen belasteten außerdem den Compiler. Die Idee war, die vielen Anweisungen oder Operationen durch mehrere kleinere Maschinenbefehle zu ersetzen, also vereinfachte Befehlssätze zu erzielen. Die Nebeneffekte waren geringere Kosten in der Herstellung aufgrund des kleineren Steuerwerkes und somit auch in der Effizienz des Prozessors [9, S. 35-36].

Durch das Pipelining-Prinzip wurden die Maschinenbefehle durch Überlagerung der Anweisungen bzw. Operationen abgearbeitet [9, S. 45-46]. Bei RISC waren somit weniger Anweisungen in einem Maschinenbefehl, die zu einer höheren Effizienz bei der Ausführung in einer Pipeline gegenüber den umfangreichen Befehlssätzen der CISC-Architektur führten.

Die RISC-Architektur zeichnet sich durch folgende Eigenschaften aus [9, S. 35-36]:

- Vereinfachte Maschinenbefehle, also weniger Anweisungen pro Befehl
- Kleineres Steuerwerk, da keine Mikroprogrammierung
- Effizienteres Pipelining-Prinzip
- Günstiger in der Herstellung als bei älteren Prozessorarchitekturen wie CISC

Die erste sogenannte 32-Bit-RISC-Architektur hatte keine Gleitkommaarithmetik und Betriebssystemfunktionen. Der Befehlssatz bei der RISC-Architektur aus Berkeley bestand aus 31 Befehlen mit einer Länge von je 32-Bit mit zwei Befehlsformaten, wobei der Registersatz 78 Register groß war. Dieser Registersatz ist durch eine Fensterstruktur bekannt, die zeitlich jeweils einen Teil der Registerdatei ausblendet. Die Anwendung dieser Fensterstruktur führt zu einer günstigen Übergabe von Parametern bei einer geringen Menge an kleinen Unterprogrammaufrufen [9, S. 37-38]. Die Fenster stellen eine effiziente Organisationsform für prozessorientierte Register dar, mit dem Multithreading ohne das erneute Laden der Registerinhalte aus dem Stack möglich ist. Das Registerfenster kann dabei bei Gebrauch des aktuellen Prozesses geschaltet werden [9, S. 365-369].

Der erste Prototyp in 2µm NMOS Technologie wurde im Oktobers 1982 gefertigt. Der Prozessor bestand aus 44 500 Transistoren. Die Prozessorsteuerung hat dabei einen Anteil von 6% der Chip-Fläche bei der ersten RISC-Architektur eingenommen und war damit wesentlich geringer als bei CISC-Architekturen, bei denen das Steuerwerk einen Flächenanteil von bis zu 50% erreicht hatten. Die Leistungsfähigkeit kommerzieller Mikroprozessoren wurde durch einen RISC-Chip der ersten Generation RISC I mit einer Taktfrequenz von 1,5MHz erreicht. 1983 wurde die zweite Generation RISC II der RISC-Architektur zunächst in der 2µm NMOS Technologie gefertigt und umfasste 41 000 Transistoren. Hier wurde der Anteil der Chip-Fläche um 25% verringert. Die erste und zweite Generation der RISC-Architekturen waren unvollendet und wurden im Rahmen weiterer RISC-Architekturen wie Entwürfen der SPARC-Architektur Mitte der 80er von der Firma Sun weiterentwickelt [9, S. 37-38].

3 Der Mikroprozessor

In diesem Kapitel wird für das bessere Verständnis der RISC-V Architektur der grundlegende Aufbau eines Mikroprozessors und Mikrocontrollers erklärt und eine Klärung der Begriffe Mikrocontroller, Mikroprozessor, Mikroprozessorsystem, Mikrorechner, Mikrocomputer, Mikrorechnersystem und Mikrocomputersystem durchgeführt.

3.1 Aufbau des Mikroprozessors und Mikrocontrollers

Der Mikroprozessor ist die zentrale Prozessoreinheit, kurz CPU (Central Processing Unit), einer Datenverarbeitungsanlage mit der Aufgabe Maschinenbefehle umzusetzen. Der Prozessorkern besteht aus einem Rechenwerk und einem Steuerwerk. Zusätzlich sind je nach Funktions- und Leistungsumfang des Mikroprozessors weitere Komponenten wie z.B. Cache-Speicher und virtuelle Speicherverwaltung verfügbar [9, S. 1].

Die zentrale Prozessoreinheit hat die Aufgabe [10, S. 4]:

- Komponenten, wie Speicher, Schnittstellen, Ein-/Ausgabeschnittstellen usw. zu steuern,
- Daten, also Bitmuster mit logischen und arithmetischen Operationen zu verarbeiten.

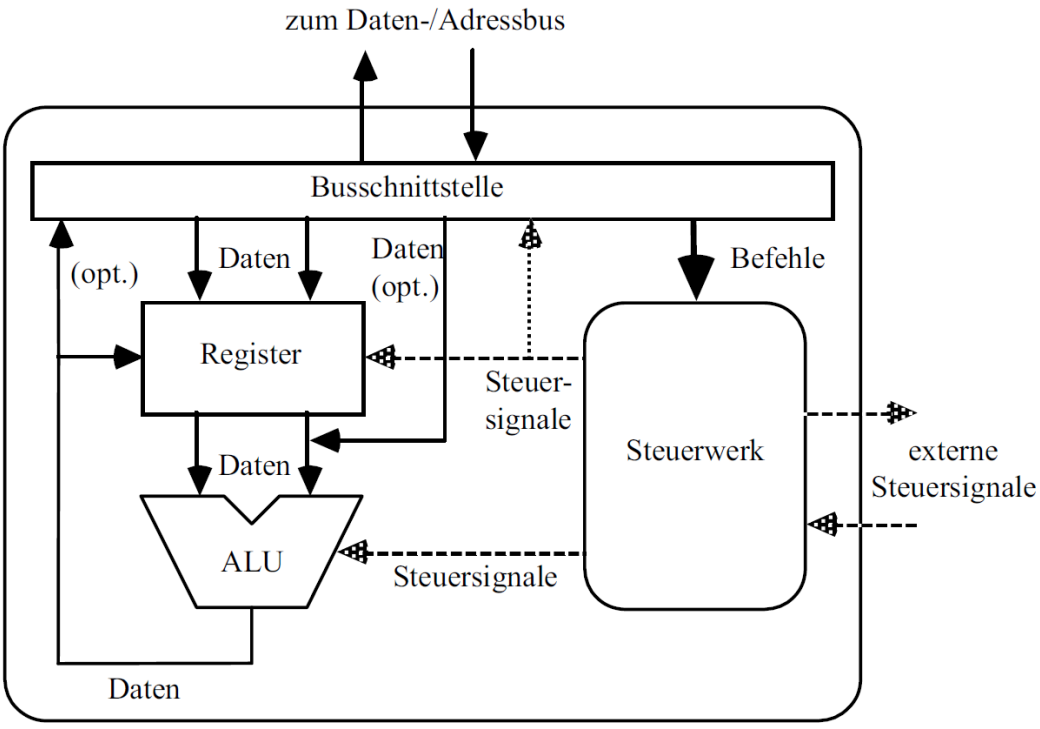

Abbildung 1: Aufbau eines Mikroprozessors [9, S. 44]

Der grundlegende Aufbau eines einfachen Mikroprozessors wird in der Abbildung 1 dargestellt. Diese Darstellung findet häufig bei Mikrocontrollern Anwendung [9, S. 44]. Der Mikrocontroller und der Mikroprozessor beinhalten einige entscheidende Elemente und folgen einem speziellen Aufbau im Kern des Prozessors. Dieser wird durch die Prozessorarchitektur definiert. Zu den Elementen des Prozessors gehören (siehe auch Abbildung 1) [9, S. 44-45]:

- das Steuerwerk,
- die ALU (Arithmetische-logische Einheit oder Arithmetic Logic Unit) oder auch Rechenwerk genannt,
- ein interner Adress-, Daten- und Steuerbus, sowie
- der Registersatz.

Der grundlegende Aufbau eines Mikroprozessors und Mikrocontrollers wird im weiteren Verlauf dieses Kapitels mit Anlehnung an die von-Neumann-Architektur beschrieben. Unter der sogenannten von-Neumann-Architektur ist der in den vierziger Jahren von John von Neumann entwickelte Aufbau eines grundlegenden und universellen Rechners zu verstehen [11, S. 39]. Sie bildet die Grundlage für heutige moderne Computer durch einen gemeinsamen Speicher von Daten und Programmbefehlen. Die Struktur setzt sich dabei aus folgenden Komponenten zusammen [9, S. 42]:

- *Das Steuerwerk*

  Die Regelung des Ablaufes im Rechner, die Steuerung der Komponenten des Rechners aus den Anweisungen der Software und die Interpretation dieser Anweisungen oder Befehle werden vom Steuerwerk übernommen [12, S. 144-145]. Das Steuerwerk ist für die Regelung des Ablaufes im Inneren des Prozessors und im verbleibenden System zuständig [10, S. 89]. Es wird auch Leitwerk, und Befehlswerk oder Control Unit (CU) bezeichnet [11, S. 40]. Der Ablauf der Verarbeitung einer Anweisung, die sich im Speicher befindet, beginnt mit der Übertragung der Anweisung aus dem Speicher in das Steuerwerk. Im Speicher befinden sich die Anweisungen eines Programms hintereinander angeordnet für die weitere Bearbeitung [12, S. 144-145]. Die Befehle werden im Steuerwerk interpretiert und die zuständigen Komponenten des Rechners aktiviert [11, S. 40]. Die Anweisung oder der Befehl befindet sich an einer Adresse im

Speicher. Diese Adresse wird im Befehlsregister (Instruction Register) [11, S. 41] bzw. dem Befehls- oder Programmzähler (Programm Counter, Instruction Pointer [11, S. 144]) im Steuerwerk abgelegt [12, S. 144-145]. Die Anweisung wird im Steuerwerk im nächsten Schritt für die weitere Operation im Rechenwerk interpretiert [11, S. 41]. Die verwendeten Variablen werden aus dem Speicher für die Durchführung der arithmetischen oder logischen Operation im Rechenwerk geholt. Bei eintretender Sprungbedingung wird in diesem Moment der Sprungbefehl ausgeführt, sowie der Befehls- oder Programmzähler geändert [11, S. 41]. Mit der Ausführung des aktuellen Befehls erhöht sich der Befehls- oder Programmzähler um den in der Computerarchitektur definierten Wert, wodurch die Adresse, die sich im Register im Steuerwerk befindet, geändert wird. Das Steuerwerk bekommt auf diese Weise die zu bearbeitenden Anweisungen eines Programms [12, S. 144-145].

- *Das Rechenwerk*

  Das Rechenwerk führt die logischen und arithmetischen Operationen aus [12, S. 144]. Die Daten für die logischen und arithmetischen Operationen werden in Speichern innerhalb des Rechenwerks in einem oder mehreren Registern oder über einen Bus (Binary Unit System) in Verbindung mit dem Hauptspeicher übertragen [9, S. 44]. Das Rechenwerk hat für die Operationen, für die Aufnahme der Variablen und deren Ergebnisse [11, S. 144] sogenannte Akkumulatoren [12, S. 144].

- *Der interne Adress-, Daten- und Steuerbus*

  Die Kommunikation zwischen den Komponenten eines Rechners erfolgt durch Bussysteme [9, S. 44-45]. Diese arbeiten bidirektional für die Übertragung der Daten zwischen dem Prozessor, RAM und der Peripherie [11, S. 41]. Die Menge der Leitungen des Datenbusses deckt sich für gewöhnlich mit der Datenbreite [9, S. 91] des Rechenwerkes [11, S. 41, 12, S. 144]. Durch die Menge der Datenbusleitungen wird die Datenrate (Bytes pro Takt) bestimmt [11, S. 41]. Die Datenbreite des Datenbusses eines 64-Bit Prozessors überträgt 64 Datenbits pro Takt zum RAM und umgekehrt [11, S. 41-42]. Die Adressierung der Speicheradressen und der Peripherie-Geräte erfolgt durch den Adressbus. Die Menge der Leitungen des Adressbusses bestimmt die maximale Menge der adressierbaren

Speicheradressen. Bei 32 Leitungen des Adressbusses sind 2^{32} Speicheradressen erreichbar. Ein Prozessor dieser Dimension kann im ganzen 2^{32} Bytes (4GB) an Speicher ansprechen [11, S. 42]. Die Lese- und Schreibbefehle, Unterbrechungsanfragen (Interrupt), Steuerung des Zugriffs auf den Bus, die Bustaktsteuerung, Leitungen für den Reset und den Status, sowie die Leitungen des Ein-/ Ausgabewerks werden durch den Steuerbus gesteuert [11, S. 42]. Der Steuerbus und der Adressbus arbeiten im Gegensatz zum restlichen Bussystem unidirektional, um Signale vom Prozessor an Komponenten zu senden [13].

- *Ein-/ Ausgabewerk*

  Wie der Name schon andeutet ist das Ein-/ Ausgabewerk eine Schnittstelle für externe Peripherie bzw. für die Ein- und Ausgabe von Programmen und Daten [9, S. 42]. Der Kern des Prozessors eines Mikrocontrollers kann keine analogen Daten bearbeiten. Daher ist oft ein Analog-/ Digitalwandler im Ein-/ Ausgabewerk vorhanden, der analoge Daten in eine digitale Form umwandelt. Digitale Übertragung von Daten bedeutet, dass Daten von A nach B übergeben werden. Diese Daten sind binär kodiert. Somit wird eine Mischung aus den Zahlen 1 und 0 übertragen. Die Zahl 5 könnte z.B. als binäre Dualzahl 00000101 dargestellt werden. Die Übertragung kann gleichzeitig über eine bestimmte Anzahl von Leitungen parallel laufen. Oft ist aber die Reduzierung der Leitungen erwünscht, sodass die Übertragung auch seriell also in einer Folge ablaufen kann. Eine analoge Übertragung von Daten würde bedeuten, dass analoge elektrische Größen wie Spannungen und Ströme übertragen werden. Beispielsweise kann die Zahl 5 durch eine analoge Spannung von 5V übertragen werden [9, S. 142].

- *Speicherwerk*

  Das Speicherwerk nimmt Daten und Programme auf. Auf den Speicher haben das Steuerwerk und das Rechenwerk Zugriff [12, S. 144].

Für Prozessorarchitekturen und für Mikroarchitekturen stellt die von-Neumann-Architektur einen Orientierungsrahmen dar [9, S. 41].

Jede Prozessorarchitektur hat seine eigene Definition. Unter den Prozessorarchitekturen finden sich beispielsweise [9, S. 383]:

- PowerPC,
- IA-64,
- MIPS IV.

3.2 Mikroprozessor und Mikrocontroller

Den Einsatz findet der Mikroprozessor unter anderem in Mikroprozessorsystemen z.B. bei technischen Gebrauchsgegenständen wie Kaffeemaschinen, die von einem Mikroprozessor gesteuert werden [9, S. 1].

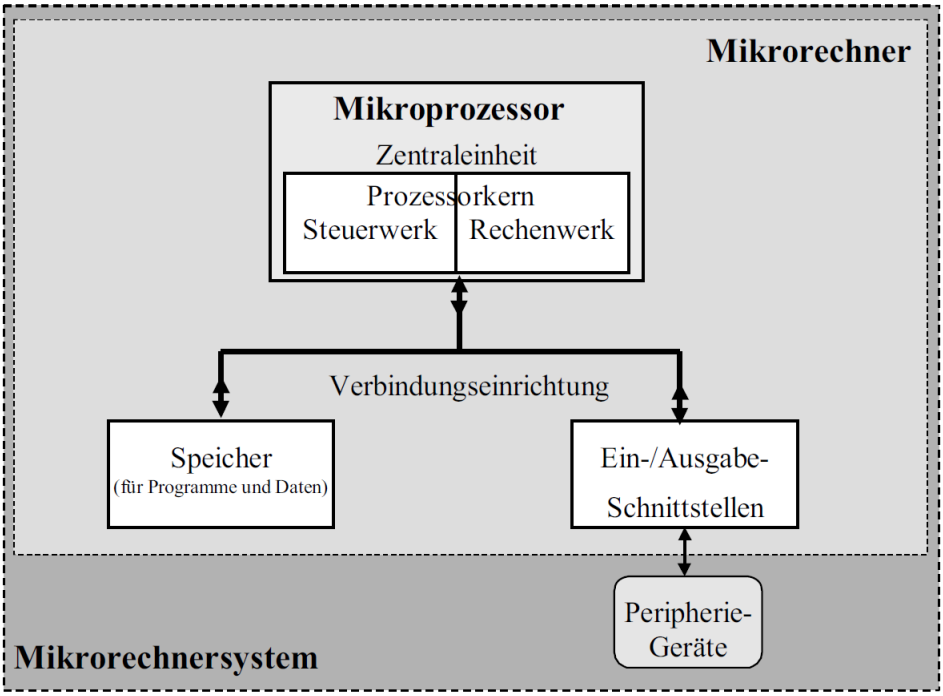

Abbildung 2: Begriffserklärung [9, S. 2]

Bei Mikrorechnern oder Mikrocomputern sind ein oder mehrere Mikroprozessoren im Einsatz. Zusätzlich sind in Verbindung mit dem Mikroprozessor Speicher, Ein-/Ausgangsschnittstellen und ein Verbindungssystem vorhanden.

Angeschlossene Peripherie-Geräte wie z.B. Eingabegeräte, ein Anzeigebildschirm, ein Drucker oder ähnliches führen zum Begriff Mikrorechnersystem oder Mikrocomputersystem.

Ein Mikrocontroller soll einen Mikrorechner auf einem Chip darstellen. Das Ziel ist es, möglichst wenige Bausteine für Steuerung oder Kommunikation zu nutzen [9, S. 1].

Die Integration der CPU, Speicher, Peripheriekomponenten und Interrupt-Systemen auf einem Chip des Mikrocontrollers führt zu einer geringeren Anzahl an externen Bausteinen für den Betrieb.

Durch die funktionelle Integration, die im Gegensatz zu dem Mikroprozessor im Vordergrund eines Mikrocontrollers steht, werden die Kosten in der Produktion in Verbindung mit dem vereinfachtem Schaltungsentwurf, der kompakten Bauweise und durch die geringe Anzahl an externen Bausteinen geringer als bei Mikroprozessoren. Die geringe Anzahl von Sockeln, Leitungen und Steckern, die durch die wenigen externen Bausteine wegfallen, verringern zusätzlich Störungen auf den Verbindungen [10, S. 258].

3.2.1 Eingebettete Systeme

Mikrocontroller und Mikroprozessoren werden unteranderem in eingebetteten Systemen (Embedded Systems) eingesetzt. Eingebettete Systeme sind Rechner, die in einem technischen Zusammenhang eingebunden sind, wie z.B. bei Unterhaltungselektronik, Kraftfahrzeugen oder Kaffeemaschinen. Sie steuern und regeln Vorgänge in einem technischem System [1, 9, S. 8].

Bei Kaffeemaschinen steuern und regeln die eingebetteten Systeme z.B. die Kommunikation zwischen Heizelement, Wasserbehälter und Ventile [9, S. 8].

Die eingesetzten Mikrocontroller oder Mikroprozessoren übernehmen [1, 9, S. 8]:

- Überwachungs-, Steuerungs- oder Regelfunktionen
- Daten- bzw. Signalverarbeitung

Die kompakte Bauweise eines Mikrocontrollers, sowie die funktionelle Integration von z.B. Speichern, CPU, Interrupt-Systeme und Peripheriekomponenten auf einem Chip können kostengünstig produziert und getestet werden. Sie haben sich so als Steuerungszentrale in eingebetteten Systemen etabliert [10, S. 258]. Der Vorteil von Mikrocontrollern besteht in der Flexibilität, Aufgaben auf Basis der Software bzw. Firmware, zu erledigen. Die Funktion eines Mikrocontrollers kann mit der Software bzw. Firmware geändert oder erweitert werden [10, S. 258].

4 Sipeed Longan Nano

Bei dem kleinen Entwicklungsboard Sipeed Longan Nano handelt es sich um ein Board mit einem 32-Bit RISC-V Mikrocontroller, mit der genauen Bezeichnung GD32VF103CBT6 von der Firma GigaDevice [14]. Das Board ist mit einem 0,96 Zoll IPS RGB LCD Bildschirm mit einer Auflösung von 160 x 80 Pixel bestückt [15], welches z.B. die Darstellung von Bildern ermöglicht, die sich auf einer microSD-Karte befinden.

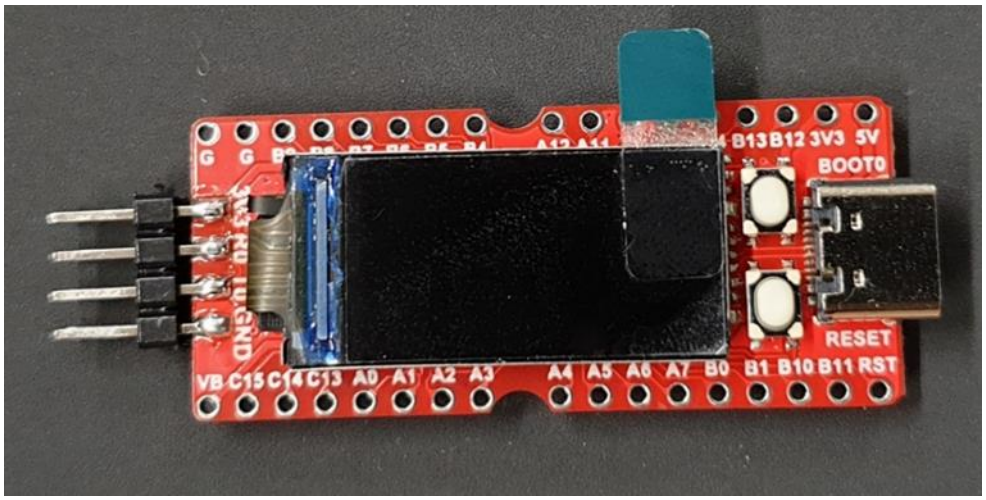

Abbildung 3: Sipeed Longan Nano

Das Board, das in der Abbildung 3 zu sehen ist, unterstützt einige Download-Methoden für Programm-Code. Diese sind [15]:

- UART ISP
- JTAG
- USB DFU

Für die Programmierung des Sipeed Longan Nanos wird die Stromversorgung des Boards über den USB-Anschluss durch ein Netzteil oder durch das Anstecken an einen PC gewährleistet. Die Verbindung mit dem PC erfolgt über die JTAG-Schnittstelle für die Inbetriebnahme in Eclipse über OpenOCD.

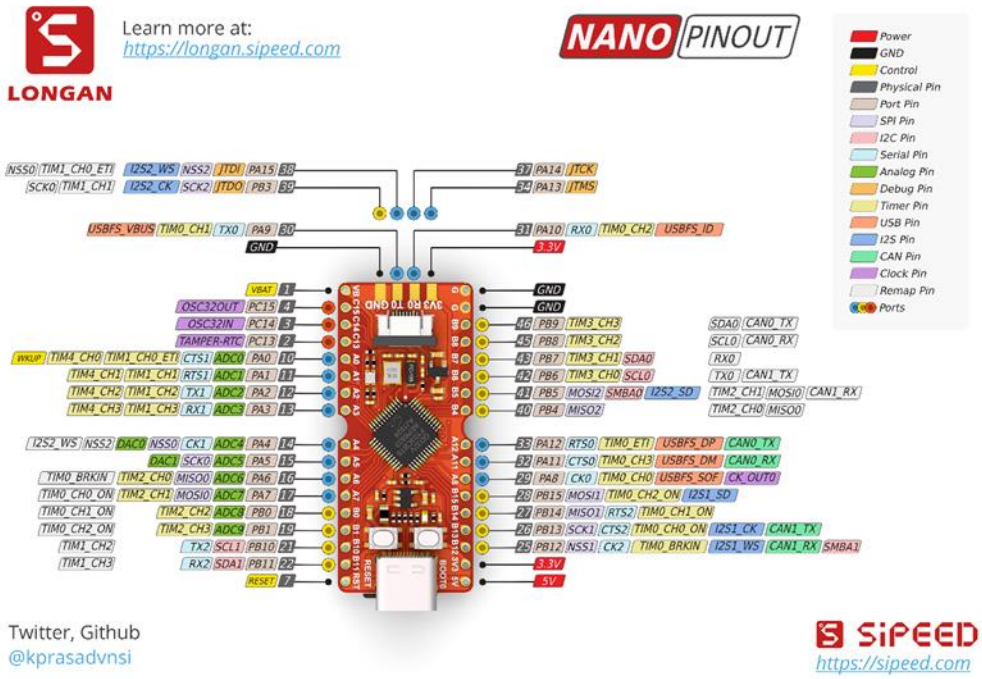

Abbildung 4: PIN MAP [14]

Abbildung 4 zeigt die Anschlüsse mit den jeweiligen für die einzelnen PINs verfügbaren Konfigurationsmöglichkeiten an.

Das Board verfügt beispielsweise über die folgenden Timer-Module [14]:

- 4 x universal 16-bit timer
- 2 x basic 16-bit timer
- 1 x advanced 16-bit timer

Die Zähler werden in der Regel für die Erfassung und Generierung von Impulsen, die Ermittlung von Zeiten und die Generierung von PWM (Pulsweitenmodulation) Signalen verwendet [10, S. 263]. Die Ermittlung der Zeit erfolgt durch die Zählung eines periodischen Signals einer nicht fremden Frequenz, d.h. eines Taktes. Die Timer (Zeitgeber) und Counter (Zähler) sind daher eng miteinander verbunden. Die Funktion des Zeitgebers hängt somit vom Zähler ab [9, S. 169].

Des Weiteren verfügt der Mikrocontroller über einen Watchdog-Timer [14]. Dieser setzt im Fehlerfall z.B. beim Absturz des Mikrocontrollers bei Ausführung eines

fehlerhaften Programmcodes das Board zurück, sodass der Mikrocontroller seine Arbeit weiterverrichten kann [16, S. 200].

Außerdem verfügt der Mikrocontroller noch über einen weiteren speziellen Timer:

- RTC (Real Time Clock) [14]

Dieser interne Zeitgeber dient dazu, bei einer Abschaltung der Energiezufuhr die Zeit für den Mikrocontroller festzuhalten [16, S. 211].

- Systick [14]

Der Systick oder System Tick Timer wird benötigt, um durch Interrupts einen Kontextwechsel zwischen verschiedenen gleichzeitig ablaufenden Vorgängen im Mikrocontroller durchzuführen [17, S. 2-5].

- 3 x USART [14]

Der Mikrocontroller weist eine Vielzahl von seriellen Kommunikationsschnittstellen auf. Unter anderem gehören dazu drei USART Schnittstellen. USART steht für “Universal Synchronous Asynchronous Receiver Transmitter“ und ermöglicht den Datenaustausch zwischen zwei Teilnehmern [9, S. 219]. Hierbei werden die zu sendenden Daten parallel-seriell gewandelt und nach einem festen Ablauf über die Schnittstelle bidirektional gesendet [9, S. 219-220]. In dem Ablauf der USART Schnittstelle befindet sich das Start/Stop und Parity-Bit für die Übertragung einer Bitfolge [9, S. 153].

Bei der Übermittlung einer Bitfolge ist es anhand eines sogenannten Parity-Bits möglich Übertragungsfehler zu ermitteln. Sender und Empfänger einigen sich zunächst auf eine gerade oder ungerade Paritätsberechnung und nutzen dafür die Werte 0 und 1 als Paritätsbit. Hierfür wird von beiden Teilnehmern die Anzahl aller 1-Bits in einer Bitfolge zusammengerechnet und bestimmt, ob es sich bei dieser Summe um eine gerade oder ungerade Zahl (Parität) handelt. Schließlich wird je nach Parität und der von den Teilnehmern vorher definierten Paritätsberechnung ein entsprechendes Parity-Bit festgehalten. So könnte etwa das Paritätsbit 1 für eine gerade Zahl und der Wert 0 für eine ungerade Anzahl aller 1-Bits in einer Bitfolge stehen. Wenn der Empfänger nun nach der Übertragung ein anderes Paritätsbit feststellt als der Sender, kann man von einem Übertragungsfehler ausgehen [18].

- 2 x I2C [14]

Der Mikrocontroller beinhaltet zwei I2C (Inter-Integrated Circuit) Kommunikationsschnittstellen, wobei sich eine intern und eine extern in einem separaten Bauteil befindet. Die I2C Schnittstelle arbeitet nach dem Master/Slave Prinzip und wird unter anderem für den bidirektionalen Datenaustausch mit verschiedenen Komponenten auf einer Leiterplatte verwendet [16, S. 350]. Bei der I2C Schnittstelle handelt es sich um einen Kommunikationsbus, an dem mehrere Teilnehmer angeschlossen sein können [12, S. 87].

- 3 x SPI [14]

Außerdem verfügt der Mikrocontroller über drei SPI (Serial Peripheral Interface) Busse. SPI ist eine übliche Kommunikationsschnittstelle mit der externe Peripheriegeräte mit einem Mikrocontroller verbunden werden. Die SPI Schnittstelle arbeitet ebenfalls nach dem Master/Slave Prinzip, wobei die Kommunikation synchron nach dem Takt des Masters abläuft und Teilnehmer über drei Leitungen an den BUS angeschlossen werden. Bei den Signalen handelt es sich um die Auswahlleitung SS (Slave Select Line), welche den entsprechenden Slave selektiert, und zwar die Leitung für die Übertragung der Signale vom “Master Output“ zum “Slave Input“ (MOSI) und die Leitung für die Übertragung “Slave Output“ zum “Master Input“ (MISO). Der Takt des Masters wird auf der SCK (Serial Clock) Leitung übertragen [9, S. 156-157].

- 2 x I2S [14]

Darüber hinaus sind in dem Mikrokontroller zwei I2S (Inter-IC Sound) Schnittstellen integriert. Der I2S ist ein Standard für die Übertragung von Audiodaten. Dieser BUS verfügt über drei Leitungen und wird ebenfalls in Master-Slave-Funktionsweise betrieben [19, S. 43].

- 2 x CAN [14]

Der Mikrocontroller verfügt auch über zwei CAN Schnittstellen. Diese Schnittstellen werden häufig im Automobilbereich für die Kommunikation zwischen elektrischen Fahrzeugkomponenten verwendet. Bei der CAN-BUS-Schnittstelle (Controller Area Network) handelt es sich um ein differentielles Bussystem, bei dem alle Informationen über ein einzige gemeinsame Datenleitung übertragen werden [10, S. 272].

- 1 x USBFS(OTG) [14]

Der Mikrocontroller ermöglicht die Verbindung zu USB-fähigen Geräten zum Informationsaustausch über das USB-Device-Filesystem.

- 2 x ADC(10 channel) [14]

Zudem verfügt der Mikrocontroller über zwei “Analog to Digital Converter“, mit denen analoge Signale in digitale Signale umgewandelt werden können. Die ADCs besitzen eine maximale Auflösung von 12 Bits und können eine maximale Arbeitsgeschwindigkeit von 2 MSPS (Mega-Samples Per Second) erreichen [16, S. 151].

- 2 x DAC [14]

Des Weiteren beinhaltet der Mikrocontroller zwei “Digital to Analog Converter“, die digitale Signale in analoge Signale umwandeln. Die DACs besitzen eine maximale Auflösung von 12 Bits [16, S. 186].

- 8MHz passive crystal [14]

Auf dem Board ist ein Quarzoszillator für die Generierung einer Schwingung mit präziser Frequenz vorhanden. Der Quarzoszillator dient als Taktgeber des Mikrocontrollers [12, S. 534].

- 32.768KHz RTC Low-Speed crystal Oszillator [15]

Ein weiterer präziser Oszillator mit geringerer Frequenz wird für die Einstellung des RTC Taktes verwendet.

- USB TYPE C Anschluss [14]

Das Board nutzt einen USB Type C Anschluss für die Energieversorgung bei einer Versorgungsspannung von 5V [16, S. 460] und den Datenaustausch mit der Entwicklungsumgebung über DFU.

- Mini TF card slot [14]

Das Mikrocontroller-Board stellt einen SD-Kartenslot bereit, mit dem beispielsweise Bilddateien für die Darstellung auf dem Display eingelesen werden können.

5 Ablauf der Erstellung von Software

Der Ablauf bei der Erstellung von Software wird im weiteren Verlauf mit den Themen Kompilieren, Verknüpfen, Startcode, Ortung, Makefile und Debugging für einen Einblick in die Softwareerstellung allgemein erklärt.

Annahmen über die Zielhardware werden bei Entwicklungswerkzeugen für Embedded Software nicht oft gestellt. Diese müssen für die weiteren Vorgänge vom Programmierer den Entwicklungswerkzeugen im Detail mitgeteilt werden. Eine fertig ausführbare Software umfasst folgende Schritte [20, S. 54]:

1. Die Quelldateien sollen kompiliert oder in Objektdateien zusammengebaut (assembliert) werden.
2. Die Objektdateien müssen zu einer Objektdatei oder einem Programm zusammengestellt bzw. verknüpft werden.
3. Innerhalb der Objektdatei des verschiebbaren Programms sollen physikalische Speicheradressen in einem Ablauf (Verschiebung) den relativen Offsets zugewiesen werden.

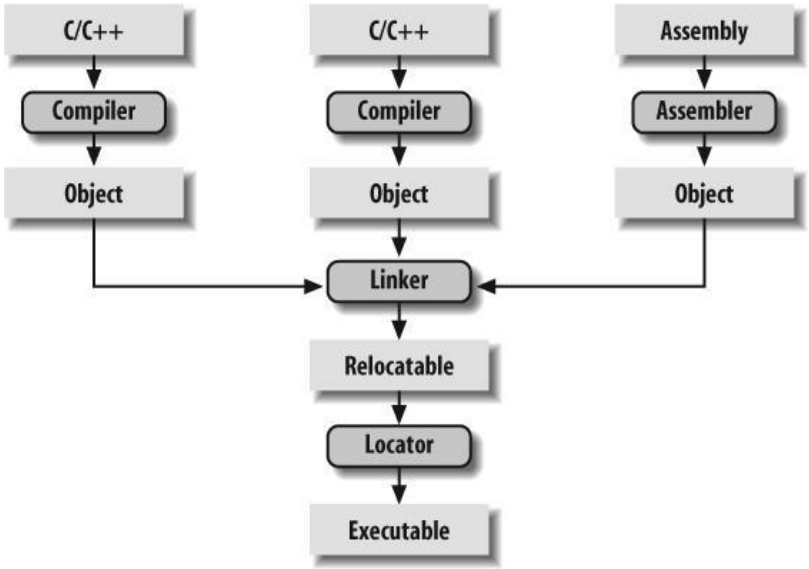

Abbildung 5: Ablauf der Erstellung von Embedded Software [20, S. 55]

Die Abbildung 5 stellt den Ablauf der Erstellung von Embedded Software dar. Auf einem Host-Computer arbeiten folgende Werkzeuge: der Compiler, der Assembler, der Linker und der Locator. Der Host-Computer wird für die Entwicklung der Embedded Software benutzt. Es handelt sich häufig um einen Personal Computer oder eine Unix-Workstation. Zusammen erstellen sie eine

ausführbare Software, die nur auf der eingebetteten Zielhardware richtig funktioniert [20, S. 54-55].

5.1 Kompilieren

Der Programmcode liegt für den Menschen in verständlicher Sprache vor und muss für den Prozessor entsprechend angepasst werden. Diese Umsetzung in die Maschinensprache, der sogenannte Opcode, wird vom Compiler übernommen. Neben dem Compiler ist auch der Assembler für den Embedded-Software-Build-Prozess zuständig. Er hat wie der Compiler die Übersetzung in Maschinensprache (Opcode) zum Ziel. Der Compiler muss entsprechend des Prozessors ausgewählt werden, welcher Programme für den entsprechenden Prozessor erstellen kann. Dieser Compiler läuft bei Embedded Systems vorwiegend auf dem Host-Computer. Der auf dem Host-Computer laufende Compiler, der für die Zielhardware Codes produziert, wird Cross-Compiler genannt. Der Compiler GCC (GNU C Compiler) und der Assembler können sowohl als native Compiler und auch als Cross-Compiler eingestellt werden. GCC kann auf allen bekannten Personal Computer- und Mac-Betriebssystemen ausgeführt werden. Insgesamt werden sehr viele Prozessoren unterstützt. Dazu zählen MIPS, Intel x86, PowerPC, ARM, SPARC und AVR.

Der Cross-Compiler erzeugt eine Objektdatei aus dem Programmcode z.B. aus C, C++, Assembler und weitere. Eine Objektdatei ist eine besondere zusammengestellte Binärdatei, die übersetzte Anweisungen und Daten beinhaltet. Auch maschinenlesbarer Programmcode ist enthalten, der aber nicht direkt ausgeführt werden kann. Die Objektdatei ist ein Teil eines großen Programms. Dateiformate wie COFF (Common Object File Format) und ELF (Executable and Linkable Format) sind oft für Objektdateien in Verwendung. Im Zusammenspiel mit weiteren Compilern für weitere Programmiersprachen müssen die Compiler das gleiche Format der Objektdatei erstellen können.

Oft starten die Objektdateien mit einer Kopfzeile (Header). Diese sind mit einem oder mehreren Blöcken aus Code und Daten aus dem Programmcode bestückt. In ähnliche Bereiche werden die Blöcke durch den Compiler unterteilt. Beispielsweise werden in *gcc* die Codeblöcke in den sogenannten *text*-Breich, die initialisierten globalen Variablen (sowie die Startwerte) in den sogenannten Bereich *data* und die nicht initialisierten globalen Variablen in den Bereich des

sogenannten *bss* unterteilt [20, S. 56-57]. Namen und Speicherorte der gesamten Variablen und Funktionen, welche aus dem Programmcode verwiesen werden, werden üblicherweise in einer Symboltabelle in der Objektdatei abgelegt. Bei lückenhaftem Inhalt der Tabelle ist es die Aufgabe des Linkers, die Probleme der Verweise zu lösen [20, S. 57].

5.2 Verknüpfen/ Linken

Die gesamten einzelnen Objektdateien aus dem Compiler-Prozess werden im nächsten Schritt verbunden bzw. verknüpft. Der Grund für diese Zusammenfügung ist, dass die einzelnen Objektdateien selbständig nicht vollständig sind, sowie dass die Verweise der Variablen und Funktionen noch aufgeklärt werden müssen. Wenn die Aufgabe des Linkers abgeschlossen ist, wird aus mehreren Objektdateien eine einzelne Objektdatei zusammengestellt, die den Code und die Daten aus den einzelnen Objektdateien enthält, sowie dasselbe Format wie die verbundenen Objektdateien besitzt.

Die Bereiche *text*, *data* und *bss* der einzelnen Objektdateien werden für diesen Vorgang zusammengestellt. Aus den einzelnen Bereichen *text*, *data* und *bss* der einzelnen Objektdateien wird so ein gemeinsamer Bereich *text*, *data* und *bss*. Während des Vorgangs sucht der Linker nach nicht aufgeklärten Verweisen der Bereiche *text*, *data* und *bss*. Ein Beispiel ist der Abgleich von zwei Variablen mit gleichem Namen in zwei verschiedenen Objektdateien. Bei einem nicht aufgeklärten Verweis einer Variablen, wird der richtige Verweis der Variable zugeordnet. Beispielsweise wenn sich die Variable an Offset 14 des Bereichs *data* befindet, wird für die Variable die Adresse an Offset 14 in die Symboltabelle eingetragen.

Der GNU Linker läuft wie der GNU Compiler auf dem Host-Computer. Der GNU Linker ist ein Werkzeug zur Verknüpfung von Bezeichnungen der Objektdateien und Bibliotheken. Bei Embedded Software soll eine Objektdatei mit dem kompilierten Startcode enthalten sein und in die Symboltabelle aufgenommen werden. Der GNU Linker verwendet eine Skriptsprache, um die Erzeugung der gemeinsamen Objektdatei aus den verknüpften Objektdateien zu steuern. Wenn sich gleiche Elemente in mehreren Objektdateien befinden, gibt der Linker dem Programmierer eine Fehlermeldung aus. Der Linker beendet dann seine Arbeit und fährt mit seiner Aufgabe nicht mehr fort. Nicht aufgeklärte Verweise versucht

der Linker selbst zu lösen [20, S. 57]. Er vergleicht z.B. Systemfunktionen *memcpy*, *strlen* oder *malloc* der Standard C Bibliothek mit dem Verweis, da der Verweis einen Zusammenhang mit diesen Funktionen in den angegebenen Bibliotheken (nach vorgegebener Ordnung) haben könnte und prüft ihre Symboltabellen. Auf diese Art und Weise wird versucht die aufzuklärenden Verweise zu ordnen, sodass die Programm- und Datensätze in die Objektdatei, die am Abschluss der Aufgabe des Linkers erzeugt werden, geschrieben werden können. Die weiteren Funktionen, die nicht verwiesen wurden, werden durch die selektive Arbeitsweise des Linkers aus der Objektdatei ausgeschlossen.

Die C-Standartbibliothek erfordert in vielen Entwicklungsumgebungen Anpassungen vor der Verwendung in Embedded Software, da sie als Objektform vorliegt. Auf den Code dieser Bibliotheken ist jedoch oft kein Zugriff für Änderungen vorhanden. Cygnus hat die kostenlose Version *newlib* der C-Standardbibliotheken für Embedded Systems erstellt. Nachdem die Bibliothek eingerichtet wird, kann diese mit der Embedded Software verknüpft werden. Durch diesen Schritt können die nicht aufgeklärten Aufrufe auf die Standardbibliotheken zugewiesen werden. Am Ende des Linkervorgangs wird nach der Verknüpfung der gesamten Code- und Datensätze, sowie der Aufklärung aller Symbolverweise, eine Objektdatei erstellt. Diese Objektdatei wird auch als verschiebbare Kopie des Programms bezeichnet. Die Speicheradressen sollen Code und Datensätzen bei Embedded Systems zugewiesen werden. Bei Verwendung eines integrierten Betriebssystems im Embedded System benötigen die Adressen der Symbole in der Objektdatei sehr wahrscheinlich ein eindeutig lokalisierbares Binärabbild. Durch das integrierte Betriebssystem sind mit hoher Wahrscheinlichkeit Code- und Datensätze in dem verschiebbaren Programm zu finden. Oft wird die gesamte Embedded Software inkl. des Betriebssystems miteinander verknüpft, sodass ein ausführbares Binärabbild entsteht [20, S. 58].

5.3 Startcode

Die Entwicklungswerkzeuge bereiten den Programmcode für die Ausführung vor. Dafür wird der Startcode aus einem Abschnitt, der in Assemblercode geschriebenen wird, in den Programmteil, der in einer Hochsprache geschrieben wurde, eingesetzt. Hochsprachen stellen an die Entwicklungsumgebung

spezielle Anforderungen [20, S. 58]. Programme, die in der Programmiersprache C geschrieben sind, verwenden beispielsweise ein Stack für die ordnungsgemäße Ausführung des Programms.

Für diesen Stapelspeicher wird durch den Startcode Speicher zugewiesen. In Cross-Compilern sind oft die Assemblerdateien *Startup.asm* und *crt0.s* enthalten. Der Inhalt und der Ort bzw. der Speicherplatz dieser Dateien wird in der Dokumentation des Compilers beschrieben. Die Abfolge der Aufgaben des Startcodes ist wie folgt [20, S. 59]:

- Interrupts stoppen,
- Übertragung der initialisierten Daten aus dem Speicher *ROM* (Read Only Memory [12, S. 1]) in den Arbeitsspeicher *RAM* (Random Access Memory [12, S. 1]),
- den nicht initialisierten Datenraum zurücksetzen,
- Stack Speicher zuweisen und initialisieren,
- Hauptprogramm *main* aufrufen.

Nach dem Aufruf des Hauptprogramms *main,* sind in der Regel Anweisungen im Startcode enthalten. Mit diesen Anweisungen ist es je nach eingesetztem Embedded System möglich, den Prozessor anzuhalten, das Embedded System zurückzusetzen oder die Steuerung an das Debugging-Werkzeug weiterzugeben. Die Ausführung der Anweisungen erfolgt mit der Beendung des Hauptprogramms *main*. Der Startcode wird in der Regel nicht automatisiert eingefügt, sondern muss vom Programmierer manuell einfügt und in die Reihe der Eingangsdateien des Linkers der entsprechenden Objektdatei eingebunden werden. Gegebenenfalls muss dem Linker eine zusätzliche Befehlszeilenoption mitgeteilt werden, um den Standart-Startcode nicht einzubinden [20, S. 59].

5.4 Ortung

Das Entwicklungswerkzeug Locator wandelt das verschiebbare Programm in ein lauffähiges Binärabbild, welches in die Ziel-Hardware übertragen werden kann. Dafür muss der Speicher des Ziel-Systems konfiguriert werden, um Code- und Datensätzen physikalischen Speicheradressen mitzuteilen. Den Programmteilen des verschiebbaren Programms müssen Adressen zugewiesen werden. Die Zuweisung der Adressen wird durch ein spezielles Entwicklungswerkzeug namens Locator durchgeführt. Bei GNU Werkzeugen ist der Locator im Linker

enthalten [20, S. 59]. Durch das Linker-Script werden Informationen über die physischen Speicheradressen mitgeteilt und die Steuerung der Anordnung der Code- und Datensätze im verschiebbaren Programm erleichtert [20, S. 60].

5.5 Makefiles

Eine Vielzahl von Quelldateien sorgen für eine große Zahl an Linker- und Compilerbefehlen, welche manuell ohne das Makefile in die Befehlszeile eingetragen werden müssen. Durch das Makefile werden diese Befehle automatisiert der Befehlszeile übergeben. Dafür wird das Makefile dem Werkzeug *make* für die Erstellung eines Programms übergeben. Make wird typischerweise mit den GNU Werkzeugen installiert. Die Anweisungen in dem Makefile werden durch das Werkzeug *make* ausgeführt und entsprechende automatisierte Ausgaben durch die Eingabedateien getätigt. Der hohe Aufwand der Erstellung der Makefile spart bei der häufigen Erstellung von Projektdateien viel Zeit [20, S. 66].

5.6 Debugging

Debugger sind Hilfsprogramme, welche die Suche nach Softwarefehlern erleichtern. Sie übertragen den Programmcode in den Programmspeicher nach dem Erstellungsprozess oder Build-Prozess. Für die Suche nach Fehlern in der Software kann der Programmcode schrittweise im Prozessor mit Hilfe des Debuggers bearbeitet werden. Dabei kann der Debugger den momentanen Programmcode, Inhalte von Speicheradressen, Variablen oder gesamten Registern anzeigen [21, S. 13].

Die Übertragung eines maschinenlesbaren Programms erfolgt beispielweise über die serielle Schnittstelle mit Hilfe des Monitor-Programms in die Zielhardware. Diese Methode hat den Vorteil, dass sie sehr zeitsparend bei der Übertragung des Programms auf die Zielhardware ist. Die Funktionen des Monitor-Programms sind [9, S. 107-108]:

- Debugging (Fehlersuche),
- Anzeige von Register- und Speicherdaten und
- das schnelle Erkennen von Fehlern durch den Quellcode-Debugger. Dies wird durch mögliche Haltepunkte und der anschließenden schrittweisen Ausführung des Programms ermöglicht.

Die Nachteile des Monitor-Programms sind, dass [9, S. 107-108]:

- die Programmumgebung nicht eins zu eins der Zielhardware entspricht, weshalb das maschinenlesbare Programm nicht auf den Festwertspeicher übertragen wird, sondern in den Schreib-/ Leseabschnitt der Zielhardware;
- die Initialisierung des Schnittstellen-Monitor-Programms Fehler in der Zielhardware generieren kann, welche durch das Monitor-Programm versteckt werden.

Eine andere Methode ist die Übertragung bzw. Programmierung in den Festwertspeicher. Dies erfolgt durch externe Programmiergeräte, welche sich mit der Zielhardware verbinden und das Programm in den Festwertspeicher übertragen. Als eine mögliche Erleichterung bei der Übertragung oder Programmierung ist die Integration eines Flash-Code-Loaders im Festwertspeicher. Dieser besteht aus ergänzender Hardware, sowie aus Software, welche die Festwertspeicher bestehend aus Flash-Speichern löschen und programmieren können. Mit beiden Möglichkeiten, dem Monitor-Programm und der direkten Übertagung auf den Festwertspeicher, kann der Programmcode auf die Zielhardware übertragen werden, sodass das Programm auf der Zielhardware ausgeführt werden kann. Die Methode mit Hilfe des Monitor-Programms ist schneller bezüglich der Ladezeiten als die Methode mit der Übertragung auf den Festwertspeicher. Sie bietet auch mehr Möglichkeiten zum Test des Programmablaufes auf Fehler als die Methode mit der Übertragung in den Festwertspeicher [9, S. 108].

Bevor das Programm auf das Zielsystem übertragen wird, gibt es in Entwicklungsumgebungen für Mikrocontroller auch die Möglichkeit Simulatoren auf dem Host-Computer zu verwenden. Mit einem Simulator können nur grobe Tests ohne Zielhardware durchgeführt werden, da diese das Zeitverhalten verzerren. Die hohe Leistung des Host-Computers erlaubt es, durch den Simulator Eindrücke in Bezug auf den Programmablauf zu sammeln, was in der Zielhardware in der Regel nicht möglich ist. Dazu kommt die Überwachung von Adressgebieten oder Variabelwerten, welche ungültigen Zugriffen auf dem Speicher entsprechen, die zum Absturz des Programms im Mikrocontroller führen, ausfindig machen. Programme werden durch diese zwei möglichen Varianten auf die Hardware übertragen [9, S. 107].

6 Die Inbetriebnahme in Eclipse

In diesem Kapitel wird die Inbetriebnahme des Entwicklungsboards Sipeed Longan Nano in Eclipse erklärt. Die Entwicklungsumgebung Eclipse wird mit der Version Eclipse IDE for Embedded C/C++ Developers für die erste Programmierung des Mikrocontrollers GD32VF103CBT6 der RISC-V Architektur mit der xPack GNU RISC-V Embedded GCC Toolchain in Verbindung mit den xPack Windows-Build-Tools für den Erstellungsprozess des Programmcodes verwendet. Um den Programmcode auf das Board zu flashen, wird die an die Mikrocontroller der GD32VF103 Serie von GigaDevice angepasste Version der Firma Nuclei System Technology Co. Ltd. von OpenOCD mit der JTAG-Schnittstelle des Entwicklungsboards genutzt. Um das Entwicklungsboard Sipeed Longan Nano mit USB über JTAG mit der Entwicklungsumgebung zu verbinden, wird der Olimex ARM-USB-TINY-H JTAG Adapter (siehe Abbildung 6) verwendet. Neben der Möglichkeit über JTAG kann man den erstellten Programmcode über eine *Hex* Datei mit dem DFU Tool von GigaDevice auf das Sipeed Longan Nano über den USB-Type C Anschluss übertragen. Im weiteren Verlauf wird die Installation und Einrichtung der Entwicklungsumgebung detailliert beschrieben. Anschließend wird das Eclipse CDT (C Development Tool) in Verbindung mit der xPack GNU RISC-V Embedded GCC Toolchain erläutert.

Abbildung 6: Olimex ARM-USB-TINY-H JTAG Adapter

6.1 Vorbereitung

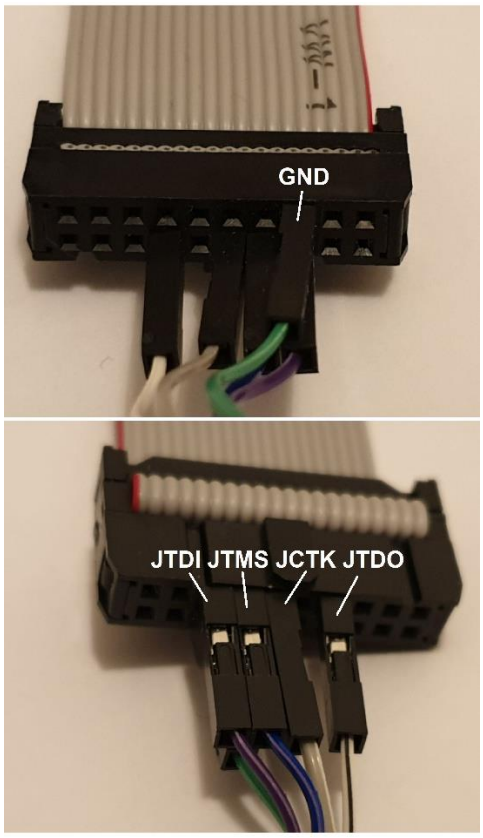

Abbildung 7: Anschluss des Entwicklungsboards an den JTAG Adapter

Zunächst muss das Sipeed Longan Nano mit dem Olimex ARM-USB-TINY-H JTAG Adapter und mit dem Personal Computer, sowie der Entwicklungsumgebung Eclipse verbunden werden. In Abbildung 7 wird der Olimex Adapter an die JTAG Schnittstelle des Entwicklungsboards Sipeed Longan Nano angeschlossen. Für eine stabile Energieversorgung wird zusätzlich ein Netzteil oder eine Verbindung zwischen dem USB-Port des PCs und dem Entwicklungsboard benötigt. Im weiteren Verlauf werden zur ersten Programmierung des *hello_led* Projektes [22, 23] auf dem Entwicklungsboard folgende Dateien benötigt, die aus dem GitHub Repository [22, 24] und teilweise über GigaDevice [22] zu entnehmen sind:

- Linker-Script,
- Bibliotheken,
- Startcode,
- Vektortabelle.

6.1.1 Die Installation und Einrichtung von Eclipse

Abbildung 8: Startbild von Eclipse in der Version eclipse IDE 2021-09

Die integrierte Entwicklungsumgebung Eclipse wurde ursprünglich für die Programmiersprache Java entwickelt. Mit Erweiterungen haben sich Projekte in weiteren Programmiersprachen realisieren lassen [25, S. 38]. Diese Erweiterungen finden sich beispielsweise im Marktplace von Eclipse oder in externen Quellen. Mit Eclipse ist es möglich, Programmcode für unterschiedliche Softwareplattformen zu schreiben [26].

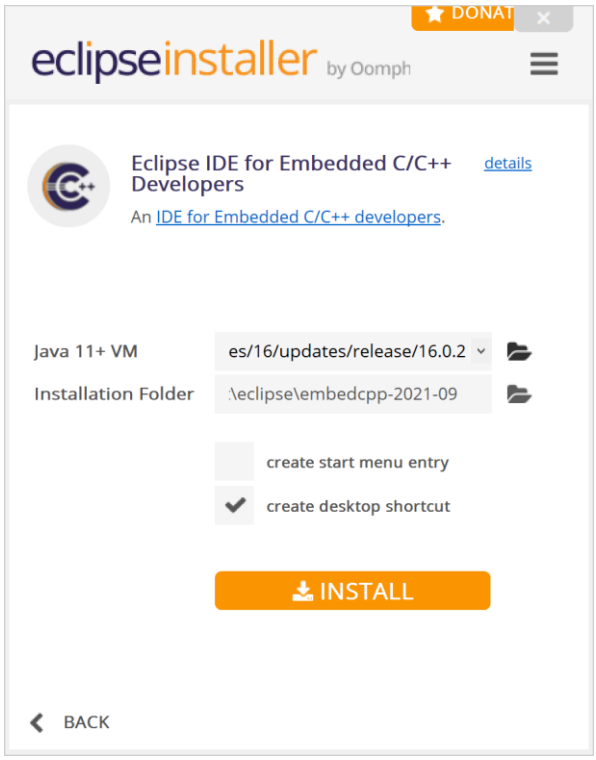

Abbildung 9: Eclipse Installer

Die integrierte Entwicklungsumgebung Eclipse wird mit einem Installer (siehe Abbildung 9) im Standard-Benutzerverzeichnis entpackt. Im Installer kann man die speziell an die Programmiersprache angepasste Version von Eclipse auswählen. Für die Nutzung der intergrierten Entwicklungsumgebung von Eclipse wird eine Java-Laufzeitumgebung (JRE, Java Runtime Environment) benötigt [27]. Mit der Java-Technik oder Java-Technologie lassen sich Programme, die auf dieser Technologie basieren, in einer Java-Laufzeitumgebung auf unterschiedlichen Computersystemen ausführen [25, S. 38-39].

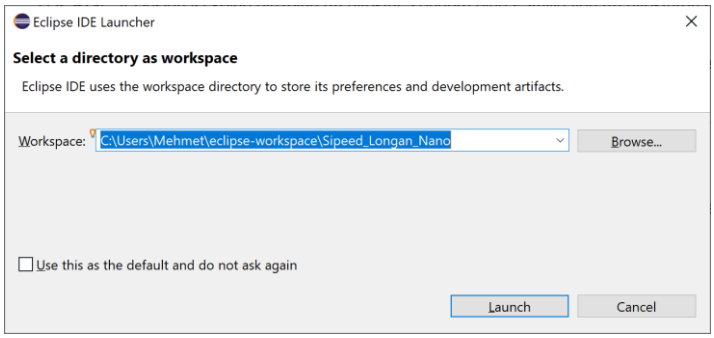

Abbildung 10: Erstellen der Workspace

Nachdem die Version von Eclipse ausgewählt worden ist, kann man die Entwicklungsumgebung ausführen. Im ersten Schritt legt man einen Ordner für seine Projekte an. Dieser Ordner wird Workspace genannt (siehe Abbildung 10). Man kann mehrere Projektordner verwalten und unterschiedliche Versionen von Projekten pflegen. Der Projektordner kann auch in anderen Eclipse-Versionen verwendet werden. Anschließend wird ein neues C/C++ Projekt mit der RISC-V Toolchain erstellt.

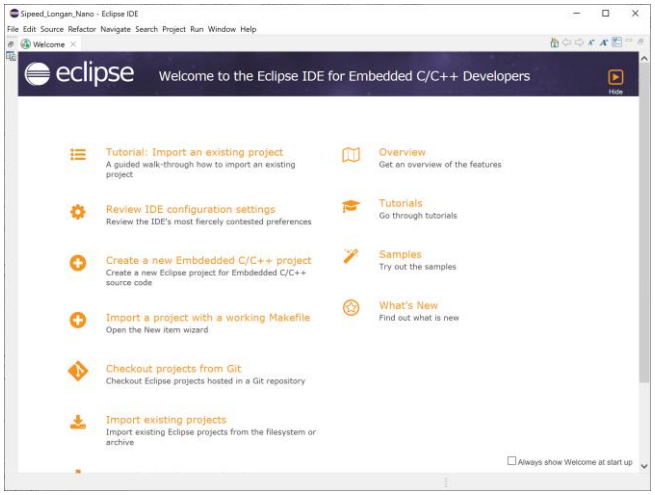

Abbildung 11: Willkommensbildschirm der Ecipse IDE

Um das erste C/C++ Projekt zu erstellen kann man entweder den Willkommensbildschirm (siehe Abbildung 11) oder die Menüleiste, wie in Abbildung 12 zu sehen ist, im oberen Teil der Entwicklungsumgebung verwenden.

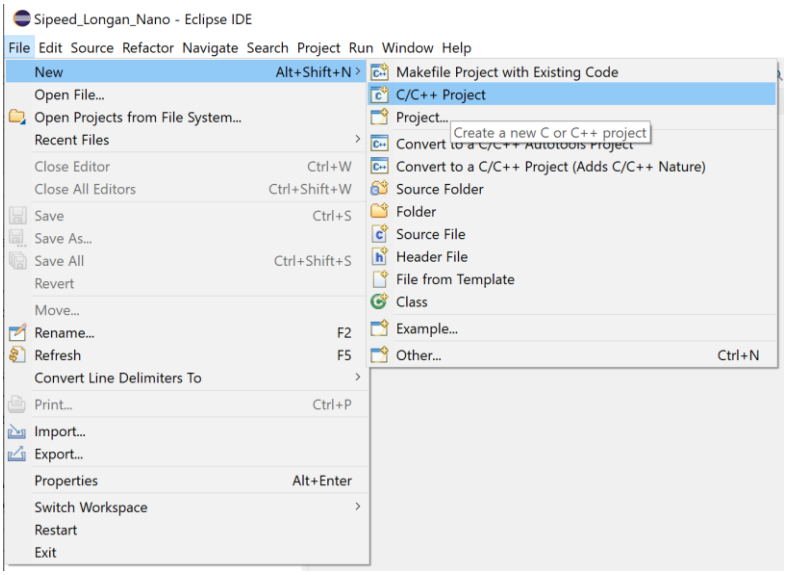

Abbildung 12: Neues C/C++ Projekt erstellen

Im Menü *Templates for New C/C++ Project* (siehe Abbildung 13) muss nun unser Projekttyp ausgewählt werden. Der Typ des Projektes wird als *C Managed Build* festgelegt, um das Blink-Beispielprogramm *hello_led* aus dem GitHub Repository zu erstellen. Dafür wird die Konfiguration der RISC-V Toolchain und für die Übertragung auf das Board die Konfiguration der Nuclei System Technology OpenOCD Version benötigt.

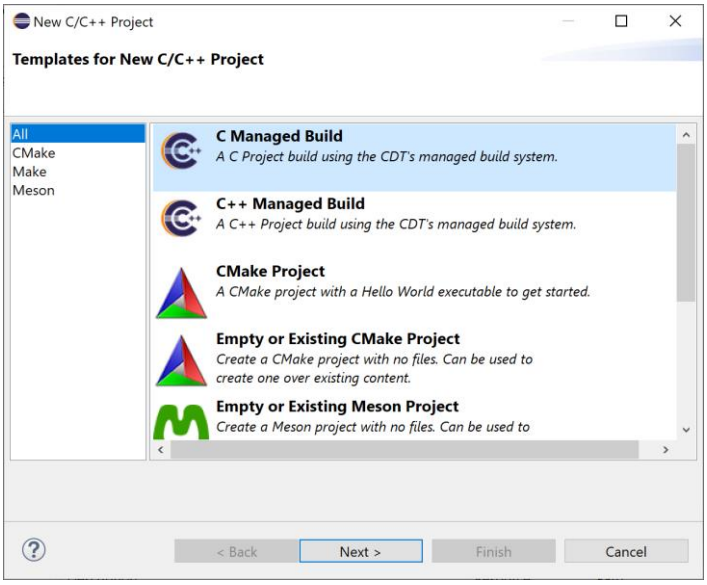

Abbildung 13: Templates for New C/C++ Project

Um einen RISC-V Projekt-Typen festzulegen, wird *Hello World RISC-V Project* mit der RISC-V Cross GCC Toolchain ausgewählt. In diesem Fenster werden bereits konfigurierte und mit einem Beispielprogramm versehene Projekte für unterschiedliche Mikrocontroller oder Boards zur Verfügung gestellt. Es wird das Sipeed Longan Nano Board mit einem RISC-V Mikrocontroller der GD32VF103 Serie von GigaDevice verwendet, das sich in dieser Version von Eclipse nicht in den Beispielprojekten unter *Executable* befindet. Dieser muss entsprechend in der Toolchain konfiguriert werden. Im nächsten Schritt werden die xPack Tools installiert und eingebunden.

6.1.2 xPack, GNU RISC-V Toolchain und Windows-Build-Tools

Die xPack GNU RISC-V Embedded GCC Toolchain, die xPack Windows-Build-Tools für *make* und die angepasste Version von OpenOCD von Nuclei werden unter anderem für den GD32VF103CBT6 Mikrocontroller von GigaDevice für die Erstellung von Software und das Debugging benötigt. Die Installation der GNU RISC-V Toolchain erfolgt durch xpm (xPack project manager), eine Command Line Interface (CLI) [28] Anwendung von Node.js. Xpm ist ein Tool um Installationen von Paketen zu vereinfachen und zu verwalten [29]. Dazu wird zunächst auf einem Windows Betriebssystem das Tool Node.js installiert. Mit dieser Anwendung lassen sich selbstständige JavaScript-Programme, unter anderem Skripte auf dem Server, Netzwerktools und WebApps programmieren. Diese sind unabhängig von Host-Anwendungen [30]. Die Version LTS (Long Term Support [31]) der aktuellen Version [32] wird für die xpm Erweiterung benötigt. Nach der Installation der aktuellen Version von Node.js erfolgt die Installation der xpm-Erweiterung mit folgendem Befehl in CMD [33]:

- npm install --global xpm@latest

Der Befehl installiert die aktuelle Version der Erweiterung xpm in node.js. Die Bezeichnung npm bedeutet Node Package Manager und ist für die Installation, sowie die Verwaltung von Erweiterungen zuständig [30]. Die GNU RISC-V Toolchain wird mit dem Befehl im Folgenden installiert [34]:

- xpm install --global @xpack-dev-tools/riscv-none-embed-gcc@latest –verbose

Die xPack GNU RISC-V GCC Toolchain ist aufbauend auf der SiFive-Toolchain [35]. Um wie im vorherigen Kapitel erklärt, Compiler- und Linkerbefehle automatisiert der Befehlszeile zu übergeben, werden die Windows-Build-Tools für das Tool *make* installiert. Dies kann vereinfacht über xPack mit dem Befehl

- xpm install --global @xpack-dev-tools/windows-build-tools@latest –verbose

erfolgen [36]. Die in dieser Arbeit genannten Befehle zur Installation der Pakete der Toolchain und der Windows-Build-Tools installieren die aktuelle Version. Durch diese Befehle lassen sich dementsprechend auch Updates einrichten. Die installierten Werkzeuge werden in Eclipse automatisch erkannt und können global oder für bestimmte Projekte eingestellt werden.

Abbildung 14: Eclipse Konfiguration der Toolchain für bestimmte Projekte

Der Speicherort der Werkzeuge kann beliebig geändert und unter Eclipse konfiguriert werden (siehe Abbildung 14). Durch den Button xPack erkennt Eclipse die installierte Version der Toolchain und fügt diese automatisch ein. Projekt *Properties* können durch einen Rechtsklick auf das Projekt oder bei ausgewähltem Projekt unter *Project* in der oberen Menüleiste eingestellt werden.

6.1.3 Nuclei OpenOCD

Die Nuclei OpenOCD Version findet sich auf der Webseite von Nuclei [37]. Dort kann das Verzeichnis der Installationsdateien auf den Host-Computer übertragen und auf einem Speicherort festgelegt werden, der dann in Eclipse global oder für ein bestimmtes Projekt konfiguriert werden kann (siehe Abbildung 15). Um auf die globalen Konfigurationsmöglichkeiten zu gelangen, wird in der oberen Menüleiste im Programmfenster von Eclipse *Window* und anschließend

Preferences gewählt. Dort angekommen wird unter Folder das entpackte Verzeichnis des Nuclei OpenOCD Tools ausgewählt.

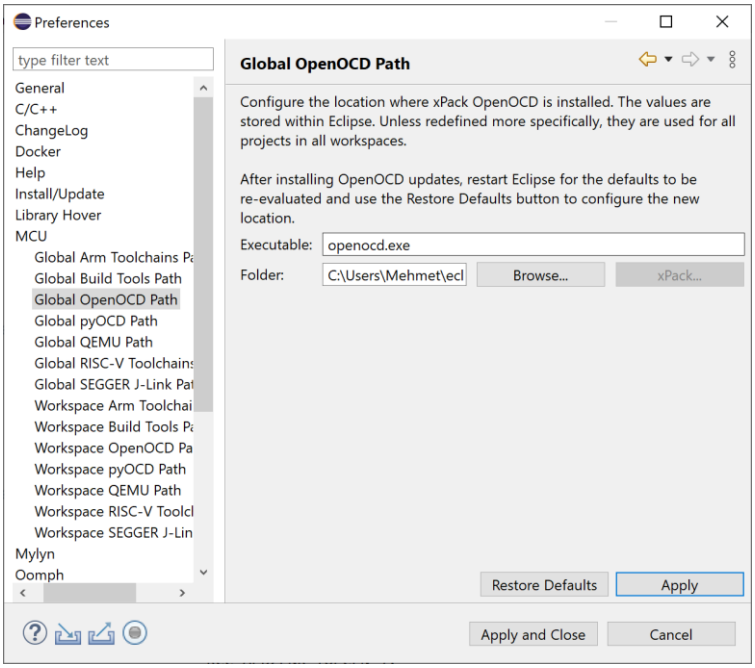

Abbildung 15: Eclipse globale Konfigurationsmöglichkeit

Dabei ist zu beachten, dass der *bin* Ordner ausgewählt wird und die Bezeichnung des Tools unter *Executable* eingetragen wird. Konfigurationen müssen mit *Apply* bestätigt werden.

Die Konfigurationsdatei *.cfg* des entsprechenden JTAG Adapters von Olimex kann unter dem OpenOCD Verzeichnis gefunden und in Eclipse unter Debug *Configurations* eingetragen werden (siehe Abbildung 17). Mit dem Befehl

- -f “C:\Users\Mehmet\eclipse\...\olimex-arm-usb-tiny-h.cfg“

wird die Konfiguration des JTAG Adapters für OpenOCD durchgeführt und unter *Config options* unter dem Begriff Debugger eingetragen.

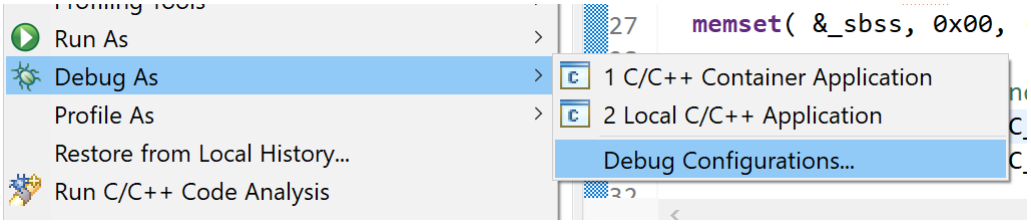

Abbildung 16: Debug Configurations

Wie in Abbildung 16 zu sehen ist, wird durch ein Rechtsklick auf das Projekt die Konfiguration des Debuggers erreicht.

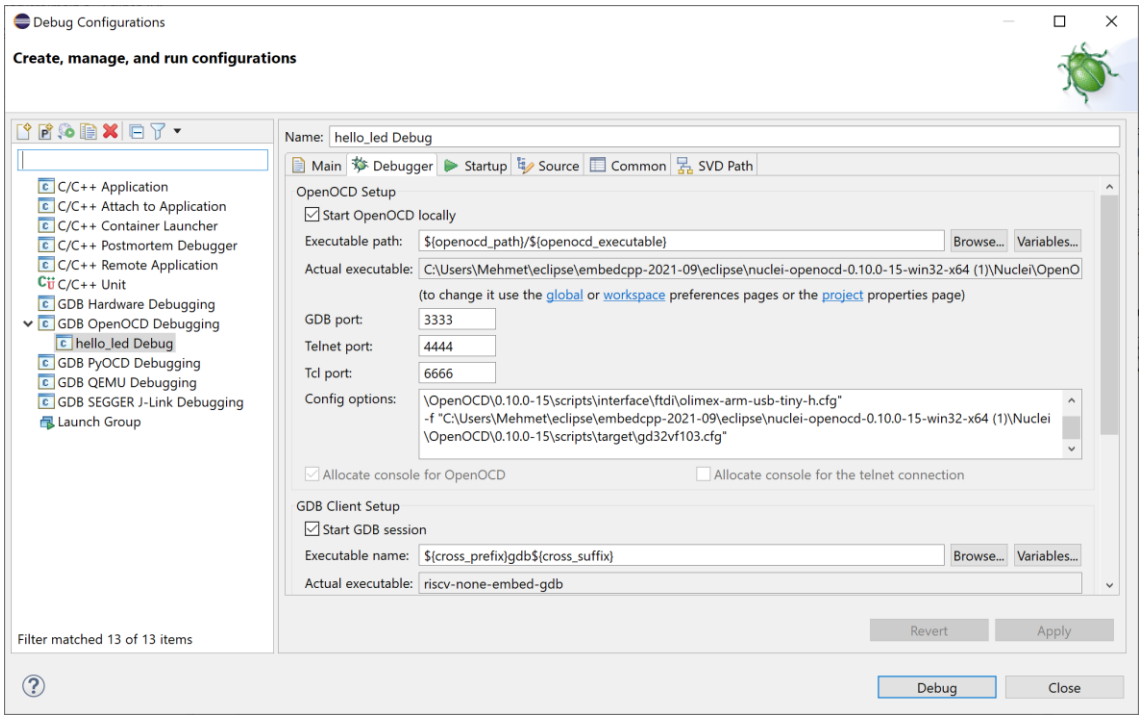

Abbildung 17: Config options

Die Konfigurationsdateien können mit dem Editor in Windows geöffnet und verändert werden. Im Prozess des Debuggings wurde der Eintrag *adpter_khz* vermisst, weshalb die gewünschte Arbeitsgeschwindigkeit des Adapters ergänzt wurde. Die Konfiguration des Olimex-ARM-USB-TINY-H JTAG Adapters ist wie folgt:

```
adapter_khz 10000

interface ftdi

ftdi_device_desc "Olimex OpenOCD JTAG ARM-USB-TINY-H"

ftdi_vid_pid 0x15ba 0x002a


ftdi_layout_init 0x0808 0x0a1b

ftdi_layout_signal nSRST -oe 0x0200

ftdi_layout_signal nTRST -data 0x0100 -oe 0x0100

ftdi_layout_signal LED -data 0x0800
```

Die Konfiguration gd32vf103.cfg wurde aus dem GitHub Repository [38] entnommen und in eine separate Konfigurationsdatei eingefügt, um den JTAG Adapter und das Board getrennt konfigurieren zu können. Die Konfiguration des Sipeed Longan Nanos bzw. des Mikrocontrollers erfolgt an Hand der Datei openocd_ft2232.cfg [38], welche die folgenden Inhalte besitzt:

```
set _CHIPNAME riscv

jtag newtap $_CHIPNAME cpu -irlen 5 -expected-id 0x1e200a6d


set _TARGETNAME $_CHIPNAME.cpu

target create $_TARGETNAME riscv -chain-position $_TARGETNAME

$_TARGETNAME configure -work-area-phys 0x20000000 -work-area-size 10000 -work-area-backup 1


set _FLASHNAME $_CHIPNAME.flash

flash bank $_FLASHNAME gd32vf103 0x08000000 0 0 0 $_TARGETNAME


init

halt
```

Der GDB-Debugger wird ausgewählt und Folgendes wird eingetragen:

- set mem inaccessible-by-default off
- set arch riscv:rv32

Der Mikrocontroller und sein Adressraum des Speichers müssen OpenOCD mitgeteilt werden. Dies wird durch die Konfigurationsdatei gd32vf103.cfg durchgeführt. Dem GDB-Debugger wird die Architektur mit *set arch risc:rv32* mitgeteilt. Nach erfolgreicher Konfiguration von OpenOCD, sowie der Installation und Einrichtung von Eclipse, ist es möglich, das Entwicklungsboard mit dem Personal Computer zu verbinden. Für die erste Programmierung fehlt die

Konfigurationen der xPack GNU RISC-V Embedded GCC Toolchain, die im weiteren Verlauf durchgeführt werden.

6.2 xPack GNU RISC-V Embedded GCC Toolchain

In Eclipse CDT (C Development Tool) werden die Einstellungen der Toolchain vorgenommen, welche die Werkzeuge für die Entwicklung der Software beinhalten. Ein anderer Begriff ist GNU Toolchain, der als Gesamtbegriff für Projekte aus dem GNU-Projekt stammt. Diese Projekte ergeben zusammen die Werkzeugkette. In einer Toolchain ist beispielsweise Folgendes enthalten (siehe Abbildung 18):

- GNU RISC-V Cross C Compiler,
- GNU RISC-V Cross Assembler,
- GNU RISC-V Cross C Linker,
- GNU RISC-V Cross Create Flash Image und
- GNU RISC-V Cross Print Size.

Die Konfiguration erfolgt bei den Projekteinstellungen unter *Settings*. Die Einstellung erfolgt unter dem Menü im oberen Abschnitt des Bildschirms *Project*, *Properties* und *Settings*. Unter diesem Punkt befindet sich an erster Stelle *Target Processor (Zielprozessor)*. Dieser muss in diesem Fenster konfiguriert werden. Die RISC-V Architektur hat vier Basis-Integer-ISAs RV32I mit 32-Bit Adressraum, RV32E mit einem reduziertem Registersatz der Basis-Integer-ISA RV32I, RV64I mit 64-Bit Adressraum und RV128I mit 128-Bit großem Adressraum [39, S. 4]. Entsprechend des Mikrocontrollers wählt man unter *Architecture* die Basis-Integer-ISA und die Erweiterungen aus. Um die nachfolgenden Konfigurationen besser zu verstehen, werden in den nächsten Zeilen einige Befehlszeilenargumente, die mit der Bezeichnung *-m* anfangen, *-march*, *-mtune* und *-mabi,* erklärt. Die Befehlszeilenargumente stehen für [40]:

- *-march=ISA*: Die Ziel-Architektur wird für die Informationsangabe der verfügbaren Anweisungen und Register dem Compiler mitgeteilt;
- *-mabi=ABI*: Die Aufrufkonvention ABI wird mit dem Argument *-mabi=ABI* festgelegt. Sie steuert die ABI (die Übertragung der Parameter in die entsprechenden Register), sowie die Form oder Anordnung, in der die Daten im Speicher liegen werden;

- *-mtune=CODENAME*: Die angestrebte Mikroarchitektur wird ausgewählt, um den GCC bezüglich der Leistungsfähigkeit der jeweiligen Befehle zu unterrichten, wodurch spezifische Zieloptimierungen vorgenommen werden können.

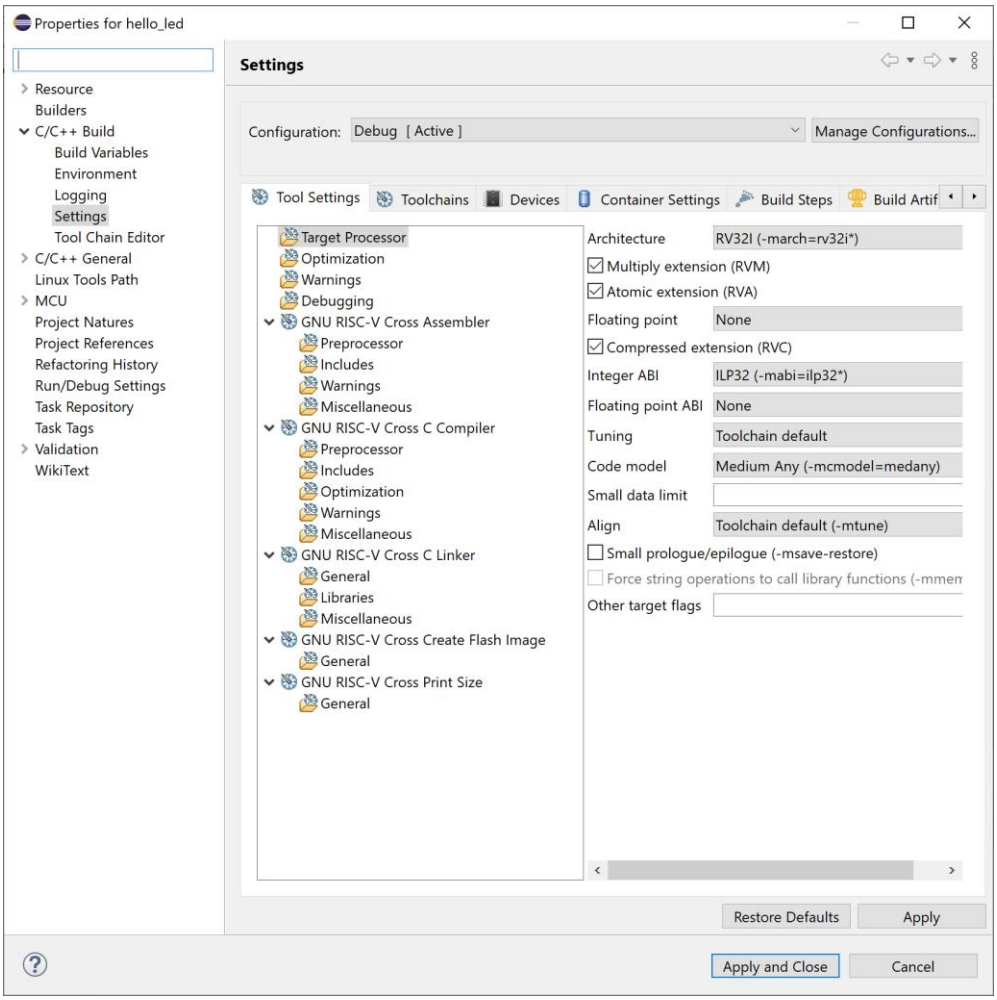

Abbildung 18: Einstellung des Target Processor's

Der Mikrocontroller GD32VF103CBT6 vom Entwicklungsboard Sipeed Longan Nano hat die Basis-Integer-ISA in Kombination mit den Erweiterungen RV32IMAC und hat einen sogenannten Bumblebee-Kern von Nuclei System Technology [15]. Das Präfix I wird für die Bezeichnung Integer angenommen und umfasst Integer-Berechnungen, -Ladevorgänge, -Speichervorgänge und Steuerbefehle [39, S. 6]. Diese Daten werden im *Target Processor* eingetragen, wie in der Abbildung 18 dargestellt.

Die Befehlssatzerweiterungen bedeuten [39, S. 6, 40]:

- M: Multiplikations- und Divisionserweiterungen, um Befehle zum Multiplizieren und Dividieren von Werten hinzuzufügen, die in den Integer-Registern gehalten werden;
- A: Hinzufügen von Befehlen zum atomaren Lesen, Modifizieren und Schreiben von Speicher für die Interprozessor-Synchronisation;
- F: Die Rechenbefehle, sowie Laden und Speichern mit einfacher Genauigkeit (Single Precision), fügen die Floating-Point Erweiterung F mit Floating-Point Registern hinzu;
- D: Die Double-Precision Floating-Point-Erweiterung fügt doppelte Genauigkeit (Double Precision) für die Rechenbefehle, sowie Lade- und Speichervorgänge hinzu und erweitert die Floating-Point-Register;
- C: Komprimierte Befehlserweiterung bietet ein 16-Bit-Format für allgemeine Befehle.

Die Befehlssatzerweiterungen werden in der genannten Anordnung der RISC-V Basis-Integer-ISA angehängt. Die RISC-V Basis-ISA RV32 mit 32-Bit Integerregistern und der Multiplikations- und Divisionserweiterung M heißt entsprechend RV32IM. Die Bezeichnung der Basis-ISA und deren Zeichenfolge kann der Programmierer in Kleinbuchstaben an den Compiler GCC für die Erstellung des Assemblycodes *-march=rv32im* übergeben. Bei nicht unterstützten Rechenoperationen lassen sich die fehlenden Rechenoperationen durch Emulationsfunktionen nachstellen.

Die Bezeichnung *-mabi* steht für die Angabe der Integer ABI und Floating-Point ABI an den Compiler GCC, die den erstellten Programmcode unterstützt. Die Bezeichnung *-march* gibt an, mit was für einer Hardware der Programmcode ausführbar ist. Mit was für einem softwareerzeugtem Programmcode der Programmcode mit dem Linker verknüpft werden kann, gibt die Bezeichnung *-mabi* an. Übliche Bezeichnungen für die Integer ABIs sind *ilp32* und *lp64* mit einem angehängten Buchstaben, welche angeben, ob Floating-Point-Register z.B. *ilp32f* oder Double-Precision Floation-Point-Register z.B. *ilp32d* verwendet werden. Für die Verknüpfung der Objekte dürfen die ABIs nicht unterschiedlich sein. Es sind zwei Integer ABIs und drei Floating-Point ABIs durch RISC-V definiert. Die Integrer ABIs entsprechen der ABI-Zeichenfolge und sind [40]:

- *ilp32*: Der Buchstabe i steht für int und l für long, sowie p für pointer. Int, long und pointer sind 32-Bit lang. Der Kürzel long ist ein 64-Bit Typ. Short besteht aus 16-Bit und char aus 8-Bit;
- *lp64*: Der Buchstabe l steht für long und p für pointer. Beide sind 64-Bit lang. Int ist ein 32-Bit Typ. Short und char sind wie bei der ilp32 definiert.

Die Floating-Point ABIs sind bestimmte Ergänzungen von RISC-V [40]:

- *“”* (leere Zeichenkette): Floating-Point-Variablen werden in Registern nicht abgelegt;
- *f*: Floating-Point-Variablen, die 32-Bit oder kleiner sind, werden in Registern abgelegt. Die F-Erweiterung wird bei dieser ABI gefordert. Ohne die F-Erweiterung sind Floating-Point Register nicht möglich;
- *d*: Floating-Point-Variablen, die 64-Bit oder kleiner sind, werden in Registern abgelegt. Die D-Erweiterung wird bei dieser ABI gefordert.

Unter anderem sind für die Übergabe an den GCC beispielsweise möglich [40]:

- *-march=rv32imac -mabi=ilp32*: Die Floating-Point-Variablen können in Registern nicht abgelegt werden und es sind keine Floating-Point-Befehle möglich;
- *-march=rv32imafdc -mabi=ilp32*: Die Hardware Floating-Point-Befehle sind möglich. Floating-Point-Variablen werden in Registern nicht abgelegt;
- *-march=rv32imafdc -mabi=ilp32d*: Hardware Floating-Point-Befehle sind möglich und die Floating-Point-Variablen können Registern übergeben werden;
- *-march=rv32imac -mabi=ilp32d*: Diese Kombination ist nicht erlaubt. Die Floating-Point-Variable setzt voraus, dass Floating-Point-Register für die Übergabe der Floating-Point-Variablen an dem Register definiert worden sind.

Die Integer ABI wird entsprechend der Architektur des Mikrocontrollers eingestellt (siehe Abbildung 18). An dieser Stelle stehen

- Toolchain Default,
- ILP32 (-mabi=ilp32*),
- ILP32E (-mabi=ilp32e*) und
- LP64 (-mabi=lp64*)

zur Auswahl. Da die Basis-Integer-ISA *RV32I (-mabi=rv32i)* in Kombination mit den Erweiterungen RV32IMAC steht, wird hier *ILP32 (-mabi-ilp32*)* ausgewählt, die unter anderem von Eclipse unterstützt wird. Diese Auswahl ist auch die Voreinstellung der Toolchain laut Eclipse. Die Größe des Integer-Typen und der verwendeten Register des Floating-Point-Typens beinhalten die ABI-Zeichenfolge. Einige Beispiele sind [41, S. 396]:

- *-march=rv64ifd -mabi=lp64d* bedeutet, dass long und Zeiger 64-Bit sind, was automatisch zu der Erkenntnis führt, dass int 32-Bit ist. Die Floating-Point-Werte bis zu einer Größe von 64-Bit können in F-Registern übertragen werden;
- *-march=rv64ifd -mabi=lp64f* bedeutet, dass der Programmcode die Erweiterungen F und D verwendet und Floating-Point-Werte in Registern bis zu einer Länge von 32-Bit zulässt;
- *-march=rv64ifd -mabi=lp64* bedeutet, dass keine Floating-Point-Werte den Registern übertragen werden.

Die Abkürzung ABI steht für application binary interface. Diese Schnittstelle definiert die Details der Implementierung des Programmcodes, genauer das Kompilierungsergebnis des Codes. Die ABI wird durch den Compiler gesteuert [42]. Zu den Aufgaben gehören unter anderem die Regelung der Aufrufkonvention. Diese regelt die Übergabe von Daten in Programmen an Unterprogramme. Dabei ist wichtig, dass die entwickelten Programme und deren Teile in vielerlei Programmiersprachen kompatible Aufrufkonventionen unterstützen [40, 42]. Folgende Punkte sollten im Ablauf der Aufrufkonvention vorhanden sein [42]:

- Die Übergabe der Steuerung an den Empfänger und zurück an den Absender,
- Speicherung und Wiederherstellung des aktuellen Standes des Absenders,
- die Übertragung von Informationen an den Empfänger und
- der Erhalt des Wertes der Rückgabe des Empfängers.

In den nächsten Zeilen wird die Regelung der Richtung des Stacks allgemein für eine Funktion der ABI Aufrufkonvention erklärt.

Parameter werden bei einem Abruf einer Funktion auf dem Stack gespeichert. Ein einziger Wert bei der Speicherung wird weniger Schwierigkeiten machen als mehrere Werte. Die Werte der Funktionen werden in Abstimmung auf dem Stack abgelegt, was ein Problem bei mehreren Werten darstellt. Der Abruf des Wertes im Stack wird “von links nach rechts“ benannt, wenn die Position des Wertes am ersten Platz ist. Wenn die Position des Wertes am letzten Platz ist, wird es “von rechts nach links“ benannt. Die Werte stehen in dieser Vorstellung in einer Reihe [43].

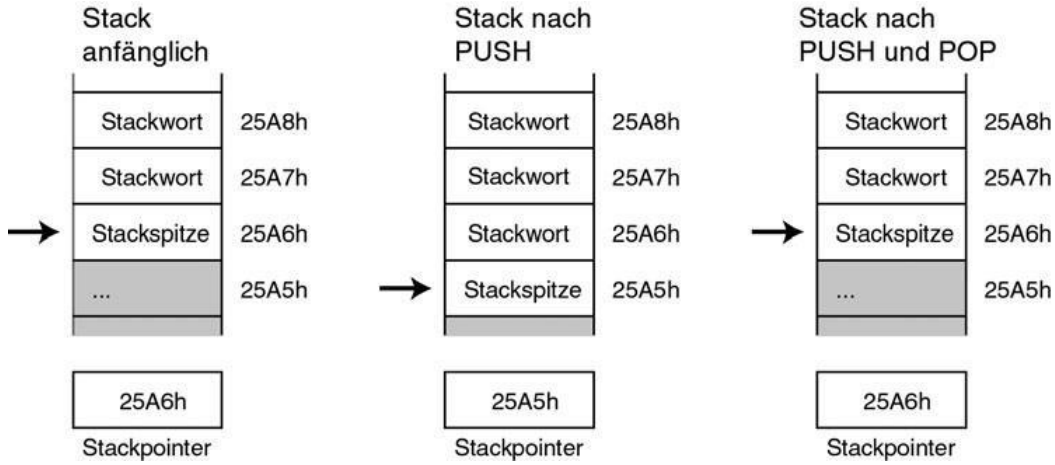

Abbildung 19: Stack [10, S. 99]

Man hat nach dem LIFO (Last-In-First-Out) Prinzip Zugriff auf den Stack. Das bedeutet, dass man Werte nur in der obersten oder letzten Position des Stapels aufrufen kann und auch nur wieder auf die letzte Position des Stapels ablegen kann. Näher betrachtet sind die Befehle an den Stack [44, S. 660-663]:

- push: Die Werte oder das Objekt auf den Stapel ablegen,
- pop: Abruf der zuletzt abgelegten Werte oder Objekte, sowie die Löschung aus dem Stapel der Aufgerufenen.

In der Abbildung 19 wird ein Stack gezeigt, der von oben nach unten wächst und Objekte aufnimmt. Dabei wird vom Stapelzeiger das unterste oder das zuletzt hinzugefügte Element angezeigt. Der Befehl *push* würde den Stapelzeiger dekrementieren und Daten auf den Stapel ablegen.

Im Gegensatz würde der Befehl *pop* Daten aus dem Stapel abrufen und diese aus dem Stapel entfernen, sowie den Stapelzeiger inkrementieren [10, S. 98-99, 12, S. 45-46]. Die Rückgabeinformationen [44, S. 221] und lokale Variablen einer Funktion werden in dem Stack abgelegt [44, S. 309]. Im Speicher hat der Stack

für das Zwischenspeichern von Daten eine feste Adresse, an dem er startet [10, S. 98]. Die Ausbreitung eines Stacks ist auf einen Adressbereich begrenzt. Der Stapelzeiger kann die Ausbreitung nach oben oder nach unten in die Richtung der Vergrößerung des Stacks anzeigen. Der Stack beginnt in der Regel bei Prozessoren mit einer hohen Adresse und wächst gegen 0. Dabei wird der Stapelzeiger durch den Befehl *push* bei geschriebenen Daten in dem Stack vermindert und die Position des Stapelzeigers gegen Ende des Stacks, also gegen die Richtung 0 geändert. Der Befehl *pop* ruft Daten ab und löscht diese aus dem Stack. Der Stapelzeiger erhöht sich hier gegen die Richtung der hohen Adresse. Somit vergrößert sich der Stack gegen niedrigere Speicheradressen [44, S. 660-661].

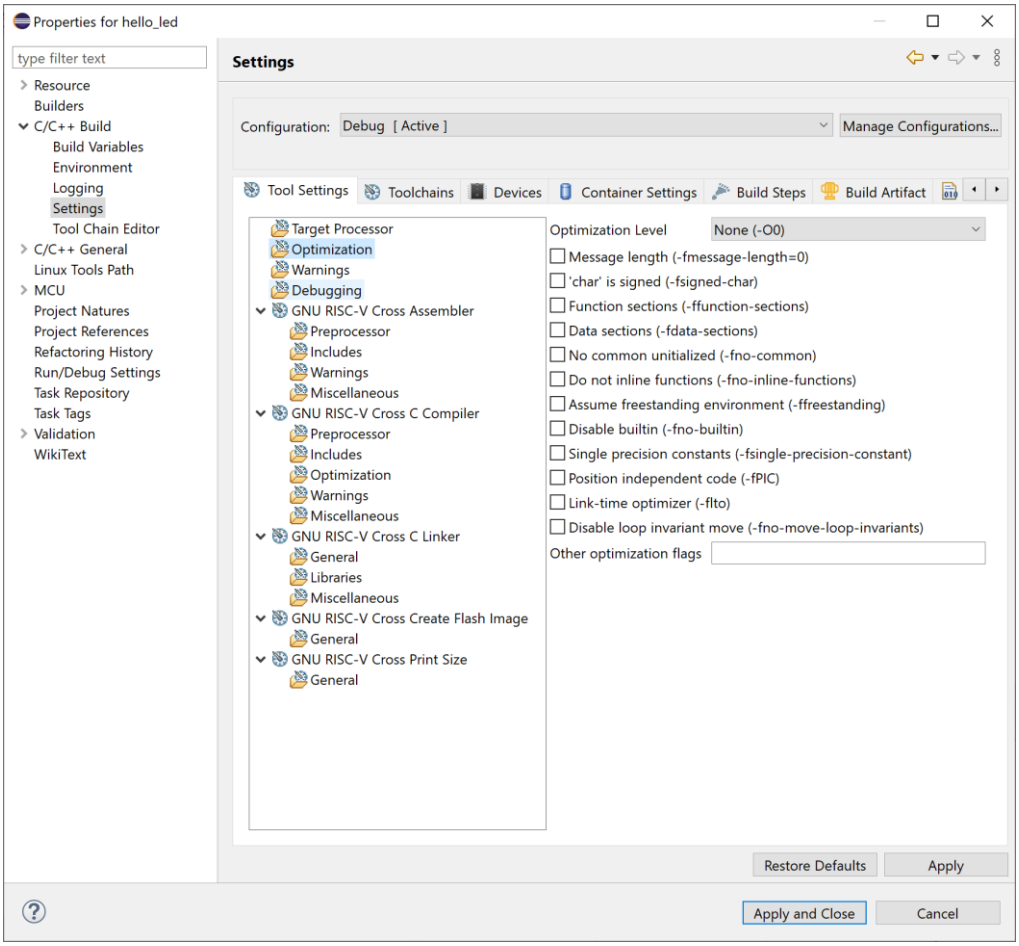

Abbildung 20: Einstellung der Optimization

In der Abbildung 20 werden die Einstellungen zu der Optimierung getätigt. Das Ziel ist eine höhere Effizienz des Zielprogramms mit Änderungen im Programmcode. Die hohe Effizienz wird durch Verringerung des Bedarfs von Zeit oder Speicher oder beides bei der Ausführung des Programms soweit möglich

auf ein Minimum erzielt. Die Optimierung benötigt dafür entsprechend Zeit [44, S. 410]. Die möglichen Einstellungen in der Eclipse IDE sind [41, S. 151-154]:

- *-O1*: Das Ziel in dieser Einstellung des Compilers ist die Reduzierung der Größe des Programmcodes und der Zeit für die Ausführung. Entsprechende Flags zu dieser Optimierungseinstellung werden freigeschaltet;
- *-O2*: In dieser Einstellung werden fast alle möglichen Optimierungen durchgeführt, die keinen Kompromiss zwischen Platz und Geschwindigkeit bedeuten. Sie führt zur Leistungserhöhung des Programmcodes und zu erhöhter Zeit für die Kompilierung. Die entsprechenden Flags zur dieser Optimierungseinstellung werden freigeschaltet;
- *-O3*: Mit dieser Option werden zusätzliche Flags für die Optimierung zur Optimierungseinstellung *-O2* freigeschaltet und eine weitergehende Optimierung durchgeführt;
- *-Os*: Die *-O2* Optimierungseinstellung wird ohne die Optimierungsoption, die zur erhöhten Größe des Programmcodes führt, freigeschaltet;
- *-Og*: Bei dieser Einstellung wird eine Menge von Optimierungen deaktiviert und somit gibt es keine Auswirkungen von Optimierungsoptionen. Die Flags der Optimierungseinstellung *-O1* werden freigeschaltet ohne die Flags, welche die Suche nach Fehlern beeinträchtigen können;
- *-Ofast*: Standartvorgaben werden mit dieser Option nicht eingehalten. Die Flags zur Optimierungseinstellung *-O3* und zusätzliche werden freigeschaltet;
- *-O0*: Die Standardeinstellung ermöglicht die Reduzierung der Zeit für die Kompilierung und wird besonders bei der Suche nach Fehlern verwendet.

In dieser Arbeit wird eine kleine Anwendung, die Demonstration der RGB LED auf das Entwicklungsboard geladen, sodass kein großer Programmcode entsteht und eine Optimierung nicht nötig ist. Die Standarteinstellung *-O0* wird beibehalten um die Kompilierung des Programmcodes ohne Optimierungen durchzuführen. Ein weiterer Begriff im Zusammenhang mit dem Compiler ist der sogenannte Präprozessor oder auch Preprocessor. Unter *GNU RISC-V Cross Assembler* wird der *Preprocessor* durch ein Kontrollkasten aktiviert. Der Präprozessor ist für

die Einbindung von Header Dateien von Bibliotheksfunktionen in das Programm, für die Definition von Makros, das Setzen von Compilereinstellungen und die Angabe von Übersetzungsbedingungen zuständig. Die Anweisungen *#include* und *#define* sind beispielsweise Anweisungen an den Präprozessor. Dies erkennt man am # Symbol [44, S. 619-620]. Unter *Includes* werden unter anderem die Header Dateien von Bibliotheksfunktionen eingebunden (siehe Abbildung 21).

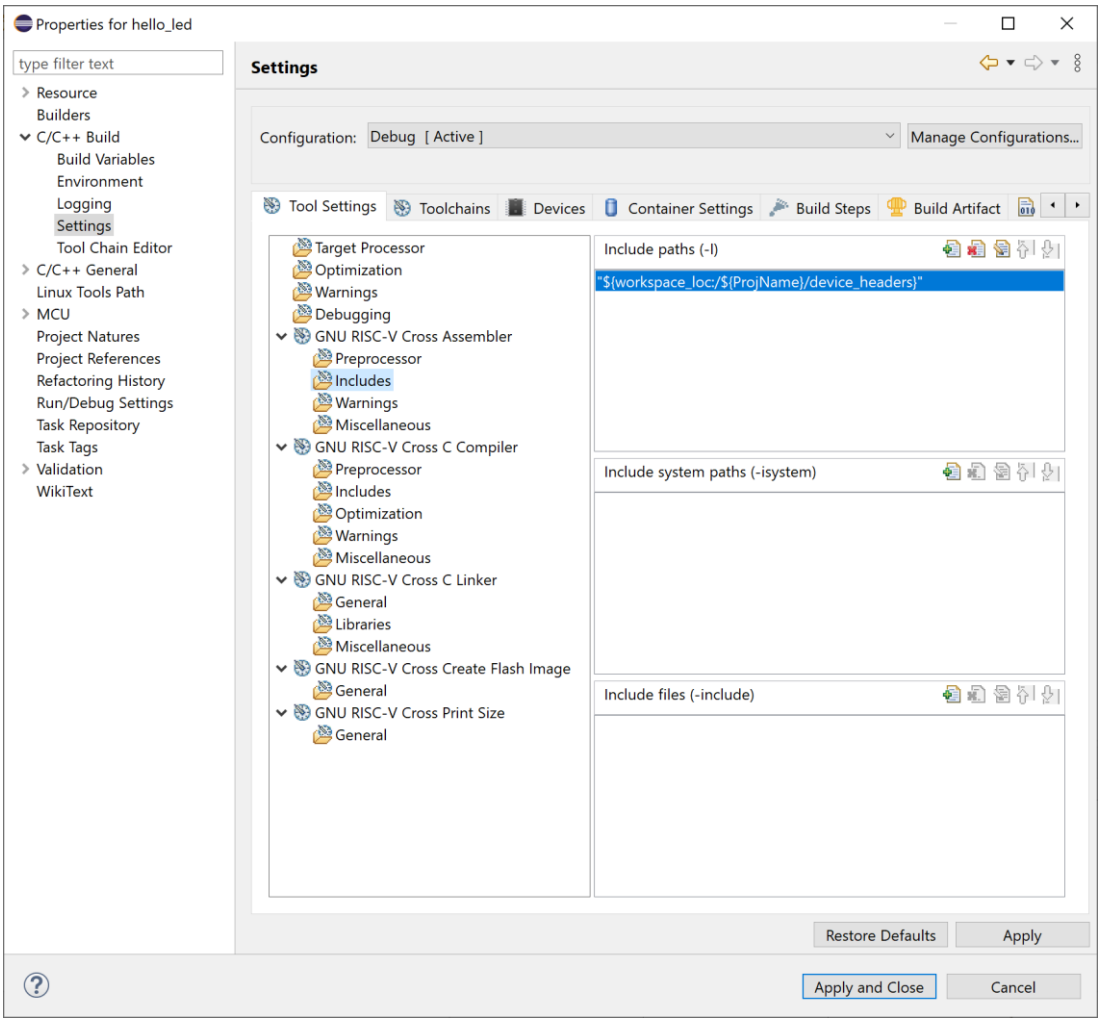

Abbildung 21: Eclipse Header einbinden

Im weiteren Verlauf werden die Einstellungen zum GNU RISC-V Cross C Linker beschrieben. Außerdem wird das Linker-Script, der Startcode und die Vektortabelle allgemein erklärt.

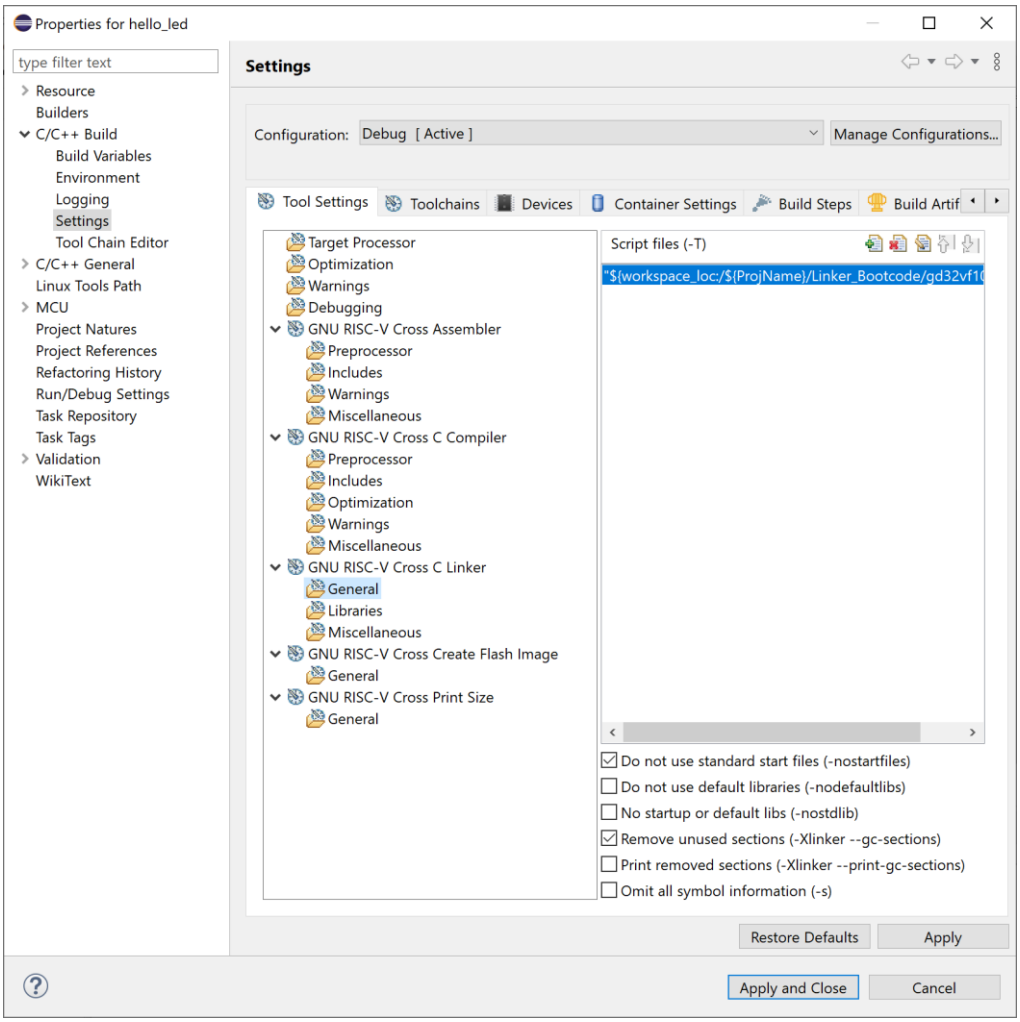

Abbildung 22: Einstellung des GNU RISC-V Cross Linker's

Linkerbefehle werden automatisiert durch das Linker-Script der Befehlszeile übergeben. Das Linker-Script wird dem *GNU RISC-V Cross C Linker* unter *General* zur Verfügung gestellt (siehe Abbildung 22). Die Auswahl *Do not use standard start files* wird ausgewählt, um ein eigenes *start file* zu verwenden. Um die Binärdatei am Ende des Build-Prozesses der Software zu verkleinern, wird *Remove unused sections (-Xlinker--gc-sections)* ausgewählt. Mit dieser Option werden die benötigten Elemente aus Bibliotheken in das Programm von dem Linker eingebunden. Ohne diese Option würde die Binärdatei oder das Programm einen größeren Speicherbedarf haben. Auf einem Embedded System könnte die Binärdatei somit aufgrund von Speichermangel nicht auf den Mikrocontroller übertragen werden [25, S. 87]. Bei dem Programm aus dem Projekt *hello_led* [23, 24] benötigt man diese Option nicht zwingend.

Im Folgenden wird das Linker-Script, der Startcode oder Bootcode, sowie die Vektortabelle zum Projekt *hello_led* allgemein erklärt. Für das *hello_led* Projekt benötigen wir folgende Elemente [22]:

- Linker-Script: Angabe zu den Größen zu den Speichern RAM und Flash-Speicher, sowie die Standorte der Abschnitte werden im Linker-Script vorgenommen,
- Vektortabelle: die Definition von Speicheradressen zum Booten und der Sprungbefehle für Interrupt Handler Funktionen werden gesetzt,
- Reset Handler: Vor dem Hauptprogramm werden elementare Initialisierungsvorgänge vorbereitet.

Diese Elemente werden im Verlauf dieser Arbeit beschrieben. An erster Position in der Beschreibung der Elemente befindet sich das Linker-Script [24]:

```
OUTPUT_ARCH( "riscv" )
ENTRY( reset_handler )
```

Mit *OUTPUT_ARCH* wird entsprechend des Mikrocontrollers die Architektur für die Ausgabe des ausführbaren Programms festgelegt [45, S. 45]. Mit *ENTRY* wird die Ausführung des Programms, also der Einstiegspunkt definiert [45, S. 39]. Bevor das Hauptprogramm *main* ausgeführt wird, startet in unserem Fall der *Reset Handler*, der sich in der Vektortabelle befindet (siehe Anhang).

```
MEMORY
{
  FLASH (rx)  : ORIGIN = 0x08000000, LENGTH = 128K
  RAM   (rwx) : ORIGIN = 0x20000000, LENGTH = 32K
}
```

MEMORY gibt die Größe des Flashspeichers und RAM-Speichers, sowie die Beschreibung der Positionen der Speicheradressen des Mikrocontrollers an [25, S. 100]. Um das Programm vor Veränderungen zu schützen, wird für den Flashspeicher mit *rx* ein Schreibschutz vorgenommen. Der Schreib- und Lesezugriff auf den RAM wird mit *rwx* ermöglicht [46].

```
SECTIONS
{
  __stack_size = DEFINED(__stack_size) ? __stack_size : 1K;

  .vector_table :
  {
    KEEP (*(SORT_NONE(.vector_table)))
  } >FLASH
```

Die Abschnitte zur Vektortabelle, des Datenbereichs *.text*, der initialisierten Variablen *.data* und der nicht initialisierten Variablen *.bss* werden beschrieben. Dabei ist die Vektortabelle im Flashspeicher an erster Position [47].

```
.text :
{
  *(.rodata .rodata.*)
  *(.srodata .srodata.*)
  *(.text .text.*)
} >FLASH

. = ALIGN(4);

PROVIDE (__etext = .);
PROVIDE (_etext = .);
PROVIDE (etext = .);
```

Die Bezeichnung *.text* bekommt den Programmcode in den definierten Abschnittsbereich. Die schreibgeschützten Daten wie Konstanten bekommt der Abschnitt *.rodata*. Dabei zeigt der Buchstabe s den Start und e das Ende des Abschnitts [47].

```
_sidata = .;
.data : AT( _sidata )
{
  _sdata = .;
  *(.rdata)
  *(.data .data.*)
  *(.sdata .sdata.*)
  . = ALIGN(4);
  _edata = .;
} >RAM

PROVIDE( _edata = . );
PROVIDE( edata = . );
PROVIDE( _fbss = . );
PROVIDE( __bss_start = . );
```

Die Bezeichnung *.data* bekommt z.B. veränderbare Variablen. Hier gilt die Erklärung der Buchstaben *s* und *e* ebenfalls [47].

```
.bss :
{
  _sbss = .;
  *(.sbss*)
  *(.bss .bss.*)
  *(COMMON)
  . = ALIGN(4);
  _ebss = .;
} >RAM
```

Der Abschnitt *.bss* wird für das Lesen und Schreiben von Null initialisierten Daten genutzt [45, S. 42, 47]. Dies bedeutet, dass nicht initialisierte Daten erst während

der Ausführung des Programms definiert werden. Dabei wird dem Compiler vom Linker mitgeteilt, dass der dynamische Speicher nicht vollgeschrieben werden soll [25, S. 548]. Der Befehl *.ALIGN(4);* wird für die Prüfung der aktuellen Adresse auf Teilbarkeit durch 4 verwendet, da 32-Bit eines Mikrocontrollers 4 Byte ergeben. Der Punkt vor dem Befehl zeigt die aktuelle Adresse [25, S. 101]. Für die Angabe der Ablage der Abschnitte in einem Adressbereich wird der Befehl *} >RAM* oder *} >FLASH* angegeben [25, S. 102].

```
  . = ALIGN(8);
  PROVIDE( _end = . );
  PROVIDE( end = . );

  .stack ORIGIN(RAM) + LENGTH(RAM) - __stack_size :
  {
    PROVIDE( _heap_end = . );
    . = __stack_size;
    PROVIDE( _sp = . );
  } >RAM
}
```

Bei zu geringem Speicherplatz entsteht ein Fehler durch den Linker. Deshalb wird der gemeinsame Abschnitt für Heap und Stack der dynamischen Speicher auf dem Adressbereich des RAMs im Linker-Script beschrieben [25, S. 550, 47].

Vektortabelle:

Bei Mikrocontrollern wird eine Vektortabelle für die Konfiguration von Interrupts verwendet. Die Funktion eines Interrupts kann bei Erfüllung einer Bedingung angesprochen werden. Der vorherige Programmablauf des Mikrocontrollers wird angehalten, um die Interrupt Service Routine auszuführen.

Die Adresse, an die der Mikrocontroller im Programmablauf springen soll, wenn ein Interrupt erscheint, wird durch die Vektortabelle beschrieben. Die .global-Zeilen ermöglichen, dass die verwendeten Definitionen auch anderen Teilprogrammen, die mit ihnen verknüpft sind, zur Verfügung gestellt werden. Die Vektortabelle beinhaltet den *Reset Handler* für die Definition der Funktion, welche beim Reset der Hardware oder beim Start des Systems ausgeführt wird [46]. Im Programm wird durch den Befehl *.word* ein 4-Byte-Wert abgelegt. Die Beschriftungen in der Vektortabelle werden durch den Compiler mit den Speicheradressen ersetzt [46].

Mit der Anweisung *.weak* und *.set* werden schwache Definitionen für Ausnahmen oder Ausnahmehandler auf den *default-interrupt-handler* verwiesen. Eine

Funktion mit identischer Bezeichnung oder Definition führt zur Überschreibung des Verweises [22, 46, 47].

Der Ausnahmehandler führt die Fehlerbehandlung bei Erscheinen von Ausnahmen durch. Als Ausnahmen werden Fehler in der Ausführung des Programms bezeichnet. Diese stören den Ablauf der Ausführung [48].

Reset Handler (Startcode):

Für den Startvorgang des Hauptprogramms *main* wird der Speicherplatz mit der Reset-Handler-Funktion eingerichtet.

Die Aufgaben der Reset-Handler-Funktion für das Projekt hello_led sind [22, 24]:

- ein Bit für die Deaktivierung der globalen Hardware-Interrupts aus dem CSR (Control and Status Register) der CPU zu entfernen, bis die Anwendung diese wieder konfiguriert,
- das Prüfen, ob das Programm aus der Adresse 0x00000000 oder 0x08000000 (Flashspeicher) läuft, um frühzeitig Fehler in der Ausführung des Programms durch das Springen auf die Adresse 0x08000000 zu vermeiden,
- den Stapelzeiger oder Stack-Pointer (*sp*) auf das Ende des RAMs zu richten,
- den Inhalt des Abschnittes *.data* vom Flashspeicher in den RAM-Speicher zu kopieren,
- den Inhalt des Abschnittes *.bss* zu löschen, um den Inhalt auf null zu setzen und
- das Hauptprogramm *main* aufzurufen.

Das Linker-Script erwartet die Ausführung des Programms aus der Adresse *0x08000000*. Der Adressraum des Mikrocontrollers erlaubt, dass zwei Speicheradressen sich auf die gleiche physische Speicheradresse richten. Dies kann zu Problemen bei der Ausführung des Programms führen. Beim Einschalten des Entwicklungsboards beginnt die Speicheradresse bei *0x00000000*, da der Programm Counter beim Einschalten des Entwicklungsboards und beim Ausführen der Sprunganweisung auf die Adresse *0x00000000* verweist. Durch das Betätigen der *Boot0-Taste* des Boards wird die Adresse *0x00000000* dem Flashspeicher zugeordnet [22]. Das Linker-Script erwartet jedoch die Adresse

0x08000000 und genau dadurch können Probleme bei der Ausführung des Programms entstehen. Deshalb wird ein frühzeitiger Sprung auf die Adresse des Flashspeichers angeordnet. Der nächste Schritt ist es die *globalen Interrupts* zu deaktivieren. Dies wird mit der Entfernung des *MIE-Bits (Machine-mode Interrupts Enabled)* aus dem *MSTATUS-CSR* erreicht. Der Grund für diese Maßnahme ist, dass der Kern des Mikrocontrollers nicht zwingend die Peripheriegeräte bei einem System-Reset zurücksetzt. So können auch Fehler in der Ausführung vermieden werden. Die Speicheradresse der Vektortabelle wird *MTVT CSR* mit der Beschriftung *vtable* zugewiesen. *MTVEC CSR* wird auf *default_interupt_handler* verwiesen, um Speicherbeschädigungen bei der ungewollten Aktivierung des gemeinsamen Interrupt-Handlers zu vermeiden. Die nächsten Schritte mit den Abschnitten *.data* und *.bss* werden im Hauptprogramm durchgeführt [22].

Die Reset-Handler-Funktion ist wie folgt [22, 24]:

```
/*
 * Assembly 'reset handler' function to initialize core CPU registers.
 */
.global reset_handler
.type reset_handler,@function
reset_handler:
  // Disable interrupts until they are needed.
  csrc CSR_MSTATUS, MSTATUS_MIE
  // Move from 0x00000000 to 0x08000000 address space if necessary.
  la   a0, in_address_space
  li   a1, 1
  slli a1, a1, 27
  bleu a1, a0, in_address_space
  add  a0, a0, a1
  jr   a0
  in_address_space:
  // Load the initial stack pointer value.
  la   sp, _sp
  // Set the vector table's base address.
  la   a0, vtable
  csrw CSR_MTVT, a0
  // Set non-vectored interrupts to use the default handler.
  // (That will gracefully crash the program,
  //  so only use vectored interrupts for now.)
  la   a0, default_interrupt_handler
  csrw CSR_MTVEC, a0
  // Call 'main(0,0)' (.data/.bss sections initialized there)
  li   a0, 0
  li   a1, 0
  call main
```

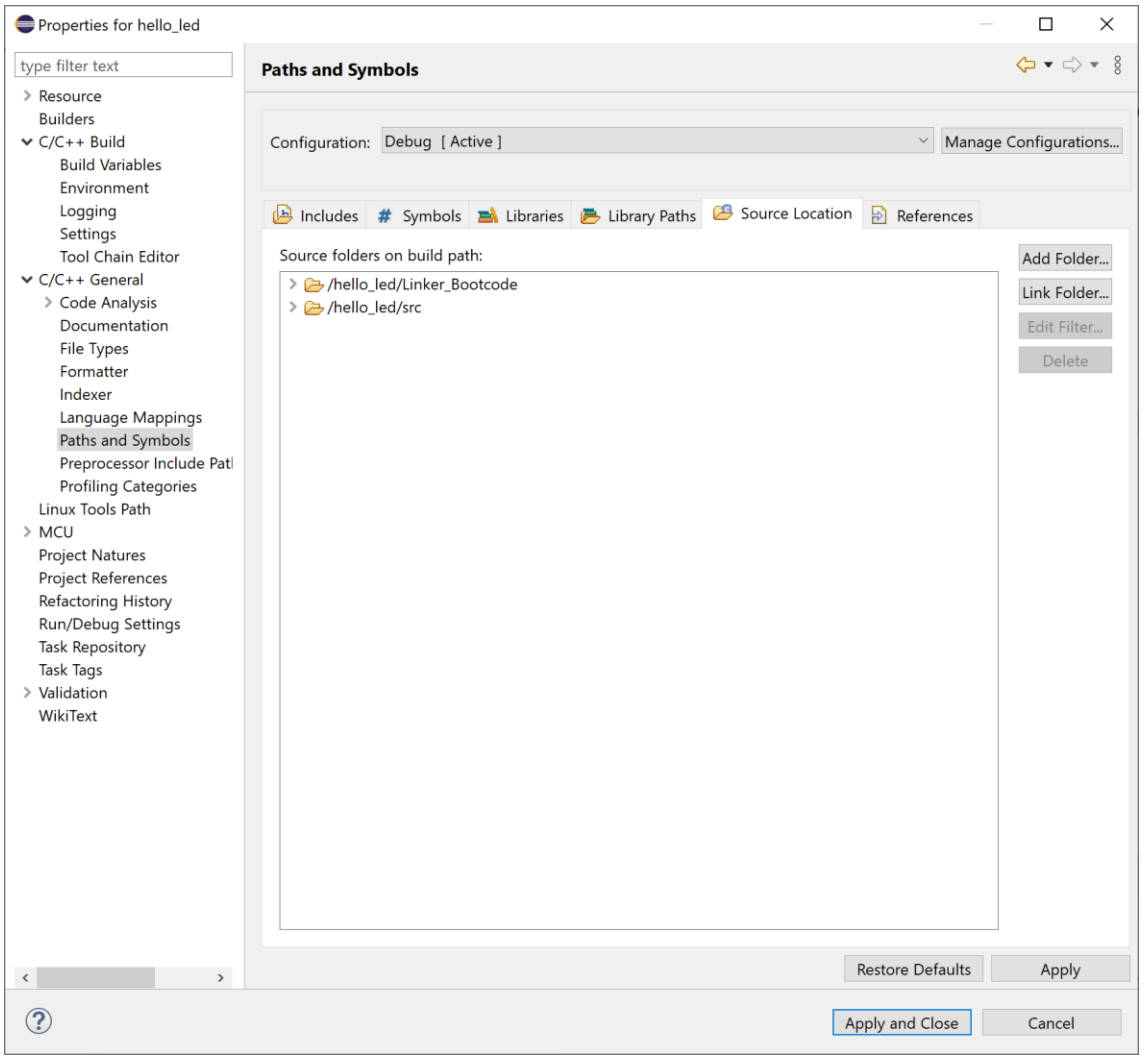

Abbildung 23: Path Probleme beheben

Es kann vorkommen, dass Eclipse einige Verweise, Bibliotheken und Daten nicht finden kann und bei der Erstellung der Software einen Fehler ausgibt. Um dieses Problem zu beheben und zu vermeiden, fügt man alle Projektdateien laut Eclipse unter den Projekteinstellungen *Properties* unter *C/C++ General* bei dem Punkt *Path and Symbols* und unter dem Unterpunkt *Source Location* hinzu (siehe Abbildung 23).

Abbildung 24: Pause button in Eclipse

Der mit einem Pfeil markierte Button in Abbildung 24 muss bei dem Sipeed Longan Nano Entwicklungsboard am Ende des Debuggings gedrückt werden, um manuell das Board zurückzusetzen. Der neue Programmcode wird so ausgeführt. Wenn man versucht, das Board manuell zurückzusetzen, ohne das Debugging zu pausieren, bekommt man den Fehler:

Error: Hart 0 is unavailable.

Info : Hart 0 unexpectedly reset!

Info : In debug mode, Please don't reset target by reset key

Info : To reset target, please click 'restart' button on Eclipse IDE.

Info : Otherwise the breakpoint will be all lose!!

Info : Now, please click pause button, and restart again to re-control the target.

Info : Hart 0 unexpectedly reset!

Info : In debug mode, Please don't reset target by reset key

Info : To reset target, please click 'restart' button on Eclipse IDE.

Info : Otherwise the breakpoint will be all lose!!

Info : Now, please click pause button, and restart again to re-control the target.

Bei erfolgreichem Debugging leuchtet die RGB LED, wie in Abbildung 25 zu sehen ist.

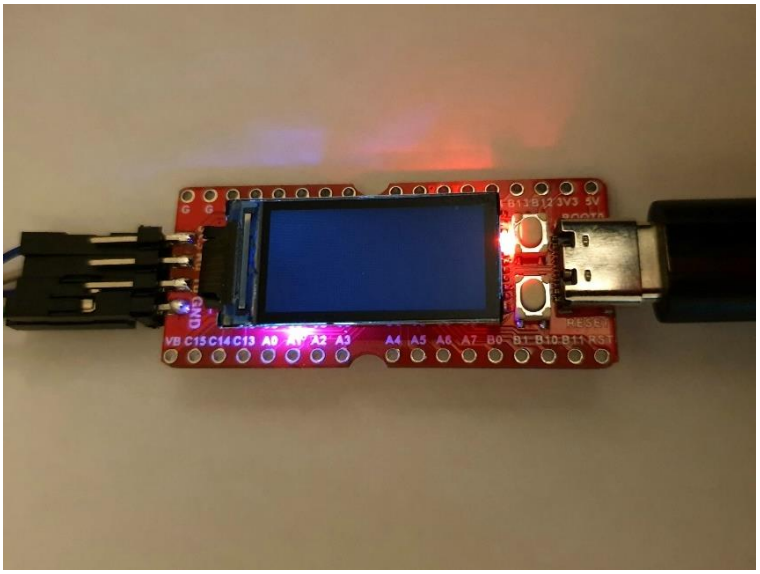

Abbildung 25: Sipeed Longan Nano mit hello_led Projekt

6.3 Beispielprogramm: hello_led

In den nächsten Zeilen wird das Beispielprogramm aus GitHub [22, 23] erklärt.

6.3.1 Headerdateien und Definitionen

```
// Standard library includes.
#include <stdint.h>
#include <string.h>

// Device header files.
#include "gd32vf103.h"
#include "riscv_encoding.h"
```

In den oberen Zeilen werden die Header-Dateien initialisiert. Die Header zum Mikrocontroller werden von GigaDevice zur Verfügung gestellt [22].

```
// Pre-defined memory locations for program initialization.
extern uint32_t _sidata, _sdata, _edata, _sbss, _ebss;
// Current system core clock speed.
volatile uint32_t SystemCoreClock = 8000000;

// Simple 'busy loop' delay method.
__attribute__( ( optimize( "O0" ) ) )
void delay_cycles( uint32_t cyc ) {
  uint32_t d_i;
  for ( d_i = 0; d_i < cyc; ++d_i ) { __asm__( "nop" );
  }
}
```

Vor dem Hauptprogramm *main* werden die Schritte zu *.data* und *.bss* in der Reset-Handler-Funktion durchgeführt. Die Taktsteuerung des Mikrocontrollers wird benötigt, um GPIOx-Peripheriegeräte zu aktivieren. Es wird eine Funktion *delay_cycles* für die Geschwindigkeit, mit der der Wechsel der Farbe der RBG-LED stattfindet, vorgefertigt, um diese später im Hauptprogramm zu nutzen [22].

6.3.2 Hauptprogramm

```
// 'main' method which gets called from the boot code.
int main( void ) {
  // Copy initialized data from .sidata (Flash) to .data (RAM)
  memcpy( &_sdata, &_sidata, ( ( void* )&_edata - ( void* )&_sdata ) );
  // Clear the .bss RAM section.
  memset( &_sbss, 0x00, ( ( void* )&_ebss - ( void* )&_sbss ) );

  // Enable the GPIOA and GPIOC peripherals.
  RCC->APB2ENR |=  ( RCC_APB2ENR_IOPAEN |
                     RCC_APB2ENR_IOPCEN );

  // Configure pins A1, A2, and C13 as low-speed push-pull outputs.
  GPIOA->CRL   &= ~( GPIO_CRL_MODE1 | GPIO_CRL_CNF1 |
                     GPIO_CRL_MODE2 | GPIO_CRL_CNF2 );
  GPIOA->CRL   |=  ( ( 0x2 << GPIO_CRL_MODE1_Pos ) |
                     ( 0x2 << GPIO_CRL_MODE2_Pos ) );
  GPIOC->CRH   &= ~( GPIO_CRH_MODE13 | GPIO_CRH_CNF13 );
```

```
  GPIOC->CRH   |=  ( 0x2 << GPIO_CRH_MODE13_Pos );

  // Turn all three LEDs off.
  // The pins are connected to the LED cathodes, so pulling
  // the pin high turns the LED off, and low turns it on.
  GPIOA->ODR   |=  ( 0x1 << 1 |
                     0x1 << 2 );
  GPIOC->ODR   |=  ( 0x1 << 13 );
```

Die LEDs Rot, Grün und Blau sind mit einer gemeinsamen Anode auf dem Board; die Kathoden hingegen an den Pins *C13 (Rot)*, *A1 (Grün)* und *A2 (Blau)* angeschlossen. Ausgeschaltet werden die LEDs durch einen gemeinsamen High-Pin. Eingeschaltet werden die LEDs mit einem Low-Pin. Die LEDs werden umgeschaltet, indem das Register *ODR (Output Data Register)* beschrieben wird. Das Register *CRL* ist für die Steuerung der Low-Pins (0 bis 7) und das Register *CRH* ist für die Steuerung der High-Pins (8 bis 15) zuständig. Für *Low* wird der 2-Bit-Modus verwendet [22].

```
  // Cycle the RGB LED through a pattern of colors.
  #define DELAY_CYCLES ( 300000 )
  while( 1 ) {
    // Green on (Green)
    GPIOA->ODR &= ~( 0x1 << 1 );
    delay_cycles( DELAY_CYCLES );
    // Red on (Yellow)
    GPIOC->ODR &= ~( 0x1 << 13 );
    delay_cycles( DELAY_CYCLES );
    // Blue on (White)
    GPIOA->ODR &= ~( 0x1 << 2 );
    delay_cycles( DELAY_CYCLES );
    // Green off (Purple)
    GPIOA->ODR |=  ( 0x1 << 1 );
    delay_cycles( DELAY_CYCLES );
    // Red off (Blue)
    GPIOC->ODR |=  ( 0x1 << 13 );
    delay_cycles( DELAY_CYCLES );
    // Blue off (Off)
    GPIOA->ODR |=  ( 0x1 << 2 );
    delay_cycles( DELAY_CYCLES );
  }
  return 0;
}
```

Durch eine While-Schleife werden die LEDs umgeschaltet, indem die Werte im Register *ODR* verändert werden [22].

7 Fazit und Ausblick

Die Inbetriebnahme des RISC-V Mikroprozessors nimmt einige Schritte in Anspruch. Die Installation von Eclipse und der GNU RISC-V Toolchain erfordert bestimmte Kenntnisse. Die GNU RISC-V Toolchain lässt sich mit einer einfachen Möglichkeit über xPack installieren. xPack muss in die Node.js Anwendung integriert und entsprechend durch einen Befehl in der Befehlszeile installiert werden. Anschließend wird die Toolchain mit einem Befehl in der Befehlszeile installiert. Unter Eclipse müssen alle Einstellungen zum Mikrocontroller, d.h. der Toolchain vorgenommen werden. Auch die Einstellungen zum Debugger in OpenOCD sind entsprechend der Zielhardware zu erledigen. Andernfalls kann das Entwicklungsboard Sipeed Longan Nano nicht richtig mit dem System für die Programmierung verbunden werden.

Nachdem die Installation der Entwicklungsumgebung und der GNU RISC-V Toolchain erfolgreich abgeschlossen ist, wird das Beispielprogramm zur Demonstration der RGB LED geladen. Für diesen Schritt sind an Eclipse die entsprechenden Einstellungen in der GNU RISC-V Toolchain für den Compiler, Linker und Debugger zu übergeben. Das Beispielprogramm schaltet eine Reihe von LEDs, die sich an derselben Position auf dem Entwicklungsboard befinden.

Die Programmierung erfordert einiges an Zeit, da das Entwicklungsboard bestimmte Funktionen durch GigaDevice bekommen hat, welche man zuerst erlenen muss. Diese stehen in dem Handbuch zum eingesetzten Mikrocontroller der GD32VF103 Serie.

Zusammengefasst lässt sich der Sipeed Longan Nano nicht ohne weitere Kenntnisse und ohne Zeitaufwand in Betrieb nehmen.

Für die Zukunft wird die RISC-V Architektur eine spannende Alternative für vielerlei Zwecke darstellen. Schon heute lässt sich in den Medien lesen, dass große Konzerne sich mit dieser Architektur beschäftigen und schon einige Mikrocontroller sogar im Einsatz sind. Die Vorteile liegen auf der Hand. Die RISC-V Architektur fordert unter anderem keine Lizenzgebühr und sie ist ressourcenschonender in der Herstellung, sowie im Betrieb als z.B. die CISC-Architektur. Diese Eigenschaften werden mit zunehmender Zeit aufgrund von globalen Problemen wie der Ressourcenknappheit immer wichtiger.

8 Literatur

[1] RS Components GmbH, *Alles, was Sie über Mikrocontroller wissen müssen | RS Components | RS Components.* [Online]. Verfügbar unter: https://de.rs-online.com/web/generalDisplay.html?id=ideen-und-tipps/mikrocontroller-leitfaden (Zugriff am: 13. Dezember 2021).

[2] U. Ostler und A. Rüdiger, lic.rer.publ., „Was ist RISC? Und was ist CISC?“, *DataCenter-Insider*, 5. Juli 2019, 2019. [Online]. Verfügbar unter: https://www.datacenter-insider.de/was-ist-risc-und-was-ist-cisc-a-842995/. Zugriff am: 9. März 2021.

[3] RISC-V International, *Frequently Asked Questions (FAQ) - RISC-V International.* [Online]. Verfügbar unter: https://riscv.org/about/faq/ (Zugriff am: 24. Februar 2021).

[4] openocd, *Open On-Chip Debugger.* [Online]. Verfügbar unter: https://openocd.org/ (Zugriff am: 15. Dezember 2021).

[5] RISC-V International, *History - RISC-V International.* [Online]. Verfügbar unter: https://riscv.org/about/history/ (Zugriff am: 24. Februar 2021).

[6] RISC-V International, *About RISC-V - RISC-V International.* [Online]. Verfügbar unter: https://riscv.org/about/ (Zugriff am: 24. Februar 2021).

[7] RISC-V International, *Membership - RISC-V International.* [Online]. Verfügbar unter: https://riscv.org/membership/ (Zugriff am: 24. Februar 2021).

[8] RISC-V International, *Branding Guidelines - RISC-V International.* [Online]. Verfügbar unter: https://riscv.org/about/risc-v-branding-guidelines/ (Zugriff am: 24. Februar 2021).

[9] U. Brinkschulte und T. Ungerer, *Mikrocontroller und Mikroprozessoren*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2010, ISBN 978-3-642-05398-6.

[10] K. Wüst, *Mikroprozessortechnik: Grundlagen Architekturen Schaltungstechnik und Betrieb von Mikroprozessoren und Mikrocontrollern ; mit 44 Tabellen*. Wiesbaden: Vieweg + Teubner, 2011, ISBN 978-3-8348-0906-3.

[11] C. Baun, *Betriebssysteme kompakt*. Berlin, Heidelberg: Springer Berlin Heidelberg; Imprint: Springer Vieweg, 2017, ISBN 978-3-662-53143-3.

[12] H. Bernstein, *Mikrocontroller: Grundlagen der Hard- und Software der Mikrocontroller ATtiny2313 ATtiny26 und ATmega32*. Wiesbaden: Springer Vieweg, 2020, ISBN 978-3-658-30066-1.

[13] K. Warschkow, *Bussysteme.* [Online]. Verfügbar unter: https://www.technik-kiste.de/wissen/computer/aufbau/9-bussysteme (Zugriff am: 15. Dezember 2021).

[14] Sipeed, *Introduction · Longan DOC.* [Online]. Verfügbar unter: http://longan.sipeed.com/en/ (Zugriff am: 25. Februar 2021).

[15] seeedstudio, *Sipeed Longan Nano - RISC-V GD32VF103CBT6 Development Board - Seeed Studio.* [Online]. Verfügbar unter: https://www.seeedstudio.com/Sipeed-Longan-Nano-RISC-V-GD32VF103CBT6-Development-Board-p-4205.html (Zugriff am: 26. Februar 2021).

[16] 王南飞, *GD32VF103_User_Manual_EN_V1.2.* [Online]. Verfügbar unter: http://www.gd32mcu.com/data/documents/shujushouce/GD32VF103_User_Manual_EN_V1.2.pdf (Zugriff am: 10. April 2021).

[17] A. Reber, F. Kesel und J. Hampel, *Aufgabe 1 - SysTick und ADC*. Labor Mikrocontroller mit NUC130. [Online]. Verfügbar unter: http://eitidaten.fh-pforzheim.de/daten/labore/mclt/pdf/mc_eit_labor1.pdf (Zugriff am: 12. April 2021).

[18] ComputerWeekly.de und TechTarget, *Parität.* [Online]. Verfügbar unter: https://www.computerweekly.com/de/definition/Paritaet (Zugriff am: 13. Dezember 2021).

[19] GigaDevice, *GD32VF103_Datasheet_Rev_1.2.* [Online]. Verfügbar unter: https://www.gigadevice.com/datasheet/gd32vf103xxxx-datasheet/ (Zugriff am: 10. April 2021).

[20] M. Barr, *Programming embedded systems with C and GNU development tools: With C and GNU development tools,* 2. Aufl. Beijing [u.a.]: O'Reilly, 2006, ISBN 978-0-596-00983-0.

[21] P. Gliwa, *Embedded Software Timing: Methodik Analyse und Praxistipps am Beispiel Automotive*. Wiesbaden: Springer Vieweg, 2021. [Online]. Verfügbar unter: https://link.springer.com/content/pdf/10.1007%2F978-3-658-26480-2.pdf, ISBN 978-3-658-26479-6.

[22] Vivonomicon, *Bare-metal RISC-V Development with the GD32VF103CB – Vivonomicon's Blog.* [Online]. Verfügbar unter: https://vivonomicon.com/2020/02/11/bare-metal-risc-v-development-with-the-gd32vf103cb/ (Zugriff am: 16. Oktober 2021).

[23] WRansohoff, *GitHub - WRansohoff/GD32VF103_templates at 74111acdb2ba5fca6e860e95a76129f60fa68100.* [Online]. Verfügbar unter: https://github.com/WRansohoff/GD32VF103_templates/tree/74111acdb2ba5fca6e860e95a76129f60fa68100/hello_led (Zugriff am: 28. November 2021).

[24] WRansohoff, *GD32VF103_templates/gd32vf103xb.ld at master · WRansohoff/GD32VF103_templates.* [Online]. Verfügbar unter: https://github.com/WRansohoff/GD32VF103_templates/tree/74111acdb2ba5fca6e860e95a76129f60fa68100 (Zugriff am: 28. November 2021).

[25] R. Jesse, *ARM Cortex-M3 Mikrocontroller: Einstieg und Praxis*. Frechen: mitp-Verlag, 2014, ISBN 978-3-8266-9600-8.

[26] M. Milinkovich, *About the Eclipse Foundation | The Eclipse Foundation.* [Online]. Verfügbar unter: https://www.eclipse.org/org/ (Zugriff am: 13. Dezember 2021).

[27] Eclipse Foundation, *Eclipse Installer 2021-09 R | Eclipse Packages.* [Online]. Verfügbar unter: https://www.eclipse.org/downloads/packages/installer (Zugriff am: 23. September 2021).

[28] ComputerWeekly.de und TechTarget, *Was ist Kommandozeile (Command Line Interface, CLI)? - Definition von WhatIs.com.* [Online]. Verfügbar unter: https://whatis.techtarget.com/de/definition/Kommandozeile-Command-Line-Interface-CLI (Zugriff am: 13. Dezember 2021).

[29] L. Ionescu, *The xPack Project Manager.* [Online]. Verfügbar unter: https://xpack.github.io/xpm/ (Zugriff am: 28. November 2021).

[30] S. Augsten und MirkoK, „Was ist Node.js?“, *Dev-Insider*, 8. Dez. 2020, 2020. [Online]. Verfügbar unter: https://www.dev-insider.de/was-ist-nodejs-a-972703/. Zugriff am: 29. November 2021.

[31] S. Augsten und Gedeon Rauch, „Was bedeutet LTS?“, *Dev-Insider*, 11. Sep. 2020, 2020. [Online]. Verfügbar unter: https://www.dev-insider.de/was-bedeutet-lts-a-959219/. Zugriff am: 28. November 2021.

[32] Node.js, OpenJS Foundation und Joyent, Inc., *Node.js.* [Online]. Verfügbar unter: https://nodejs.org/en/ (Zugriff am: 28. November 2021).

[33] L. Ionescu, *How to install xpm.* [Online]. Verfügbar unter: https://xpack.github.io/xpm/install/ (Zugriff am: 29. November 2021).

[34] L. Ionescu, *How to install the xPack GNU RISC-V Embedded GCC binaries.* [Online]. Verfügbar unter: https://xpack.github.io/riscv-none-embed-gcc/install/ (Zugriff am: 29. November 2021).

[35] L. Ionescu, *The xPack GNU RISC-V Embedded GCC.* [Online]. Verfügbar unter: https://xpack.github.io/riscv-none-embed-gcc/ (Zugriff am: 29. November 2021).

[36] L. Ionescu, *How to install the xPack Windows Build Tools binaries.* [Online]. Verfügbar unter: https://xpack.github.io/windows-build-tools/install/ (Zugriff am: 29. November 2021).

[37] Nuclei System Technology, *DOCS & TOOLS_Nuclei-Best RISC-V Processor IP.* [Online]. Verfügbar unter: https://www.nucleisys.com/download.php (Zugriff am: 29. November 2021).

[38] WsHelloWorld und H. Fang, *GD32VF103_Firmware_Library/Template at master · riscv-mcu/GD32VF103_Firmware_Library.* [Online]. Verfügbar unter: https://github.com/riscv-mcu/GD32VF103_Firmware_Library/blob/afd33c1a1d324af44497471408eba4ef24fe8669/Template/openocd_ft2232.cfg (Zugriff am: 30. November 2021).

[39] W. Andrew *et al., The RISC-V Instruction Set Manual: Volume I: Unprivileged ISA*. Document Version 20191213. [Online]. Verfügbar unter: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf (Zugriff am: 26. September 2021).

[40] P. Dabbelt, *All Aboard, Part 1: The -march, -mabi, and -mtune arguments to RISC-V Compilers - SiFive.* [Online]. Verfügbar unter: https://www.sifive.com/blog/all-aboard-part-1-compiler-args (Zugriff am: 1. Dezember 2021).

[41] R. M. Stallman und GCC Developer Community, *Using the GNU Compiler Collection: For gcc version 11.2.0*. (GCC). [Online]. Verfügbar unter: https://gcc.gnu.org/onlinedocs/gcc-11.2.0/gcc.pdf (Zugriff am: 2. Dezember 2021).

[42] Z. Jorquera, *What is an ABI?* | *Engineering Education (EngEd) Program* | *Section.* [Online]. Verfügbar unter: https://www.section.io/engineering-education/what-is-an-abi/ (Zugriff am: 30. November 2021).

[43] M. Puff, *Aufrufkonventionen.* [Online]. Verfügbar unter: http://michael-puff.de/Programmierung/Artikel/Aufrufkonventionen.shtml (Zugriff am: 6. Dezember 2021).

[44] H. Ernst, J. Schmidt und G. H. Beneken, *Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis ; eine umfassende praxisorientierte Einführung // Grundlagen und Konzepte für die erfolgreiche IT-Praxis - Eine umfassende praxisorientierte Einführung,* 5. Aufl. Wiesbaden: Springer Vieweg; Springer Fachmedien Wiesbaden, 2015. [Online]. Verfügbar unter: https://link.springer.com/content/pdf/10.1007%2F978-3-658-01628-9.pdf, ISBN 978-3-658-01628-9.

[45] S. Chamberlain und I. L. Taylor, *The GNU linker*. ld (Sourcery G++ Lite 2010q1-188) Version 2.19.51. [Online]. Verfügbar unter: https://www.eecs.umich.edu/courses/eecs373/readings/Linker.pdf (Zugriff am: 9. Dezember 2021).

[46] Vivonomicon, *“Bare Metal” STM32 Programming (Part 1): Hello, ARM! – Vivonomicon's Blog.* [Online]. Verfügbar unter: https://vivonomicon.com/2018/04/02/bare-metal-stm32-programming-part-1-hello-arm/ (Zugriff am: 13. Dezember 2021).

[47] Vivonomicon, *“Bare Metal” STM32 Programming (Part 2): Making it to ‘Main’ – Vivonomicon's Blog.* [Online]. Verfügbar unter: https://vivonomicon.com/2018/04/20/bare-metal-stm32-programming-part-2-making-it-to-main/ (Zugriff am: 13. Dezember 2021).

[48] B. Allen, „Unterschied zwischen Exception-Handler und Event-Handler“, *Barry Allen*, 2. Aug. 2020, 2020. [Online]. Verfügbar unter: https://gocoding.org/de/difference-between-exception-handler-and-event-handler/. Zugriff am: 14. Dezember 2021.

[49] Sipeed, *下载站 - Sipeed.* [Online]. Verfügbar unter: https://dl.sipeed.com/shareURL/LONGAN/Nano/HDK/Longan%20Nano%203302/2_Schematic.

Anhang

Vektortabelle gd32vf103xb_boot.S aus dem GitHub Repository [23]:

```
#include "riscv_encoding.h"

/*
 * Main vector table entries.
 */
.global vtable
.type vtable, %object
.section .vector_table,"a",%progbits
vtable:
  j reset_handler
  .align 2
  .word 0
  .word 0
  .word eclic_msip_handler
  .word 0
  .word 0
  .word 0
  .word eclic_mtip_handler
  .word 0
  .word 0
  .word 0
  .word 0
  .word 0
  .word 0
  .word 0
  .word 0
  .word eclic_bwei_handler
  .word eclic_pmovi_handler
  .word watchdog_IRQn_handler
  .word LVD_IRQn_handler
  .word tamper_IRQn_handler
  .word RTC_IRQn_handler
  .word FMC_IRQn_handler
  .word RCU_IRQn_handler
  .word EXTI0_IRQn_handler
  .word EXTI1_IRQn_handler
  .word EXTI2_IRQn_handler
  .word EXTI3_IRQn_handler
  .word EXTI4_IRQn_handler
  .word DMA0_chan0_IRQn_handler
  .word DMA0_chan1_IRQn_handler
  .word DMA0_chan2_IRQn_handler
  .word DMA0_chan3_IRQn_handler
  .word DMA0_chan4_IRQn_handler
  .word DMA0_chan5_IRQn_handler
  .word DMA0_chan6_IRQn_handler
  .word ADC0_1_IRQn_handler
  .word CAN0_TX_IRQn_handler
  .word CAN0_RX0_IRQn_handler
  .word CAN0_RX1_IRQn_handler
  .word CAN0_EWMC_IRQn_handler
  .word EXTI5_9_IRQn_handler
  .word TIM0_break_IRQn_handler
  .word TIM0_update_IRQn_handler
```

```
.word TIM0_trigger_commutation_IRQn_handler
.word TIM0_channel_IRQn_handler
.word TIM1_IRQn_handler
.word TIM2_IRQn_handler
.word TIM3_IRQn_handler
.word I2C0_EV_IRQn_handler
.word I2C0_ER_IRQn_handler
.word I2C1_EV_IRQn_handler
.word I2C1_ER_IRQn_handler
.word SPI0_IRQn_handler
.word SPI1_IRQn_handler
.word USART0_IRQn_handler
.word USART1_IRQn_handler
.word USART2_IRQn_handler
.word EXTI10_15_IRQn_handler
.word RTC_alarm_IRQn_handler
.word USB_wakeup_IRQn_handler
.word 0
.word 0
.word 0
.word 0
.word 0
.word EXMC_IRQn_handler
.word 0
.word TIM4_IRQn_handler
.word SPI2_IRQn_handler
.word UART3_IRQn_handler
.word UART4_IRQn_handler
.word TIM5_IRQn_handler
.word TIM6_IRQn_handler
.word DMA1_chan0_IRQn_handler
.word DMA1_chan1_IRQn_handler
.word DMA1_chan2_IRQn_handler
.word DMA1_chan3_IRQn_handler
.word DMA1_chan4_IRQn_handler
.word 0
.word 0
.word CAN1_TX_IRQn_handler
.word CAN1_RX0_IRQn_handler
.word CAN1_RX1_IRQn_handler
.word CAN1_EWMC_IRQn_handler
.word USB_IRQn_handler

/*
 * Weak aliases to point each exception hadnler to the
 * 'default_interrupt_handler', unless the application defines
 * a function with the same name to override the reference.
 */
.weak eclic_msip_handler
.set  eclic_msip_handler,default_interrupt_handler
.weak eclic_mtip_handler
.set  eclic_mtip_handler,default_interrupt_handler
.weak eclic_bwei_handler
.set  eclic_bwei_handler,default_interrupt_handler
.weak eclic_pmovi_handler
.set  eclic_pmovi_handler,default_interrupt_handler
.weak watchdog_IRQn_handler
.set  watchdog_IRQn_handler,default_interrupt_handler
.weak LVD_IRQn_handler
.set  LVD_IRQn_handler,default_interrupt_handler
.weak tamper_IRQn_handler
```

```
.set  tamper_IRQn_handler,default_interrupt_handler
.weak RTC_IRQn_handler
.set  RTC_IRQn_handler,default_interrupt_handler
.weak FMC_IRQn_handler
.set  FMC_IRQn_handler,default_interrupt_handler
.weak RCU_IRQn_handler
.set  RCU_IRQn_handler,default_interrupt_handler
.weak EXTI0_IRQn_handler
.set  EXTI0_IRQn_handler,default_interrupt_handler
.weak EXTI1_IRQn_handler
.set  EXTI1_IRQn_handler,default_interrupt_handler
.weak EXTI2_IRQn_handler
.set  EXTI2_IRQn_handler,default_interrupt_handler
.weak EXTI3_IRQn_handler
.set  EXTI3_IRQn_handler,default_interrupt_handler
.weak EXTI4_IRQn_handler
.set  EXTI4_IRQn_handler,default_interrupt_handler
.weak DMA0_chan0_IRQn_handler
.set  DMA0_chan0_IRQn_handler,default_interrupt_handler
.weak DMA0_chan1_IRQn_handler
.set  DMA0_chan1_IRQn_handler,default_interrupt_handler
.weak DMA0_chan2_IRQn_handler
.set  DMA0_chan2_IRQn_handler,default_interrupt_handler
.weak DMA0_chan3_IRQn_handler
.set  DMA0_chan3_IRQn_handler,default_interrupt_handler
.weak DMA0_chan4_IRQn_handler
.set  DMA0_chan4_IRQn_handler,default_interrupt_handler
.weak DMA0_chan5_IRQn_handler
.set  DMA0_chan5_IRQn_handler,default_interrupt_handler
.weak DMA0_chan6_IRQn_handler
.set  DMA0_chan6_IRQn_handler,default_interrupt_handler
.weak ADC0_1_IRQn_handler
.set  ADC0_1_IRQn_handler,default_interrupt_handler
.weak CAN0_TX_IRQn_handler
.set  CAN0_TX_IRQn_handler,default_interrupt_handler
.weak CAN0_RX0_IRQn_handler
.set  CAN0_RX0_IRQn_handler,default_interrupt_handler
.weak CAN0_RX1_IRQn_handler
.set  CAN0_RX1_IRQn_handler,default_interrupt_handler
.weak CAN0_EWMC_IRQn_handler
.set  CAN0_EWMC_IRQn_handler,default_interrupt_handler
.weak EXTI5_9_IRQn_handler
.set  EXTI5_9_IRQn_handler,default_interrupt_handler
.weak TIM0_break_IRQn_handler
.set  TIM0_break_IRQn_handler,default_interrupt_handler
.weak TIM0_update_IRQn_handler
.set  TIM0_update_IRQn_handler,default_interrupt_handler
.weak TIM0_trigger_commutation_IRQn_handler
.set  TIM0_trigger_commutation_IRQn_handler,default_interrupt_handler
.weak TIM0_channel_IRQn_handler
.set  TIM0_channel_IRQn_handler,default_interrupt_handler
.weak TIM1_IRQn_handler
.set  TIM1_IRQn_handler,default_interrupt_handler
.weak TIM2_IRQn_handler
.set  TIM2_IRQn_handler,default_interrupt_handler
.weak TIM3_IRQn_handler
.set  TIM3_IRQn_handler,default_interrupt_handler
.weak I2C0_EV_IRQn_handler
.set  I2C0_EV_IRQn_handler,default_interrupt_handler
.weak I2C0_ER_IRQn_handler
.set  I2C0_ER_IRQn_handler,default_interrupt_handler
```

```
  .weak I2C1_EV_IRQn_handler
  .set  I2C1_EV_IRQn_handler,default_interrupt_handler
  .weak I2C1_ER_IRQn_handler
  .set  I2C1_ER_IRQn_handler,default_interrupt_handler
  .weak SPI0_IRQn_handler
  .set  SPI0_IRQn_handler,default_interrupt_handler
  .weak SPI1_IRQn_handler
  .set  SPI1_IRQn_handler,default_interrupt_handler
  .weak USART0_IRQn_handler
  .set  USART0_IRQn_handler,default_interrupt_handler
  .weak USART1_IRQn_handler
  .set  USART1_IRQn_handler,default_interrupt_handler
  .weak USART2_IRQn_handler
  .set  USART2_IRQn_handler,default_interrupt_handler
  .weak EXTI10_15_IRQn_handler
  .set  EXTI10_15_IRQn_handler,default_interrupt_handler
  .weak RTC_alarm_IRQn_handler
  .set  RTC_alarm_IRQn_handler,default_interrupt_handler
  .weak USB_wakeup_IRQn_handler
  .set  USB_wakeup_IRQn_handler,default_interrupt_handler
  .weak EXMC_IRQn_handler
  .set  EXMC_IRQn_handler,default_interrupt_handler
  .weak TIM4_IRQn_handler
  .set  TIM4_IRQn_handler,default_interrupt_handler
  .weak SPI2_IRQn_handler
  .set  SPI2_IRQn_handler,default_interrupt_handler
  .weak UART3_IRQn_handler
  .set  UART3_IRQn_handler,default_interrupt_handler
  .weak UART4_IRQn_handler
  .set  UART4_IRQn_handler,default_interrupt_handler
  .weak TIM5_IRQn_handler
  .set  TIM5_IRQn_handler,default_interrupt_handler
  .weak TIM6_IRQn_handler
  .set  TIM6_IRQn_handler,default_interrupt_handler
  .weak DMA1_chan0_IRQn_handler
  .set  DMA1_chan0_IRQn_handler,default_interrupt_handler
  .weak DMA1_chan1_IRQn_handler
  .set  DMA1_chan1_IRQn_handler,default_interrupt_handler
  .weak DMA1_chan2_IRQn_handler
  .set  DMA1_chan2_IRQn_handler,default_interrupt_handler
  .weak DMA1_chan3_IRQn_handler
  .set  DMA1_chan3_IRQn_handler,default_interrupt_handler
  .weak DMA1_chan4_IRQn_handler
  .set  DMA1_chan4_IRQn_handler,default_interrupt_handler
  .weak CAN1_TX_IRQn_handler
  .set  CAN1_TX_IRQn_handler,default_interrupt_handler
  .weak CAN1_RX0_IRQn_handler
  .set  CAN1_RX0_IRQn_handler,default_interrupt_handler
  .weak CAN1_RX1_IRQn_handler
  .set  CAN1_RX1_IRQn_handler,default_interrupt_handler
  .weak CAN1_EWMC_IRQn_handler
  .set  CAN1_EWMC_IRQn_handler,default_interrupt_handler
  .weak USB_IRQn_handler
  .set  USB_IRQn_handler,default_interrupt_handler

/*
 * A 'default' interrupt handler, in case an interrupt triggers
 * without a handler being defined.
 */
.section .text.default_interrupt_handler,"ax",%progbits
default_interrupt_handler:
```

```
    default_interrupt_loop:
      j default_interrupt_loop

/*
 * Assembly 'reset handler' function to initialize core CPU registers.
 */
.global reset_handler
.type reset_handler,@function
reset_handler:
  // Disable interrupts until they are needed.
  csrc CSR_MSTATUS, MSTATUS_MIE
  // Move from 0x00000000 to 0x08000000 address space if necessary.
  la   a0, in_address_space
  li   a1, 1
  slli a1, a1, 27
  bleu a1, a0, in_address_space
  add  a0, a0, a1
  jr   a0
  in_address_space:
  // Load the initial stack pointer value.
  la   sp, _sp
  // Set the vector table's base address.
  la   a0, vtable
  csrw CSR_MTVT, a0
  // Set non-vectored interrupts to use the default handler.
  // (That will gracefully crash the program,
  //  so only use vectored interrupts for now.)
  la   a0, default_interrupt_handler
  csrw CSR_MTVEC, a0
  // Call 'main(0,0)' (.data/.bss sections initialized there)
  li   a0, 0
  li   a1, 0
  call main
```

Device Header gd32vf103.h aus dem GitHub Repository [23]:

```
#ifndef GD32VF103xB
#define GD32VF103xB
/*
 * GD32VF103CB device header file.
 * Written to be as compatible as possible with STM32F1 code.
 * These definitions are incomplete and application-specific.
 */
#include <stdint.h>

/**
 * Compatibility definitions and RISC-V specific values.
 */
extern volatile uint32_t SystemCoreClock;

/**
 * Interrupt handler enumeration.
 */
typedef enum IRQn {
  CLIC_INT_RESERVED = 0,
  CLIC_INT_SFT = 3,
  CLIC_INT_TMR = 7,
  CLIC_INT_BWEI = 17,
  CLIC_INT_PMOVI = 18,
```

```
WWDG_IRQn = 19,
PVD_IRQn = 20,
TAMPER_IRQn = 21,
RTC_IRQn = 22,
FLASH_IRQn = 23,
RCC_IRQn = 24,
EXTI0_IRQn = 25,
EXTI1_IRQn = 26,
EXTI2_IRQn = 27,
EXTI3_IRQn = 28,
EXTI4_IRQn = 29,
DMA1_Channel1_IRQn = 30,
DMA1_Channel2_IRQn = 31,
DMA1_Channel3_IRQn = 32,
DMA1_Channel4_IRQn = 33,
DMA1_Channel5_IRQn = 34,
DMA1_Channel6_IRQn = 35,
DMA1_Channel7_IRQn = 36,
ADC1_2_IRQn = 37,
// These two interrupts aren't connected to the USB peripheral
// on GD32 chips, but names are maintained for STM32F1 compatibility.
USB_HP_CAN1_TX_IRQn = 38,
CAN1_TX_IRQn = 38,
USB_LP_CAN1_RX0_IRQn = 39,
CAN1_RX0_IRQn = 39,
CAN1_RX1_IRQn = 40,
CAN1_SCE_IRQn = 41,
EXTI9_5_IRQn = 42,
TIM1_BRK_IRQn = 43,
TIM1_UP_IRQn = 44,
TIM1_TRG_COM_IRQn = 45,
TIM1_CC_IRQn = 46,
TIM2_IRQn = 47,
TIM3_IRQn = 48,
TIM4_IRQn = 49,
I2C1_EV_IRQn = 50,
I2C1_ER_IRQn = 51,
I2C2_EV_IRQn = 52,
I2C2_ER_IRQn = 53,
SPI1_IRQn = 54,
SPI2_IRQn = 55,
USART1_IRQn = 56,
USART2_IRQn = 57,
USART3_IRQn = 58,
EXTI15_10_IRQn = 59,
RTC_Alarm_IRQn = 60,
USBWakeUp_IRQn = 61,
// Extra interrupts not available on STM32F1 chips:
// (Numbered peripherals are still 1-indexed)
EXMC_IRQn = 67,
TIM5_IRQn = 69,
SPI3_IRQn = 70,
UART4_IRQn = 71,
UART5_IRQn = 72,
TIM6_IRQn = 73,
TIM7_IRQn = 74,
DMA2_Channel1_IRQn = 75,
DMA2_Channel2_IRQn = 76,
DMA2_Channel3_IRQn = 77,
DMA2_Channel4_IRQn = 78,
DMA2_Channel5_IRQn = 79,
```

```
  CAN2_TX_IRQn = 82,
  CAN2_RX0_IRQn = 83,
  CAN2_RX1_IRQn = 84,
  CAN2_SCE_IRQn = 85,
  USBFS_IRQn = 86,
  ECLIC_NUM_INTERRUPTS
} IRQn_Type;

/**
 * RCC / RCU peripheral struct.
 */
typedef struct
{
  volatile uint32_t CR;
  volatile uint32_t CFGR;
  volatile uint32_t CIR;
  volatile uint32_t APB2RSTR;
  volatile uint32_t APB1RSTR;
  volatile uint32_t AHBENR;
  volatile uint32_t APB2ENR;
  volatile uint32_t APB1ENR;
  volatile uint32_t BDCR;
  /* The following registers differ from STM32F1: */
  volatile uint32_t RSTSCK;
  volatile uint32_t AHBRSTR;
  volatile uint32_t CFGR1;
  volatile uint32_t DSV;
} RCC_TypeDef;

/**
 * GPIO peripheral struct.
 */
typedef struct
{
  volatile uint32_t CRL;
  volatile uint32_t CRH;
  volatile uint32_t IDR;
  volatile uint32_t ODR;
  volatile uint32_t BSRR;
  volatile uint32_t BRR;
  volatile uint32_t LCKR;
} GPIO_TypeDef;

/**
 * AFIO peripheral struct.
 */
typedef struct
{
  volatile uint32_t EVCR;
  volatile uint32_t MAPR;
  volatile uint32_t EXTICR[ 4 ];
  uint32_t RESERVED0;
  volatile uint32_t MAPR2;
} AFIO_TypeDef;

/**
 * DMA peripheral struct (global).
 */
typedef struct
{
  volatile uint32_t ISR;
```

```
  volatile uint32_t IFCR;
} DMA_TypeDef;

/**
 * DMA peripheral struct (per-channel).
 */
typedef struct
{
  volatile uint32_t CCR;
  volatile uint32_t CNDTR;
  volatile uint32_t CPAR;
  volatile uint32_t CMAR;
} DMA_Channel_TypeDef;

/**
 * SPI peripheral struct.
 */
typedef struct
{
  volatile uint32_t CR1;
  volatile uint32_t CR2;
  volatile uint32_t SR;
  volatile uint32_t DR;
  volatile uint32_t CRCPR;
  volatile uint32_t RXCRCR;
  volatile uint32_t TXCRCR;
  volatile uint32_t I2SCFGR;
  volatile uint32_t I2SPR;
} SPI_TypeDef;

/* Global register block address definitions. */
#define RCC           ( ( RCC_TypeDef * )         0x40021000 )
#define GPIOA         ( ( GPIO_TypeDef * )        0x40010800 )
#define GPIOB         ( ( GPIO_TypeDef * )        0x40010C00 )
#define GPIOC         ( ( GPIO_TypeDef * )        0x40011000 )
#define GPIOD         ( ( GPIO_TypeDef * )        0x40011400 )
#define GPIOE         ( ( GPIO_TypeDef * )        0x40011800 )
#define AFIO          ( ( AFIO_TypeDef * )        0x40010000 )
#define DMA1          ( ( DMA_TypeDef * )         0x40020000 )
#define DMA2          ( ( DMA_TypeDef * )         0x40020400 )
#define DMA1_Channel1 ( ( DMA_Channel_TypeDef * ) 0x40020008 )
#define DMA1_Channel2 ( ( DMA_Channel_TypeDef * ) 0x4002001C )
#define DMA1_Channel3 ( ( DMA_Channel_TypeDef * ) 0x40020030 )
#define DMA1_Channel4 ( ( DMA_Channel_TypeDef * ) 0x40020044 )
#define DMA1_Channel5 ( ( DMA_Channel_TypeDef * ) 0x40020058 )
#define DMA1_Channel6 ( ( DMA_Channel_TypeDef * ) 0x4002006C )
#define DMA1_Channel7 ( ( DMA_Channel_TypeDef * ) 0x40020080 )
#define DMA2_Channel1 ( ( DMA_Channel_TypeDef * ) 0x40020408 )
#define DMA2_Channel2 ( ( DMA_Channel_TypeDef * ) 0x4002041C )
#define DMA2_Channel3 ( ( DMA_Channel_TypeDef * ) 0x40020430 )
#define DMA2_Channel4 ( ( DMA_Channel_TypeDef * ) 0x40020444 )
#define DMA2_Channel5 ( ( DMA_Channel_TypeDef * ) 0x40020458 )
#define SPI1          ( ( SPI_TypeDef * )         0x40013000 )
#define SPI2          ( ( SPI_TypeDef * )         0x40003800 )
#define SPI3          ( ( SPI_TypeDef * )         0x40003C00 )

/* RCC register bit definitions. */
/* APB2RSTR */
#define RCC_APB2RSTR_AFIORST_Pos   ( 0U )
#define RCC_APB2RSTR_AFIORST_Msk   ( 0x1UL << RCC_APB2RSTR_AFIORST_Pos )
#define RCC_APB2RSTR_AFIORST       ( RCC_APB2RSTR_AFIORST_Msk )
```

```
#define RCC_APB2RSTR_IOPARST_Pos   ( 2U )
#define RCC_APB2RSTR_IOPARST_Msk   ( 0x1UL << RCC_APB2RSTR_IOPARST_Pos )
#define RCC_APB2RSTR_IOPARST       ( RCC_APB2RSTR_IOPARST_Msk )
#define RCC_APB2RSTR_IOPBRST_Pos   ( 3U )
#define RCC_APB2RSTR_IOPBRST_Msk   ( 0x1UL << RCC_APB2RSTR_IOPBRST_Pos )
#define RCC_APB2RSTR_IOPBRST       ( RCC_APB2RSTR_IOPBRST_Msk )
#define RCC_APB2RSTR_IOPCRST_Pos   ( 4U )
#define RCC_APB2RSTR_IOPCRST_Msk   ( 0x1UL << RCC_APB2RSTR_IOPCRST_Pos )
#define RCC_APB2RSTR_IOPCRST       ( RCC_APB2RSTR_IOPCRST_Msk )
#define RCC_APB2RSTR_IOPDRST_Pos   ( 5U )
#define RCC_APB2RSTR_IOPDRST_Msk   ( 0x1UL << RCC_APB2RSTR_IOPDRST_Pos )
#define RCC_APB2RSTR_IOPDRST       ( RCC_APB2RSTR_IOPDRST_Msk )
#define RCC_APB2RSTR_IOPERST_Pos   ( 6U )
#define RCC_APB2RSTR_IOPERST_Msk   ( 0x1UL << RCC_APB2RSTR_IOPERST_Pos )
#define RCC_APB2RSTR_IOPERST       ( RCC_APB2RSTR_IOPERST_Msk )
#define RCC_APB2RSTR_ADC1RST_Pos   ( 9U )
#define RCC_APB2RSTR_ADC1RST_Msk   ( 0x1UL << RCC_APB2RSTR_ADC1RST_Pos )
#define RCC_APB2RSTR_ADC1RST       ( RCC_APB2RSTR_ADC1RST_Msk )
#define RCC_APB2RSTR_ADC2RST_Pos   ( 10U )
#define RCC_APB2RSTR_ADC2RST_Msk   ( 0x1UL << RCC_APB2RSTR_ADC2RST_Pos )
#define RCC_APB2RSTR_ADC2RST       ( RCC_APB2RSTR_ADC2RST_Msk )
#define RCC_APB2RSTR_TIM1RST_Pos   ( 11U )
#define RCC_APB2RSTR_TIM1RST_Msk   ( 0x1UL << RCC_APB2RSTR_TIM1RST_Pos )
#define RCC_APB2RSTR_TIM1RST       ( RCC_APB2RSTR_TIM1RST_Msk )
#define RCC_APB2RSTR_SPI1RST_Pos   ( 12U )
#define RCC_APB2RSTR_SPI1RST_Msk   ( 0x1UL << RCC_APB2RSTR_SPI1RST_Pos )
#define RCC_APB2RSTR_SPI1RST       ( RCC_APB2RSTR_SPI1RST_Msk )
#define RCC_APB2RSTR_USART1RST_Pos ( 14U )
#define RCC_APB2RSTR_USART1RST_Msk ( 0x1UL << RCC_APB2RSTR_USART1RST_Pos )
#define RCC_APB2RSTR_USART1RST     ( RCC_APB2RSTR_USART1RST_Msk )
/* AHBENR */
#define RCC_AHBENR_DMA1EN_Pos    ( 0U )
#define RCC_AHBENR_DMA1EN_Msk    ( 0x1UL << RCC_AHBENR_DMA1EN_Pos )
#define RCC_AHBENR_DMA1EN        ( RCC_AHBENR_DMA1EN_Msk )
// Note: DMA2 not present on STM32F103 chips.
#define RCC_AHBENR_DMA2EN_Pos    ( 1U )
#define RCC_AHBENR_DMA2EN_Msk    ( 0x1UL << RCC_AHBENR_DMA2EN_Pos )
#define RCC_AHBENR_DMA2EN        ( RCC_AHBENR_DMA2EN_Msk )
#define RCC_AHBENR_SRAMEN_Pos    ( 2U )
#define RCC_AHBENR_SRAMEN_Msk    ( 0x1UL << RCC_AHBENR_SRAMEN_Pos )
#define RCC_AHBENR_SRAMEN        ( RCC_AHBENR_SRAMEN_Msk )
#define RCC_AHBENR_FLITFEN_Pos   ( 4U )
#define RCC_AHBENR_FLITFEN_Msk   ( 0x1UL << RCC_AHBENR_FLITFEN_Pos )
#define RCC_AHBENR_FLITFEN       ( RCC_AHBENR_FLITFEN_Msk )
#define RCC_AHBENR_CRCEN_Pos     ( 6U )
#define RCC_AHBENR_CRCEN_Msk     ( 0x1UL << RCC_AHBENR_CRCENEN_Pos )
#define RCC_AHBENR_CRCEN         ( RCC_AHBENR_CRCENEN_Msk )
// Note: EXMC not present on STM32F103 chips.
#define RCC_AHBENR_EXMCEN_Pos    ( 8U )
#define RCC_AHBENR_EXMCEN_Msk    ( 0x1UL << RCC_AHBENR_EXMCEN_Pos )
#define RCC_AHBENR_EXMCEN        ( RCC_AHBENR_EXMCEN_Msk )
// Note: USB handled differently on STM32F103 chips.
#define RCC_AHBENR_USBFSEN_Pos   ( 12U )
#define RCC_AHBENR_USBFSEN_Msk   ( 0x1UL << RCC_AHBENR_USBFSEN_Pos )
#define RCC_AHBENR_USBFSEN       ( RCC_AHBENR_USBFSEN_Msk )
/* APB2ENR */
#define RCC_APB2ENR_AFIOEN_Pos   ( 0U )
#define RCC_APB2ENR_AFIOEN_Msk   ( 0x1UL << RCC_APB2ENR_AFIOEN_Pos )
#define RCC_APB2ENR_AFIOEN       ( RCC_APB2ENR_AFIOEN_Msk )
#define RCC_APB2ENR_IOPAEN_Pos   ( 2U )
#define RCC_APB2ENR_IOPAEN_Msk   ( 0x1UL << RCC_APB2ENR_IOPAEN_Pos )
```

```
#define RCC_APB2ENR_IOPAEN       ( RCC_APB2ENR_IOPAEN_Msk )
#define RCC_APB2ENR_IOPBEN_Pos   ( 3U )
#define RCC_APB2ENR_IOPBEN_Msk   ( 0x1UL << RCC_APB2ENR_IOPBEN_Pos )
#define RCC_APB2ENR_IOPBEN       ( RCC_APB2ENR_IOPBEN_Msk )
#define RCC_APB2ENR_IOPCEN_Pos   ( 4U )
#define RCC_APB2ENR_IOPCEN_Msk   ( 0x1UL << RCC_APB2ENR_IOPCEN_Pos )
#define RCC_APB2ENR_IOPCEN       ( RCC_APB2ENR_IOPCEN_Msk )
#define RCC_APB2ENR_IOPDEN_Pos   ( 5U )
#define RCC_APB2ENR_IOPDEN_Msk   ( 0x1UL << RCC_APB2ENR_IOPDEN_Pos )
#define RCC_APB2ENR_IOPDEN       ( RCC_APB2ENR_IOPDEN_Msk )
#define RCC_APB2ENR_IOPEEN_Pos   ( 6U )
#define RCC_APB2ENR_IOPEEN_Msk   ( 0x1UL << RCC_APB2ENR_IOPEEN_Pos )
#define RCC_APB2ENR_IOPEEN       ( RCC_APB2ENR_IOPEEN_Msk )
#define RCC_APB2ENR_ADC1EN_Pos   ( 9U )
#define RCC_APB2ENR_ADC1EN_Msk   ( 0x1UL << RCC_APB2ENR_ADC1EN_Pos )
#define RCC_APB2ENR_ADC1EN       ( RCC_APB2ENR_ADC1EN_Msk )
#define RCC_APB2ENR_ADC2EN_Pos   ( 10U )
#define RCC_APB2ENR_ADC2EN_Msk   ( 0x1UL << RCC_APB2ENR_ADC2EN_Pos )
#define RCC_APB2ENR_ADC2EN       ( RCC_APB2ENR_ADC2EN_Msk )
#define RCC_APB2ENR_TIM1EN_Pos   ( 11U )
#define RCC_APB2ENR_TIM1EN_Msk   ( 0x1UL << RCC_APB2ENR_TIM1EN_Pos )
#define RCC_APB2ENR_TIM1EN       ( RCC_APB2ENR_TIM1EN_Msk )
#define RCC_APB2ENR_SPI1EN_Pos   ( 12U )
#define RCC_APB2ENR_SPI1EN_Msk   ( 0x1UL << RCC_APB2ENR_SPI1EN_Pos )
#define RCC_APB2ENR_SPI1EN       ( RCC_APB2ENR_SPI1EN_Msk )
#define RCC_APB2ENR_USART1EN_Pos ( 14U )
#define RCC_APB2ENR_USART1EN_Msk ( 0x1UL << RCC_APB2ENR_USART1EN_Pos )
#define RCC_APB2ENR_USART1EN     ( RCC_APB2ENR_USART1EN_Msk )

/* AFIO register bit definitions. */

/* GPIO register bit definitions. */
/* CRL */
#define GPIO_CRL_MODE_Pos   ( 0U )
#define GPIO_CRL_MODE_Msk   ( 0x33333333UL << GPIO_CRL_MODE_Pos )
#define GPIO_CRL_MODE       ( GPIO_CRL_MODE_Msk )
#define GPIO_CRL_MODE0_Pos  ( 0U )
#define GPIO_CRL_MODE0_Msk  ( 0x3UL << GPIO_CRL_MODE0_Pos )
#define GPIO_CRL_MODE0      ( GPIO_CRL_MODE0_Msk )
#define GPIO_CRL_MODE1_Pos  ( 4U )
#define GPIO_CRL_MODE1_Msk  ( 0x3UL << GPIO_CRL_MODE1_Pos )
#define GPIO_CRL_MODE1      ( GPIO_CRL_MODE1_Msk )
#define GPIO_CRL_MODE2_Pos  ( 8U )
#define GPIO_CRL_MODE2_Msk  ( 0x3UL << GPIO_CRL_MODE2_Pos )
#define GPIO_CRL_MODE2      ( GPIO_CRL_MODE2_Msk )
#define GPIO_CRL_MODE3_Pos  ( 12U )
#define GPIO_CRL_MODE3_Msk  ( 0x3UL << GPIO_CRL_MODE3_Pos )
#define GPIO_CRL_MODE3      ( GPIO_CRL_MODE3_Msk )
#define GPIO_CRL_MODE4_Pos  ( 16U )
#define GPIO_CRL_MODE4_Msk  ( 0x3UL << GPIO_CRL_MODE4_Pos )
#define GPIO_CRL_MODE4      ( GPIO_CRL_MODE4_Msk )
#define GPIO_CRL_MODE5_Pos  ( 20U )
#define GPIO_CRL_MODE5_Msk  ( 0x3UL << GPIO_CRL_MODE5_Pos )
#define GPIO_CRL_MODE5      ( GPIO_CRL_MODE5_Msk )
#define GPIO_CRL_MODE6_Pos  ( 24U )
#define GPIO_CRL_MODE6_Msk  ( 0x3UL << GPIO_CRL_MODE6_Pos )
#define GPIO_CRL_MODE6      ( GPIO_CRL_MODE6_Msk )
#define GPIO_CRL_MODE7_Pos  ( 28U )
#define GPIO_CRL_MODE7_Msk  ( 0x3UL << GPIO_CRL_MODE7_Pos )
#define GPIO_CRL_MODE7      ( GPIO_CRL_MODE7_Msk )
#define GPIO_CRL_CNF_Pos    ( 2U )
```

```
#define GPIO_CRL_CNF_Msk    ( 0x33333333UL << GPIO_CRL_CNF_Pos )
#define GPIO_CRL_CNF        ( GPIO_CRL_CNF_Msk )
#define GPIO_CRL_CNF0_Pos   ( 2U )
#define GPIO_CRL_CNF0_Msk   ( 0x3UL << GPIO_CRL_CNF0_Pos )
#define GPIO_CRL_CNF0       ( GPIO_CRL_CNF0_Msk )
#define GPIO_CRL_CNF1_Pos   ( 6U )
#define GPIO_CRL_CNF1_Msk   ( 0x3UL << GPIO_CRL_CNF1_Pos )
#define GPIO_CRL_CNF1       ( GPIO_CRL_CNF1_Msk )
#define GPIO_CRL_CNF2_Pos   ( 10U )
#define GPIO_CRL_CNF2_Msk   ( 0x3UL << GPIO_CRL_CNF2_Pos )
#define GPIO_CRL_CNF2       ( GPIO_CRL_CNF2_Msk )
#define GPIO_CRL_CNF3_Pos   ( 14U )
#define GPIO_CRL_CNF3_Msk   ( 0x3UL << GPIO_CRL_CNF3_Pos )
#define GPIO_CRL_CNF3       ( GPIO_CRL_CNF3_Msk )
#define GPIO_CRL_CNF4_Pos   ( 18U )
#define GPIO_CRL_CNF4_Msk   ( 0x3UL << GPIO_CRL_CNF4_Pos )
#define GPIO_CRL_CNF4       ( GPIO_CRL_CNF4_Msk )
#define GPIO_CRL_CNF5_Pos   ( 22U )
#define GPIO_CRL_CNF5_Msk   ( 0x3UL << GPIO_CRL_CNF5_Pos )
#define GPIO_CRL_CNF5       ( GPIO_CRL_CNF5_Msk )
#define GPIO_CRL_CNF6_Pos   ( 26U )
#define GPIO_CRL_CNF6_Msk   ( 0x3UL << GPIO_CRL_CNF6_Pos )
#define GPIO_CRL_CNF6       ( GPIO_CRL_CNF6_Msk )
#define GPIO_CRL_CNF7_Pos   ( 30U )
#define GPIO_CRL_CNF7_Msk   ( 0x3UL << GPIO_CRL_CNF7_Pos )
#define GPIO_CRL_CNF7       ( GPIO_CRL_CNF7_Msk )
/* CRH */
#define GPIO_CRH_MODE_Pos   ( 0U )
#define GPIO_CRH_MODE_Msk   ( 0x33333333UL << GPIO_CRH_MODE_Pos )
#define GPIO_CRH_MODE       ( GPIO_CRH_MODE_Msk )
#define GPIO_CRH_MODE8_Pos  ( 0U )
#define GPIO_CRH_MODE8_Msk  ( 0x3UL << GPIO_CRH_MODE8_Pos )
#define GPIO_CRH_MODE8      ( GPIO_CRH_MODE8_Msk )
#define GPIO_CRH_MODE9_Pos  ( 4U )
#define GPIO_CRH_MODE9_Msk  ( 0x3UL << GPIO_CRH_MODE9_Pos )
#define GPIO_CRH_MODE9      ( GPIO_CRH_MODE9_Msk )
#define GPIO_CRH_MODE10_Pos ( 8U )
#define GPIO_CRH_MODE10_Msk ( 0x3UL << GPIO_CRH_MODE10_Pos )
#define GPIO_CRH_MODE10     ( GPIO_CRH_MODE10_Msk )
#define GPIO_CRH_MODE11_Pos ( 12U )
#define GPIO_CRH_MODE11_Msk ( 0x11UL << GPIO_CRH_MODE11_Pos )
#define GPIO_CRH_MODE11     ( GPIO_CRH_MODE11_Msk )
#define GPIO_CRH_MODE12_Pos ( 16U )
#define GPIO_CRH_MODE12_Msk ( 0x3UL << GPIO_CRH_MODE12_Pos )
#define GPIO_CRH_MODE12     ( GPIO_CRH_MODE12_Msk )
#define GPIO_CRH_MODE13_Pos ( 20U )
#define GPIO_CRH_MODE13_Msk ( 0x3UL << GPIO_CRH_MODE13_Pos )
#define GPIO_CRH_MODE13     ( GPIO_CRH_MODE13_Msk )
#define GPIO_CRH_MODE14_Pos ( 24U )
#define GPIO_CRH_MODE14_Msk ( 0x3UL << GPIO_CRH_MODE14_Pos )
#define GPIO_CRH_MODE14     ( GPIO_CRH_MODE14_Msk )
#define GPIO_CRH_MODE15_Pos ( 28U )
#define GPIO_CRH_MODE15_Msk ( 0x3UL << GPIO_CRH_MODE15_Pos )
#define GPIO_CRH_MODE15     ( GPIO_CRH_MODE15_Msk )
#define GPIO_CRH_CNF_Pos    ( 2U )
#define GPIO_CRH_CNF_Msk    ( 0x33333333UL << GPIO_CRH_CNF_Pos )
#define GPIO_CRH_CNF        ( GPIO_CRH_CNF_Msk )
#define GPIO_CRH_CNF8_Pos   ( 2U )
#define GPIO_CRH_CNF8_Msk   ( 0x3UL << GPIO_CRH_CNF8_Pos )
#define GPIO_CRH_CNF8       ( GPIO_CRH_CNF8_Msk )
#define GPIO_CRH_CNF9_Pos   ( 6U )
```

```
#define GPIO_CRH_CNF9_Msk   ( 0x3UL << GPIO_CRH_CNF9_Pos )
#define GPIO_CRH_CNF9       ( GPIO_CRH_CNF9_Msk )
#define GPIO_CRH_CNF10_Pos  ( 10U )
#define GPIO_CRH_CNF10_Msk  ( 0x3UL << GPIO_CRH_CNF10_Pos )
#define GPIO_CRH_CNF10      ( GPIO_CRH_CNF10_Msk )
#define GPIO_CRH_CNF11_Pos  ( 14U )
#define GPIO_CRH_CNF11_Msk  ( 0x11UL << GPIO_CRH_CNF11_Pos )
#define GPIO_CRH_CNF11      ( GPIO_CRH_CNF11_Msk )
#define GPIO_CRH_CNF12_Pos  ( 18U )
#define GPIO_CRH_CNF12_Msk  ( 0x3UL << GPIO_CRH_CNF12_Pos )
#define GPIO_CRH_CNF12      ( GPIO_CRH_CNF12_Msk )
#define GPIO_CRH_CNF13_Pos  ( 22U )
#define GPIO_CRH_CNF13_Msk  ( 0x3UL << GPIO_CRH_CNF13_Pos )
#define GPIO_CRH_CNF13      ( GPIO_CRH_CNF13_Msk )
#define GPIO_CRH_CNF14_Pos  ( 214U )
#define GPIO_CRH_CNF14_Msk  ( 0x3UL << GPIO_CRH_CNF14_Pos )
#define GPIO_CRH_CNF14      ( GPIO_CRH_CNF14_Msk )
#define GPIO_CRH_CNF15_Pos  ( 30U )
#define GPIO_CRH_CNF15_Msk  ( 0x3UL << GPIO_CRH_CNF15_Pos )
#define GPIO_CRH_CNF15      ( GPIO_CRH_CNF15_Msk )

/* Global DMA register bit definitions. */

/* Per-channel DMA register bit definitions. */
/* CCR / CHCTL */
#define DMA_CCR_EN_Pos      ( 0U )
#define DMA_CCR_EN_Msk      ( 0x1UL << DMA_CCR_EN_Pos )
#define DMA_CCR_EN          ( DMA_CCR_EN_Msk )
#define DMA_CCR_TCIE_Pos    ( 1U )
#define DMA_CCR_TCIE_Msk    ( 0x1UL << DMA_CCR_TCIE_Pos )
#define DMA_CCR_TCIE        ( DMA_CCR_TCIE_Msk )
#define DMA_CCR_HTIE_Pos    ( 2U )
#define DMA_CCR_HTIE_Msk    ( 0x1UL << DMA_CCR_HTIE_Pos )
#define DMA_CCR_HTIE        ( DMA_CCR_HTIE_Msk )
#define DMA_CCR_TEIE_Pos    ( 3U )
#define DMA_CCR_TEIE_Msk    ( 0x1UL << DMA_CCR_TEIE_Pos )
#define DMA_CCR_TEIE        ( DMA_CCR_TEIE_Msk )
#define DMA_CCR_DIR_Pos     ( 4U )
#define DMA_CCR_DIR_Msk     ( 0x1UL << DMA_CCR_DIR_Pos )
#define DMA_CCR_DIR         ( DMA_CCR_DIR_Msk )
#define DMA_CCR_CIRC_Pos    ( 5U )
#define DMA_CCR_CIRC_Msk    ( 0x1UL << DMA_CCR_CIRC_Pos )
#define DMA_CCR_CIRC        ( DMA_CCR_CIRC_Msk )
#define DMA_CCR_PINC_Pos    ( 6U )
#define DMA_CCR_PINC_Msk    ( 0x1UL << DMA_CCR_PINC_Pos )
#define DMA_CCR_PINC        ( DMA_CCR_PINC_Msk )
#define DMA_CCR_MINC_Pos    ( 7U )
#define DMA_CCR_MINC_Msk    ( 0x1UL << DMA_CCR_MINC_Pos )
#define DMA_CCR_MINC        ( DMA_CCR_MINC_Msk )
#define DMA_CCR_PSIZE_Pos   ( 8U )
#define DMA_CCR_PSIZE_Msk   ( 0x3UL << DMA_CCR_PSIZE_Pos )
#define DMA_CCR_PSIZE       ( DMA_CCR_PSIZE_Msk )
#define DMA_CCR_MSIZE_Pos   ( 10U )
#define DMA_CCR_MSIZE_Msk   ( 0x3UL << DMA_CCR_MSIZE_Pos )
#define DMA_CCR_MSIZE       ( DMA_CCR_MSIZE_Msk )
#define DMA_CCR_PL_Pos      ( 12U )
#define DMA_CCR_PL_Msk      ( 0x3UL << DMA_CCR_PL_Pos )
#define DMA_CCR_PL          ( DMA_CCR_PL_Msk )
#define DMA_CCR_MEM2MEM_Pos ( 14U )
#define DMA_CCR_MEM2MEM_Msk ( 0x1UL << DMA_CCR_MEM2MEM_Pos )
#define DMA_CCR_MEM2MEM     ( DMA_CCR_MEM2MEM_Msk )
```

```
/* SPI register bit definitions. */
/* CR1 */
#define SPI_CR1_CPHA_Pos      ( 0U )
#define SPI_CR1_CPHA_Msk      ( 0x1UL << SPI_CR1_CPHA_Pos )
#define SPI_CR1_CPHA          ( SPI_CR1_CPHA_Msk )
#define SPI_CR1_CPOL_Pos      ( 1U )
#define SPI_CR1_CPOL_Msk      ( 0x1UL << SPI_CR1_CPOL_Pos )
#define SPI_CR1_CPOL          ( SPI_CR1_CPOL_Msk )
#define SPI_CR1_MSTR_Pos      ( 2U )
#define SPI_CR1_MSTR_Msk      ( 0x1UL << SPI_CR1_MSTR_Pos )
#define SPI_CR1_MSTR          ( SPI_CR1_MSTR_Msk )
#define SPI_CR1_BR_Pos        ( 3U )
#define SPI_CR1_BR_Msk        ( 0x7UL << SPI_CR1_BR_Pos )
#define SPI_CR1_BR            ( SPI_CR1_BR_Msk )
#define SPI_CR1_SPE_Pos       ( 6U )
#define SPI_CR1_SPE_Msk       ( 0x1UL << SPI_CR1_SPE_Pos )
#define SPI_CR1_SPE           ( SPI_CR1_SPE_Msk )
#define SPI_CR1_LSBFIRST_Pos  ( 7U )
#define SPI_CR1_LSBFIRST_Msk  ( 0x1UL << SPI_CR1_LSBFIRST_Pos )
#define SPI_CR1_LSBFIRST      ( SPI_CR1_LSBFIRST_Msk )
#define SPI_CR1_SSI_Pos       ( 8U )
#define SPI_CR1_SSI_Msk       ( 0x1UL << SPI_CR1_SSI_Pos )
#define SPI_CR1_SSI           ( SPI_CR1_SSI_Msk )
#define SPI_CR1_SSM_Pos       ( 9U )
#define SPI_CR1_SSM_Msk       ( 0x1UL << SPI_CR1_SSM_Pos )
#define SPI_CR1_SSM           ( SPI_CR1_SSM_Msk )
#define SPI_CR1_RXONLY_Pos    ( 10U )
#define SPI_CR1_RXONLY_Msk    ( 0x1UL << SPI_CR1_RXONLY_Pos )
#define SPI_CR1_RXONLY        ( SPI_CR1_RXONLY_Msk )
#define SPI_CR1_DFF_Pos       ( 11U )
#define SPI_CR1_DFF_Msk       ( 0x1UL << SPI_CR1_DFF_Pos )
#define SPI_CR1_DFF           ( SPI_CR1_DFF_Msk )
#define SPI_CR1_CRCNEXT_Pos   ( 12U )
#define SPI_CR1_CRCNEXT_Msk   ( 0x1UL << SPI_CR1_CRCNEXT_Pos )
#define SPI_CR1_CRCNEXT       ( SPI_CR1_CRCNEXT_Msk )
#define SPI_CR1_CRCEN_Pos     ( 13U )
#define SPI_CR1_CRCEN_Msk     ( 0x1UL << SPI_CR1_CRCEN_Pos )
#define SPI_CR1_CRCEN         ( SPI_CR1_CRCEN_Msk )
#define SPI_CR1_BIDIOE_Pos    ( 14U )
#define SPI_CR1_BIDIOE_Msk    ( 0x1UL << SPI_CR1_BIDIOE_Pos )
#define SPI_CR1_BIDIOE        ( SPI_CR1_BIDIOE_Msk )
#define SPI_CR1_BIDIMODE_Pos  ( 15U )
#define SPI_CR1_BIDIMODE_Msk  ( 0x1UL << SPI_CR1_BIDIMODE_Pos )
#define SPI_CR1_BIDIMODE      ( SPI_CR1_BIDIMODE_Msk )
/* CR2 */
#define SPI_CR2_RXDMAEN_Pos ( 0U )
#define SPI_CR2_RXDMAEN_Msk ( 0x1UL << SPI_CR2_RXDMAEN_Pos )
#define SPI_CR2_RXDMAEN     ( SPI_CR2_RXDMAEN_Msk )
#define SPI_CR2_TXDMAEN_Pos ( 1U )
#define SPI_CR2_TXDMAEN_Msk ( 0x1UL << SPI_CR2_TXDMAEN_Pos )
#define SPI_CR2_TXDMAEN     ( SPI_CR2_TXDMAEN_Msk )
#define SPI_CR2_SSOE_Pos    ( 2U )
#define SPI_CR2_SSOE_Msk    ( 0x1UL << SPI_CR2_SSOE_Pos )
#define SPI_CR2_SSOE        ( SPI_CR2_SSOE_Msk )
// Note: 'NSS pulse mode' and 'TI mode' bits don't show up in
// STM32F103 reference documents.
#define SPI_CR2_NSSP_Pos    ( 3U )
#define SPI_CR2_NSSP_Msk    ( 0x1UL << SPI_CR2_NSSP_Pos )
#define SPI_CR2_NSSP        ( SPI_CR2_NSSP_Msk )
#define SPI_CR2_TIMOD_Pos   ( 4U )
```

```
#define SPI_CR2_TIMOD_Msk   ( 0x1UL << SPI_CR2_TIMOD_Pos )
#define SPI_CR2_TIMOD       ( SPI_CR2_TIMOD_Msk )
#define SPI_CR2_ERRIE_Pos   ( 5U )
#define SPI_CR2_ERRIE_Msk   ( 0x1UL << SPI_CR2_ERRIE_Pos )
#define SPI_CR2_ERRIE       ( SPI_CR2_ERRIE_Msk )
#define SPI_CR2_RXNEIE_Pos  ( 6U )
#define SPI_CR2_RXNEIE_Msk  ( 0x1UL << SPI_CR2_RXNEIE_Pos )
#define SPI_CR2_RXNEIE      ( SPI_CR2_RXNEIE_Msk )
#define SPI_CR2_TXEIE_Pos   ( 7U )
#define SPI_CR2_TXEIE_Msk   ( 0x1UL << SPI_CR2_TXEIE_Pos )
#define SPI_CR2_TXEIE       ( SPI_CR2_TXEIE_Msk )
/* SR / STAT */
#define SPI_SR_RXNE_Pos   ( 0U )
#define SPI_SR_RXNE_Msk   ( 0x1UL << SPI_SR_RXNE_Pos )
#define SPI_SR_RXNE       ( SPI_SR_RXNE_Msk )
#define SPI_SR_TXE_Pos    ( 1U )
#define SPI_SR_TXE_Msk    ( 0x1UL << SPI_SR_TXE_Pos )
#define SPI_SR_TXE        ( SPI_SR_TXE_Msk )
#define SPI_SR_CHSIDE_Pos ( 2U )
#define SPI_SR_CHSIDE_Msk ( 0x1UL << SPI_SR_CHSIDE_Pos )
#define SPI_SR_CHSIDE     ( SPI_SR_CHSIDE_Msk )
#define SPI_SR_UDR_Pos    ( 3U )
#define SPI_SR_UDR_Msk    ( 0x1UL << SPI_SR_UDR_Pos )
#define SPI_SR_UDR        ( SPI_SR_UDR_Msk )
#define SPI_SR_CRCERR_Pos ( 4U )
#define SPI_SR_CRCERR_Msk ( 0x1UL << SPI_SR_CRCERR_Pos )
#define SPI_SR_CRCERR     ( SPI_SR_CRCERR_Msk )
#define SPI_SR_MODF_Pos   ( 5U )
#define SPI_SR_MODF_Msk   ( 0x1UL << SPI_SR_MODF_Pos )
#define SPI_SR_MODF       ( SPI_SR_MODF_Msk )
#define SPI_SR_OVR_Pos    ( 6U )
#define SPI_SR_OVR_Msk    ( 0x1UL << SPI_SR_OVR_Pos )
#define SPI_SR_OVR        ( SPI_SR_OVR_Msk )
#define SPI_SR_BSY_Pos    ( 7U )
#define SPI_SR_BSY_Msk    ( 0x1UL << SPI_SR_BSY_Pos )
#define SPI_SR_BSY        ( SPI_SR_BSY_Msk )
// Note: "Format error" bit doesn't show up in STM32F1 documentation.
#define SPI_SR_FERR_Pos   ( 8U )
#define SPI_SR_FERR_Msk   ( 0x1UL << SPI_SR_FERR_Pos )
#define SPI_SR_FERR       ( SPI_SR_FERR_Msk )

#endif
```

Longan_nano_3302_Schematic.pdf [49]: